aws

Benutzerhandbuch für Aurora

Amazon Aurora

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Aurora: Benutzerhandbuch für Aurora

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Marken und Handelsmarken von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, die geeignet ist, Kunden irrezuführen oder Amazon in irgendeiner Weise herabzusetzen oder zu diskreditieren. Alle anderen Marken, die nicht im Besitz von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist Aurora? 1
Amazon-RDS-Modell der geteilten Verantwortung 2
Funktionsweise von Amazon Amazon Aurora mit Amazon RDS 2
Aurora-DB-Cluster 4
Aurora-Versionen 6
Unterstützte Datenbank-Engines 6
Angeben einer Aurora-Version 7
Aurora-Versionsverwaltung 7
Aurora-DB-Cluster-Upgrades 13
Support für Aurora-Versionen 16
Regionen und Availability Zones 18
AWS Regionen 19
Availability Zones 28
Lokale Zeitzone für -DB-Cluster 29
Unterstützte Aurora-Funktionen von Region und Engine 36
Tabellenkonventionen 37
Blau/Grün-Bereitstellungen 37
Aurora-Cluster-Konfigurationen 38
Datenbankaktivitätsstreams 39
Exportieren von Cluster-Daten nach Amazon S3 49
Exportieren von Snapshot-Daten nach Amazon S3 51
Globale Aurora-Datenbanken 52
IAM-Datenbankauthentifizierung 64
Kerberos-Authentifizierung 65
Aurora Machine Learning 76
Performance Insights 87
Zero-ETL-Integrationen 97
RDS-Proxy 105
Integration von Secrets Manager 118
Aurora Serverless v2 118
Aurora Serverless v1 125
RDS-Daten-API 129
Zero-Downtime-Patching (ZDP) (Patchen ohne Ausfallzeiten) 139
Aurora PostgreSQL Limitless Database 140

Engine-native Funktionen ... 140
Endpunktverbindungen ... 141
Typen von Aurora-Endpunkten ... 142
Anzeigen von Endpunkten ... 143
Endpunkte und hohe Verfügbarkeit ... 144
Cluster-Endpunkte ... 145
Reader-Endpunkte ... 146
Instance-Endpunkte ... 147
Benutzerdefinierte Endpunkte ... 149
DB-Instance-Klassen ... 170
DB-Instance-Klassenarten ... 170
Unterstützte DB-Engines ... 174
Bestimmung des Support von DB-Instance-Klassen in AWS-Regionen ... 185
Hardwarespezifikationen ... 190
Speicher ... 202
Übersicht über Aurora-Speicher ... 203
Inhalt des Cluster-Volumes ... 203
Speicherkonfigurationen von Aurora-Clustern ... 203
Wie sich die Größe des Speichers ändert ... 204
Datenabrechnung ... 206
Zuverlässigkeit ... 206
Automatische Speicherplatzreparatur ... 207
Überlebensfähiger Seiten-Cache ... 207
Wiederherstellung nach ungeplanten Neustarts ... 208
Aurora-Sicherheit ... 209
Verwenden von SSL mit Aurora-DB-Clustern ... 211
Hohe Verfügbarkeit ... 211
Hohe Verfügbarkeit für Aurora-Daten ... 211
Hohe Verfügbarkeit für Aurora-DB-Instances ... 212
Hohe Verfügbarkeit über AWS-Regionen hinweg mit globalen Aurora-Datenbanken ... 213
Fehlertoleranz ... 213
Hochverfügbarkeit mit Amazon-RDS-Proxy ... 215
Replikation ... 215
Aurora-Replikate ... 216
Aurora MySQL ... 218
Aurora PostgreSQL ... 219

Abrechnung von DB-Instances für Aurora ... 220
On-Demand-DB-Instances ... 222
Reservierte DB-Instances ... 224
Einrichten Ihrer Umgebung ... 240
Melden Sie sich an für ein AWS-Konto ... 240
Erstellen eines Benutzers mit Administratorzugriff ... 241
Erteilen programmgesteuerten Zugriffs ... 242
Ermitteln der Anforderungen ... 244
Zugriff auf den DB-Cluster bereitstellen ... 246
Erste Schritte ... 249
Erstellen und Verbinden mit einem DB-Cluster von Aurora MySQL ... 249
Voraussetzungen ... 251
Schritt 1: Eine EC2 Instanz erstellen ... 251
Schritt 2:Erstellen eines DB-Clusters von Aurora MySQL ... 257
(Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora MySQL-Cluster mit CloudFormation ... 263
Schritt 3: Herstellen einer Verbindung mit einem DB-Cluster von Aurora MySQL ... 266
Schritt 4: Löschen Sie die EC2 Instance und den DB-Cluster ... 269
(Optional) Löschen Sie die EC2 Instance und den DB-Cluster, die mit erstellt wurden CloudFormation ... 270
(Optional) Verbinden Sie Ihren DB-Cluster mit einer Lambda-Funktion ... 271
Erstellen eines DB-Clusters von Aurora PostgreSQL und Herstellen einer Verbindung ... 271
Voraussetzungen ... 273
Schritt 1: Eine EC2 Instanz erstellen ... 273
Schritt 2: Erstellen eines DB-Clusters von Aurora PostgreSQL ... 279
(Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora PostgreSQL-Cluster mit CloudFormation ... 284
Schritt 3: Herstellen einer Verbindung mit einem DB-Cluster von Aurora PostgreSQL ... 287
Schritt 4: Löschen Sie die EC2 Instance und den DB-Cluster ... 290
(Optional) Löschen Sie die EC2 Instance und den DB-Cluster, die mit erstellt wurden CloudFormation ... 291
(Optional) Verbinden Sie Ihren DB-Cluster mit einer Lambda-Funktion ... 291
Tutorial: Erstellen eines Webservers und eines Amazon Aurora-DB-Clusters ... 292
Starten Sie eine EC2-Instance, um eine Verbindung mit Ihrem DB-Cluster herzustellen. ... 294
Erstellen eines DB-Clusters ... 300
Erstellen eines Webservers ... 311

Tutorials und Beispiel-Code ... 324
Tutorials in diesem Handbuch ... 324
Tutorials in anderen AWS Leitfäden ... 325
Tutorials und Beispielcode in GitHub ... 326
AWS Datenbank-Kochbuch ... 327
AWS Werkstatt — Amazon Aurora PostgreSQL ... 328
AWS Werkstatt — Amazon Aurora MySQL ... 330
Arbeitet mit AWS SDKs ... 331
Programmatischer Zugriff auf Amazon Aurora ... 333
Console-to-Code ... 334
Einrichten eines Aurora-DB-Clusters ... 335
Erstellen eines DB-Clusters ... 336
Voraussetzungen ... 337
Erstellen eines DB-Clusters ... 344
Verfügbare Einstellungen ... 355
Einstellungen, die nicht für Aurora-DB-Cluster gelten ... 378
Einstellungen, die nicht für Aurora-DB-Instances gelten ... 379
Ressourcen erstellen mit AWS CloudFormation ... 382
Aurora und CloudFormation-Vorlagen ... 382
Weitere Informationen zu CloudFormation ... 382
Herstellen einer Verbindung mit einem DB-Cluster ... 383
Herstellen einer Verbindung mit Aurora-DB-Clustern mithilfe der AWS-Treiber ... 384
Herstellen einer Verbindung mit Aurora MySQL ... 386
Herstellen einer Verbindung mit Aurora PostgreSQL ... 393
Fehlerbehebung bei Verbindungen ... 396
Parametergruppen ... 397
Übersicht über Parametergruppen ... 397
DB-Cluster-Parametergruppen ... 402
DB-Parametergruppen ... 420
Vergleichen von DB-Parametergruppen ... 437
Festlegen von DB-Parametern ... 438
Migrieren von Daten zu einem DB-Cluster ... 444
Aurora MySQL ... 444
Aurora PostgreSQL ... 444
Erstellen eines ElastiCache-Caches aus Amazon RDS ... 445

Überblick über die ElastiCache-Cache-Erstellung mit den Einstellungen des Aurora-DB-Clusters ... 445
Erstellen eines ElastiCache-Caches mit den Einstellungen eines Aurora-DB-Clusters ... 446
Automatische Migration von EC2-Datenbanken ... 450
Übersicht ... 450
Voraussetzungen ... 451
Einschränkungen ... 452
Erstellen von IAM-Ressourcen ... 453
Einrichten der Datenmigration ... 461
Verwalten von Migrationen ... 463
Überwachen ... 466
Tutorial: Erstellen einer eines DB-Clusters für MySQL mit einem benutzerdefinierten Parameter ... 468
Voraussetzungen ... 469
Erstellen einer benutzerdefinierten Parametergruppe ... 469
Ändern des Parameterwerts ... 470
Erstellen einer eines DB-Clusters ... 470
Verwalten eines Aurora-DB-Clusters ... 472
Stoppen und Starten eines Clusters ... 473
Übersicht über das Stoppen und Starten eines Clusters ... 473
Einschränkungen ... 475
Stoppen eines DB-Clusters ... 475
Während des Stopps eines DB-Clusters ... 476
Starten eines DB-Clusters ... 477
Verbinden einer EC2-Instance ... 479
Übersicht ... 480
Verbinden einer EC2-Instance ... 485
Anzeigen verbundener Rechenressourcen ... 488
Herstellen einer Verbindung mit einer DB-Instance, auf der eine bestimmte DB-Engine ausführt wird ... 489
Verbinden einer Lambda-Funktion ... 490
Übersicht ... 492
Verbinden einer Lambda-Funktion ... 503
Anzeigen verbundener Rechenressourcen ... 505
Ändern eines Aurora-DB-Clusters ... 507
Ändern des DB-Clusters über die Konsole, die CLI und die API ... 507

Ändern einer DB-Instance in einem DB-Cluster ... 510
Ändern des Master-Benutzerpassworts ... 513
Verfügbare Einstellungen ... 515
Einstellungen, die nicht für Aurora-DB-Cluster gelten ... 552
Einstellungen, die nicht für Aurora-DB-Instances gelten ... 553
Hinzufügen von Aurora-Replicas ... 555
Amazon EC2 Auto Scaling mit Aurora Replicas ... 562
Hinzufügen einer Auto-Scaling-Richtlinie ... 568
Bearbeiten einer Auto-Scaling-Richtlinie ... 581
Löschen einer Auto-Scaling-Richtlinie ... 584
Verwalten von Performance und Skalierung ... 587
Speicherskalierung ... 587
Skalierung von Instances ... 594
Skalierung von Lesevorgängen ... 595
Verwalten von Verbindungen ... 595
Verwalten von Abfrageausführungsplänen ... 596
Klonen eines Volumes für einen Aurora-DB-Cluster ... 597
Übersicht über das Aurora-Klonen ... 597
Einschränkungen für das Aurora-Klonen ... 598
Funktionsweise des Klonens von Aurora ... 599
Erstellen eines Aurora-Klons ... 603
VPC-übergreifendes Klonen ... 614
Kontoübergreifendes Klonen ... 634
Integration mit AWS Diensten ... 653
Aurora MySQL ... 653
Aurora PostgreSQL ... 653
Warten eines Aurora-DB-Clusters ... 655
Übersicht über Wartungsupdates für DB-Cluster ... 655
Anzeigen ausstehender Wartung ... 657
Auswahl der Häufigkeit der Aurora MySQL-Wartungsupdates ... 659
Wartungsfenster ... 660
Anwenden von Updates ... 666
Automatische Nebenversions-Upgrades für Aurora-DB-Cluster ... 668
Betriebssystemupdates ... 673
AWS Organizations Rollout aktualisieren ... 682
Neustart eines Aurora DB-Clusters oder einer Instance ... 686

Neustarten einer DB-Instance in einem Aurora Cluster ... 687
Neustarten eines Aurora-Clusters mit Leseverfügbarkeit ... 688
Neustart eines Aurora-Clusters ohne Leseverfügbarkeit ... 690
Überprüfung der Betriebszeit für Aurora Cluster und Instances ... 691
Beispiele für Aurora Neustartvorgänge ... 694
Failover eines Aurora-DB-Clusters ... 711
Löschen von Aurora-Clustern und -Instances ... 713
Löschen eines Aurora-Dönnen einen Cluster nicht löschen,B-Clusters ... 713
Löschschutz für Aurora-DB-Cluster ... 722
Löschen eines angehaltenen Aurora-Clusters ... 722
Löschen von Aurora MySQL-Clustern, die Read Replicas sind ... 722
Der letzte Snapshot beim Löschen eines Clusters ... 723
Löschen einer DB-Instance aus einem Aurora-DB-Cluster ... 723
Taggen von Aurora- und RDS-Ressourcen ... 726
Warum RDS-Tags verwenden? ... 727
So funktionieren RDS-Tags ... 728
Bewährte Methoden ... 731
Kopieren von Tags in DB-Cluster-Snapshots ... 732
Hinzufügen und Löschen von Tags in Amazon RDS ... 733
Tutorial: Verwenden Sie Tags, um festzulegen, welcher Aurora-DB-Cluster stoppt. ... 738
ARNs in Amazon RDS ... 742
Erstellen eines ARN ... 742
Abrufen eines vorhandenen ARN ... 750
Aurora-Updates ... 754
Identifizieren Ihrer Amazon-Aurora-Version ... 754
RDS Extended Support ... 756
RDS Extended Support im Überblick ... 757
Gebühren für RDS Extended Support ... 758
Vermeiden von Gebühren für RDS Extended Support ... 759
Versionen mit RDS Extended Support ... 759
Aufgaben hinsichtlich RDS Extended Support ... 759
Aufgaben von Amazon Aurora ... 760
Ihre Aufgaben ... 760
Erstellen eines Aurora-DB-Clusters oder eines globalen Clusters ... 761
Verhalten von RDS Extended Support ... 761
Überlegungen zu RDS Extended Support ... 762

Erstellen Sie einen Aurora-DB-Cluster oder einen globalen Cluster mit RDS Extended Support ... 763
Anzeigen der Registrierung in RDS Extended Support ... 764
Anzeigen der Support-Daten ... 767
Wiederherstellen eines Aurora-DB-Clusters oder eines globalen Clusters ... 769
Verhalten von RDS Extended Support ... 770
Überlegungen zu RDS Extended Support ... 770
Stellen Sie einen Aurora-DB-Cluster oder einen globalen Cluster mit RDS Extended Support wieder her ... 771
Verwendung von Blue/Green Bereitstellungen für Datenbankaktualisierungen ... 773
Übersicht über Blau/Grün-Bereitstellungen ... 774
Verfügbarkeit von Regionen und Versionen ... 775
Vorteile ... 775
Workflow ... 775
Autorisieren des Zugriffs ... 782
Einschränkungen und Überlegungen ... 783
Bewährte Methoden ... 793
Erstellen einer Blau/Grün-Bereitstellung ... 798
Vorbereiten einer Blau-Grün-Bereitstellung ... 798
Angeben von Änderungen ... 801
Erstellen einer Blau/Grün-Bereitstellung ... 801
Verfügbare Einstellungen ... 804
Anzeigen einer Blau/Grün-Bereitstellung ... 806
Umstellen einer Blau/Grün-Bereitstellung ... 810
Umstellungs-Timeout ... 811
Integrationsschutz der Umstellung ... 811
Umstellungsaktionen ... 813
Bewährte Methoden für die Umstellung ... 814
CloudWatch Überprüfen der Metriken vor dem Switchover ... 815
Überwachung der Replikatverzögerung vor der Umstellung ... 815
Eine blue/green Bereitstellung umschalten ... 816
Nach der Umstellung ... 819
Löschen einer Blau/Grün-Bereitstellung ... 821
Sichern und Wiederherstellen eines Aurora-DB-Clusters ... 825
Übersicht über das Sichern und Wiederherstellen ... 826
Backups ... 826

Backup window ... 827
Wiederherstellen von Daten ... 830
Klonen von Datenbanken ... 831
Backtrack ... 832
Aufbewahren automatisierter Backups ... 833
Aufbewahrungszeitraum ... 833
Aufbewahrungskosten ... 834
Verhindern des Löschens von automatisierten Backups ... 834
Einschränkungen ... 834
Anzeigen beibehaltener automatisierter Backups ... 835
Löschen aufbewahrter automatisierter Backups ... 836
Backup-Speicher ... 838
Automatischer Backup-Speicher ... 838
Snapshot-Speicher ... 839
CloudWatch-Metriken für Backup-Speicher ... 839
Berechnung der Nutzung des Backup-Speichers ... 840
Häufig gestellte Fragen ... 842
Erstellen eines DB-Cluster-Snapshots ... 845
Feststellen, ob der Snapshot verfügbar ist ... 847
Wiederherstellen aus einem DB-Cluster-Snapshot ... 848
Parametergruppen ... 849
Sicherheitsgruppen ... 849
Überlegungen zu Aurora ... 850
Wiederherstellung aus einem Snapshot ... 850
Kopieren eines DB-Cluster-Snapshots ... 853
Kopieren eines DB-Cluster-Snapshots mithilfe der AWS-Managementkonsole ... 853
Einschränkungen ... 855
Überlegungen ... 856
Kopieren eines unverschlüsselten DB-Cluster-Snapshots in ein anderes Konto ... 859
Freigeben eines verschlüsselten DB-Cluster-Snapshots ... 861
Kopieren eines DB-Cluster-Snapshots in ein anderes Konto ... 865
Freigeben eines DB-Cluster-Snapshots ... 870
Freigeben eines Snapshots ... 871
Freigeben öffentlicher Snapshots ... 875
Freigeben verschlüsselter Snapshots ... 877
Beenden der Snapshot-Freigabe ... 881

Exportieren von DB-Cluster-Daten nach Amazon S3 ... 883
Überlegungen ... 885
Einrichten des Zugriffs auf einen S3-Bucket ... 887
Erstellen von DB-Cluster-Exportaufgaben ... 891
Überwachen von DB-Cluster-Exporten ... 895
Abbrechen eines DB-Cluster-Exports ... 897
Fehlerbehebung ... 898
Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3 ... 902
Überlegungen ... 904
Einrichten des Zugriffs auf einen S3-Bucket ... 917
Erstellen von Snapshot-Exportaufgaben ... 922
Überwachung von Snapshot-Exporten ... 926
Abbrechen eines Snapshot-Exports ... 929
Exportleistung in Aurora MySQL ... 930
Fehlerbehebung ... 931
Point-in-time Erholung ... 934
Wiederherstellen eines DB-Clusters zu einem bestimmten Zeitpunkt ... 935
Zeitpunktbezogene Wiederherstellung aus einem beibehaltenen, automatisierten Backup ... 938
Zeitpunktbezogene Wiederherstellung mit AWS Backup ... 941
Löschen eines DB-Cluster-Snapshots ... 948
Löschen eines DB-Cluster-Snapshots ... 948
Tutorial: Wiederherstellen eines DB-Clusters aus einem Snapshot ... 950
Wiederherstellen eines DB-Clusters über die Konsole ... 950
Wiederherstellen eines DB-Clusters mithilfe der AWS CLI ... 955
Überwachung von Metriken in einem Aurora-DB-Cluster ... 962
Überwachungsplan ... 962
Leistungsbasislinie ... 963
Richtlinien zur Leistung ... 963
Überwachungstools ... 965
Automatisierte Überwachungstools ... 965
Manuelle Überwachungstools ... 967
Anzeigen des Status der Cluster ... 969
Anzeigen eines DB-Clusters ... 970
Anzeigen des DB-Clusterstatus ... 976
Anzeigen von DB-Instance-Status in einem Aurora-Cluster ... 980
Empfehlungen von Amazon Aurora ... 987

Anzeigeempfehlungen 989
Anwenden von Empfehlungen 997
Verwerfen von Empfehlungen 1003
Ändern abgelehnter Empfehlungen in aktive Empfehlungen 1005
Referenz zu Empfehlungen 1007
Anzeigen von Metriken in der Amazon-RDS-Konsole 1030
Anzeigen des Performance-Insights-Dashboards 1032
Auswählen der neuen Überwachungsansicht auf der Registerkarte „Überwachung“ 1033
Auswählen der neuen Überwachungsansicht auf der Performance-Insights-Seite 1035
Erstellen eines benutzerdefinierten Dashboards 1037
Auswählen des vorkonfigurierten Dashboards 1040
Überwachen von Aurora mit CloudWatch 1042
Übersicht über Amazon Aurora und Amazon CloudWatch 1043
Anzeigen von CloudWatch-Metriken 1045
Exportieren von Performance-Insights-Metriken nach CloudWatch 1050
Erstellen von CloudWatch-Alarmen 1056
Überwachen mit Database Insights 1058
Preisgestaltung 1058
Unterstützung für Engine-, Regions- und Instance-Klassen 1058
Einschalten des erweiterten Modus 1062
Aktivieren des Standardmodus 1067
Überwachen langsamer Abfragen 1072
Überlegungen 1074
Überwachung von DB-Last mit Performance Insights 1075
Überblick über Performance Insights 1076
Aktivieren und Deaktivieren von Performance Insights 1087
Leistungsschema für Aurora MySQL 1094
Performance-Insights-Richtlinien 1099
Analyse der Metriken mit dem Performance-Insights-Dashboard 1114
Anzeigen proaktiver Empfehlungen von Performance Insights 1149
Abrufen von Metriken mit der Performance-Insights-API 1152
Protokollieren von Performance-Insights-An AWS CloudTrail 1177
VPC-Endpunkte (AWS PrivateLink) 1180
Analyse der Leistung mit DevOps Guru for RDS 1184
Vorteile von DevOps Guru für RDS 1185
Wie funktioniert DevOps Guru for RDS 1186

DevOpsGuru für RDS einrichten 1187
Überwachen vom Betriebssystem mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung) 1196
Überblick über „Enhanced Monitoring“ (Erweiterte Überwachung) 1196
Einrichten und Aktivieren von „Enhanced Monitoring“ (Erweiterte Überwachung) 1198
Anzeigen von Betriebssystem-Metriken in der RDS-Konsole 1206
Anzeigen von Betriebssystemmetriken mit CloudWatch Logs 1208
Aurora-Referenz für Metriken 1210
CloudWatch Metriken für Aurora 1210
CloudWatch Abmessungen für Aurora 1254
Verfügbarkeit von Aurora Metriken in der Amazon RDS Konsole 1254
CloudWatch-Metriken für Performance Insights 1258
Zählermetriken für Performance Insights 1261
SQL-Statistiken für Performance Insights 1297
Betriebssystemmetriken in „Erweiterte Überwachung“ 1307
Überwachen von Ereignissen, Protokollen und Datenbankaktivitäts-Streams 1316
Anzeigen von Protokollen, Ereignissen und Streams in der Amazon-RDS-Konsole 1317
Überwachung von Aurora-Ereignissen 1322
Überblick über Ereignisse für Aurora 1322
Anzeigen von Amazon-RDS-Ereignissen 1324
Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen 1328
Erstellen einer Regel, die bei einem Amazon Aurora-Ereignis ausgelöst wird 1355
Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora 1360
Überwachen von Aurora-Protokollen 1397
Anzeigen und Auflisten von Datenbank-Protokolldateien 1397
Herunterladen einer Datenbank-Protokolldatei 1399
Überwachen einer Datenbank-Protokolldatei 1400
Veröffentlichen auf CloudWatch Logs 1402
Lesen der Protokolldateiinhalte mit REST 1405
MySQL-Datenbank-Protokolldateien 1407
PostgreSQL-Datenbankprotokolldateien 1418
Überwachung von Aurora-API-Aufrufen in CloudTrail 1430
Integration von CloudTrail in Amazon Aurora 1430
Amazon Aurora-Protokolldateieinträge 1431
Überwachung von Aurora mithilfe von Datenbankaktivitätsstreams 1436
Übersicht 1436

Aurora MySQL-Netzwerkvoraussetzungen ... 1440
Starten eines Datenbankaktivitäts-Streams ... 1442
Abrufen des Status eines Aktivitätsstreams ... 1446
Stoppen eines Datenbankaktivitäts-Streams ... 1447
Überwachen von Aktivitäts-Streams ... 1449
IAM-Richtlinien-Beispiele für Aktivitäts-Streams ... 1488
Überwachen von Bedrohungen mit GuardDuty RDS Protection ... 1492
Arbeiten mit Aurora MySQL ... 1494
Übersicht über Aurora MySQL ... 1495
Amazon Aurora MySQL-Leistungserweiterungen ... 1495
Aurora MySQL und raumbezogene Daten ... 1496
Aurora mit MySQL Version 3 ist kompatibel mit MySQL 8.0 ... 1498
Aurora-MySQL-Version 2, kompatibel mit MySQL 5.7 ... 1533
Sicherheit in Aurora MySQL ... 1536
Berechtigungen von Masterbenutzerkonten mit Aurora MySQL. ... 1538
TLS-Verbindungen ... 1538
Aktualisieren von Anwendungen für neue TLS-Zertifikate ... 1547
Ermitteln, ob Anwendungen Verbindungen mit Ihrem DB-Cluster von Aurora MySQL mithilfe von TLS herstellen ... 1548
Ermitteln, ob ein Client zum Herstellen von Verbindungen Zertifikatverifizierungen erfordert ... 1548
Aktualisieren des Trust Stores Ihrer Anwendung ... 1550
Java-Beispielcode für die Herstellung von TLS-Verbindungen ... 1551
Verwenden der Kerberos-Authentifizierung für Aurora MySQL ... 1553
Übersicht über die Kerberos-Authentifizierung für Aurora MySQL ... 1554
Einschränkungen ... 1556
Einrichten der Kerberos-Authentifizierung für Aurora MySQL ... 1557
Herstellen einer Verbindung mit Aurora MySQL mit Kerberos-Authentifizierung ... 1568
Verwalten eines DB-Clusters in einer Domain ... 1572
Migrieren von Daten zu Aurora MySQL ... 1574
Migrieren von einer externen MySQL-Datenbank zu Aurora MySQL ... 1579
Migrieren von einer MySQL-DB-Instance nach Aurora MySQL ... 1608
Verwalten von Aurora MySQL ... 1635
Verwalten von Performance und Skalierung für Amazon Aurora MySQL ... 1635
Rückverfolgen eines DB-Clusters ... 1647
Testen von Amazon Aurora MySQL unter Verwendung von Fehlersimulationsabfragen ... 1671

Ändern von Tabellen in Amazon Aurora mithilfe von Fast DDL 1675
Anzeigen des Volume-Status für einen Aurora-DB-Cluster 1682
Optimieren von Aurora MySQL 1684
Grundlegende Konzepte für Aurora-MySQL-Optimierung 1684
Optimieren von Aurora MySQL mit Warteereignissen 1688
Optimierung von Aurora MySQL mit Thread-Status 1747
Optimierung von Aurora MySQL mit proaktiven Einblicken von Amazon DevOps Guru 1755
Parallel Query für Aurora MySQL 1762
Übersicht über Parallel Query 1762
Erstellen eines Parallel Query-Clusters 1767
Aktivieren und Deaktivieren von Parallel Query 1771
Optimieren von Parallel Query 1775
Überprüfen der Parallel Query-Nutzung 1780
Überwachen von Parallel Query 1784
SQL-Konstrukte für Parallel Query 1792
Advanced Auditing mit Aurora MySQL 1815
Aktivieren von erweitertem Auditing 1815
Anzeigen von Audit-Protokollen 1819
Details in Prüfprotokollen 1819
Replikation mit Aurora MySQL 1822
Aurora-Replikate 1822
Replikationsoptionen 1824
Replikationsleistung 1825
Replikationsfilter 1826
Überwachung einer -Replikation 1833
Regionsübergreifende Replikation 1834
Binäre Protokollreplikation (Binlog) 1852
GTID-basierte Replikation 1898
Lokale Schreibweiterleitung 1905
Aktivieren der lokalen Schreibweiterleitung 1906
Überprüfen, ob die Schreibweiterleitung für einen DB-Cluster aktiviert ist 1908
Anwendung und SQL-Kompatibilität mit Schreibweiterleitung 1909
Isolationsstufen für die Schreibweiterleitung 1911
Lesekonsistenz 1911
Ausführen von Multipart-Anweisungen mit Schreibweiterleitung 1916
Transaktionen mit Schreibweiterleitung 1916

Konfigurationsparameter für die Schreibweiterleitung ... 1917
Metriken für die Schreibweiterleitung ... 1918
Identifizieren weitergeleiteter Transaktionen und Abfragen ... 1923
Integrieren von Aurora MySQL mit AWS-Services ... 1925
Autorisieren von Aurora MySQL zum Zugriff auf andere AWS-Services ... 1925
Laden von Daten aus Textdateien in Amazon S3 ... 1946
Speichern von Daten in Textdateien in Amazon S3 ... 1961
Aufrufen einer Lambda-Funktion aus Aurora MySQL ... 1973
Veröffentlichen von Aurora MySQL-Protokollen in CloudWatch Logs ... 1985
Aurora MySQL-Labor-Modus ... 1992
Funktionen des Aurora-Labor-Modus ... 1992
Bewährte Methoden mit Aurora MySQL ... 1994
Feststellen, mit welcher DB-Instance Sie verbunden sind ... 1995
Bewährte Methoden für die Leistung und Skalierung ... 1995
Bewährte Methoden für die Hochverfügbarkeit ... 2006
Empfehlungen für MySQL-Funktionen ... 2008
Bewerten der DB-Instance-Nutzung für Aurora MySQL mit Amazon-CloudWatch-Metriken ... 2016
Fehlerbehebung bei der Aurora-MySQL-Leistung ... 2018
AWS-Überwachungsoptionen ... 2018
Die häufigsten Gründe für DB-Leistungsprobleme ... 2019
Beheben von Workload-Problemen ... 2020
Protokollieren bei Aurora MySQL ... 2051
Beheben von Problemen mit Datenbankverbindungen ... 2054
Behebung von Leistungsproblemen bei Abfragen ... 2072
Aurora MySQL-Referenz ... 2077
Konfigurationsparameter ... 2077
Globale Statusvariablen ... 2153
Warteereignisse ... 2172
Thread-Zustände ... 2178
Isolierungsstufen ... 2183
Hinweise ... 2190
Referenz für gespeicherte Prozeduren ... 2195
information_schema-Tabellen ... 2245
Aurora MySQL-Updates ... 2254
Überprüfen von Versionsnummern ... 2255

Langzeit-Support- und Betaversionen 2257
Vorbereitung auf das Lebenszyklusende von Aurora MySQL Version 2 2259
Vorbereitung auf das Lebenszyklusende der Aurora MySQL Version 1 2265
Upgrade von Amazon Aurora MySQL-DB-Clustern 2268
Datenbank-Engine-Updates und -Korrekturen für Amazon Aurora MySQL 2407
Arbeiten mit Aurora PostgreSQL 2408
Die Datenbank-Vorschauumgebung 2410
Unterstützte Typen von DB-Instance-Klassen 2410
Nicht in der Vorschauumgebung unterstützte Features 2411
Erstellen eines neuen DB-Clusters in der Vorschauumgebung 2412
PostgreSQL Version 17 in der Datenbank-Vorschauumgebung 2414
Sicherheit in Aurora PostgreSQL 2415
Grundlegendes zu PostgreSQL-Rollen und -Berechtigungen 2416
Sicherung von Aurora-PostgreSQL-Daten mit SSL/TLS 2433
Dynamische Maskierung von Aurora PostgreSQL 2447
Aktualisieren von Anwendungen für neue SSL/TLS-Zertifikate 2484
Ermitteln, ob Anwendungen Verbindungen mit Aurora-PostgreSQL-DB-Clustern mithilfe von SSL herstellen 2485
Ermitteln, ob ein Client zum Herstellen von Verbindungen Zertifikatverifizierungen erfordert 2486
Aktualisieren des Trust Stores Ihrer Anwendung 2487
Verwenden von SSL/TLS-Verbindungen für verschiedene Arten von Anwendungen 2487
Verwenden der Kerberos-Authentifizierung 2489
Verfügbarkeit von Regionen und Versionen 2490
Übersicht über die Kerberos-Authentifizierung 2490
Einrichtung 2492
Verwalten eines DB-Clusters von Aurora PostgreSQL in einer Active Directory-Domain 2508
Herstellen einer Verbindung mithilfe der Kerberos-Authentifizierung 2509
Verwenden von AD-Sicherheitsgruppen für die Aurora-PostgreSQL-Zugriffskontrolle 2513
Migrieren von Daten nach Aurora PostgreSQL 2526
Migrieren einer RDS-for-PostgreSQL-DB-Instance unter Verwendung eines Snapshots 2527
Migrieren einer RDS-for-PostgreSQL-DB-Instance unter Verwendung eines Aurora-Lesereplikats 2534
Optimieren der Abfrageleistung in Aurora PostgreSQL 2549
Verbesserung der Abfrageleistung mit Aurora-optimierten Lesevorgängen 2550
Optimieren von korrelierten Unterabfragen in Aurora PostgreSQL 2557

Verbessern der Abfrageleistung mit Adaptive Join ... 2564
Arbeiten mit Tabellen ohne Protokollierung in Aurora PostgreSQL ... 2566
Erstellen von Tabellen ohne Protokollierung ... 2567
Handhaben von Tabellen ohne Protokollierung während der Migration ... 2567
Konvertieren von Tabellen ohne Protokollierung in Tabellen mit Protokollierung ... 2568
Tabellen ohne Protokollierung und logische Replikation ... 2569
Arbeiten mit der PostgreSQL-Selbstbereinigung ... 2569
Zuweisen von Arbeitsspeicher für die Selbstbereinigung ... 2570
Verringern der Wahrscheinlichkeit von Transaktions-ID-Wraparounds ... 2572
Ermittlung, ob die Tabellen in Ihrer Datenbank bereinigt werden müssen ... 2573
Ermittlung, für welche Tabellen derzeit eine Selbstbereinigung nötig ist ... 2575
Ermittlung, ob die Selbstbereinigung derzeit ausgeführt wird und wie lange sie dauert ... 2576
Ausführen einer manuellen Bereinigungseinfrierung ... 2577
Neuindizierung einer Tabelle während der Ausführung einer Selbstbereinigung ... 2580
Verwalten der automatischen Bereinigung mit großen Indizes ... 2581
Weitere Parameter, die sich auf die Selbstbereinigung auswirken ... 2585
Festlegen von Selbstbereinigungsparametern auf Tabellenebene ... 2586
Protokollieren von Selbstbereinigung- und Bereinigungsaktivitäten ... 2586
Verstehen des Verhaltens der Selbstbereinigung bei ungültigen Datenbanken ... 2588
Identifizieren von Bereinigungsblockern ... 2590
Verwaltung von TOAST-OID-Konflikten ... 2616
TOAST-Operationen verstehen ... 2616
Identifizierung von Leistungsherausforderungen ... 2617
Empfehlungen ... 2619
Überwachen ... 2620
Verwenden von Babelfish for Aurora PostgreSQL ... 2625
Babelfish-Einschränkungen ... 2627
Grundlagen der Babelfish-Architektur und -Konfiguration ... 2628
Erstellen eines DB-Clusters von Babelfish for Aurora PostgreSQL ... 2676
Migrieren einer SQL-Server-Datenbank zu Babelfish ... 2687
Datenbankauthentifizierung mit Babelfish für Aurora PostgreSQL ... 2698
Verbinden mit einem Babelfish-DB-Cluster ... 2718
Arbeiten mit Babelfish ... 2731
Fehlerbehebung bei Babelfish ... 2817
Deaktivieren von Babelfish ... 2819
Versionsupdates von Babelfish verwalten ... 2820

Babelfish-Referenz ... 2843
Leistung und Skalierung für Aurora PostgreSQL ... 2901
Skalierung von Aurora PostgreSQL-DB-Instances ... 2902
Maximale Anzahl der Verbindungen ... 2903
Temporäre Speicherlimits ... 2904
Huge Pages für PostgreSQL ... 2908
Testen von Amazon Aurora PostgreSQL unter Verwendung von Fehlersimulationsabfragen ... 2909
Anzeigen des Volume-Status für einen Aurora-DB-Cluster ... 2915
RAM-Datenträger für das stats_temp_directory ... 2916
Verwalten temporärer Dateien mit PostgreSQL ... 2917
Optimierung mit Warteereignissen für Aurora PostgreSQL ... 2924
Grundlegende Konzepte für Aurora PostgreSQL-Optimierung ... 2925
Aurora PostgreSQL-Warteereignisse ... 2931
Client:ClientRead ... 2933
Client:ClientWrite ... 2937
CPU ... 2939
IO: BufFileRead und IO: BufFileWrite ... 2946
IO:DataFileRead ... 2955
IO:XactSync ... 2966
IPC:DamRecordTxAck ... 2968
IPC:parallel-Warteereignisse ... 2970
IPC: ProcArrayGroupUpdate ... 2976
Lock:advisory ... 2979
Lock:extend ... 2982
Lock:Relation ... 2985
Lock:transactionid ... 2992
Lock:tuple ... 2995
LWLock:buffer_content (BufferContent) ... 3000
LWLock:buffer_mapping ... 3004
LWLock:BufferIO (IPC:BufferIO) ... 3007
LWLock:lock_manager ... 3009
LWLock:MultiXact ... 3014
LWLock:pg_stat_statements ... 3018
Timeout:PgSleep ... 3022
Optimierung von Aurora PostgreSQL mit proaktiven Einblicken von Amazon DevOps Guru ... 3023

Die Datenbank läuft seit langem inaktiv in Transaktionsverbindung ... 3024
Bewährte Methoden mit Aurora PostgreSQL ... 3027
Vermeiden von Leistungseinbußen, automatischem Neustart und Failover für DB-Instances von Aurora PostgreSQL ... 3028
Diagnostizieren einer Überlastung von Tabellen und Indizes ... 3028
Verbesserte Arbeitsspeicherverwaltung in Aurora PostgreSQL ... 3033
Schnelles Failover ... 3035
Schnelle Wiederherstellung nach Failover ... 3048
Verwalten von Verbindungsproblemen ... 3055
Behandeln von inaktiven Verbindungen in PostgreSQL ... 3063
Ändern von Speicherparametern für Aurora PostgreSQL ... 3066
Analysieren Sie die Ressourcennutzung mit Metriken CloudWatch ... 3075
Verwenden der logischen Replikation für ein Hauptversions-Upgrade ... 3079
Fehlerbehebung bei Speicherproblemen in Aurora PostgreSQL ... 3089
Replikation mit Aurora PostgreSQL ... 3090
Aurora-Replikate ... 3091
Verbesserung der Leseverfügbarkeit von Aurora Replicas ... 3092
Überwachung einer -Replikation ... 3094
Überblick über die logische Replikation ... 3094
Einrichten der logischen Replikation ... 3096
Deaktivieren der logischen Replikation ... 3098
Überwachen des Write-Through-Caches und der logischen Slots ... 3099
Beispiel: Verwenden der logischen Replikation ... 3102
Beispiel: Logische Replikation mit Aurora PostgreSQL und AWS DMS ... 3104
Konfiguration der IAM-Authentifizierung für logische Replikationsverbindungen ... 3107
Lokale Schreibweiterleitung in Aurora PostgreSQL ... 3110
Einschränkungen und Überlegungen zur lokalen Schreibweiterleitung in Aurora PostgreSQL ... 3111
Konfiguration von Aurora PostgreSQL für die lokale Schreibweiterleitung ... 3112
Arbeiten mit der lokalen Schreibweiterleitung für Aurora PostgreSQL ... 3116
Überwachen der lokalen Schreibweiterleitung in Aurora PostgreSQL ... 3120
Verwenden von Aurora PostgreSQL als Wissensdatenbank für Amazon Bedrock ... 3127
Voraussetzungen ... 3127
Vorbereiten von Aurora PostgreSQL als Wissensdatenbank ... 3128
Erstellen einer Wissensdatenbank in der Bedrock-Konsole ... 3131
Schnellerstellen einer Aurora-PostgreSQL-Wissensdatenbank für Amazon Bedrock ... 3133

Integrieren von Aurora PostgreSQL mit AWS-Services ... 3136
Importieren von Daten aus Amazon S3 in Aurora PostgreSQL ... 3137
Exportieren von PostgreSQL-Daten nach Amazon S3 ... 3157
Aufrufen einer Lambda-Funktion aus Aurora PostgreSQL ... 3175
Veröffentlichen von Aurora-PostgreSQL-Protokollen in CloudWatch Logs ... 3191
Überwachen der Ausführungspläne von Abfragen und deren maximale Speichernutzung für Aurora PostgreSQL ... 3204
Zugreifen auf Ausführungspläne von Abfragen und deren maximale Speichernutzung mithilfe von Aurora-Funktionen ... 3205
Parameterreferenz für Ausführungspläne von Abfragen in Aurora PostgreSQL ... 3206
Verwalten von Abfrageausführungsplänen für Aurora PostgreSQL ... 3210
Übersicht über die Abfrageplanverwaltung in Aurora PostgreSQL ... 3211
Bewährte Methoden für die Aurora-PostgreSQL-Abfrageplanverwaltung ... 3219
Abfrageplanverwaltung ... 3223
Erfassung von Aurora-PostgreSQL-Ausführungsplänen ... 3225
Verwenden von Aurora-PostgreSQL-Plänen ... 3228
Untersuchen von Aurora-PostgreSQL-Abfrageplänen in der dba_plans-Ansicht ... 3233
Verbessern von Plänen ... 3234
Löschen von Plänen ... 3238
Exportieren und Importieren von verwalteten Plänen ... 3241
Parameterreferenz ... 3242
Funktionsreferenz ... 3249
Referenz für die apg_plan_mgmt.dba_plans-Ansicht ... 3261
Erweiterte Funktionen der Abfrageplanverwaltung ... 3266
Arbeiten mit Erweiterungen und Fremddaten-Wrappern ... 3281
Verwenden der Unterstützung für delegierte Erweiterungen von Amazon Aurora für PostgreSQL ... 3282
Effizienteres Verwalten großer Objekte mit dem lo-Modul ... 3297
Verwalten von Geodaten mit PostGIS ... 3301
Verwalten von Partitionen mit der Erweiterung pg_partman ... 3311
Planen der Wartung mit der Erweiterung pg_cron ... 3318
Verwenden von pgAudit zur Protokollierung der Datenbankaktivität ... 3328
Verwenden von pglogical, um Daten zu synchronisieren ... 3342
Unterstützte Fremddaten-Wrapper in Amazon Aurora PostgreSQL ... 3357
Arbeiten mit Trusted Language Extensions für PostgreSQL ... 3373
Terminologie ... 3374

Anforderungen für die Verwendung von Trusted Language Extensions ... 3375
Einrichten von Trusted Language Extensions ... 3378
Übersicht über Trusted Language Extensions ... 3382
Erstellen von TLE-Erweiterungen ... 3384
Löschen Ihrer TLE-Erweiterungen aus einer Datenbank ... 3389
Deinstallieren von Trusted Language Extensions ... 3391
Verwenden von PostgreSQL-Haken mit Ihren TLE-Erweiterungen ... 3392
Funktionsreferenz für Trusted Language Extensions ... 3398
Hakenreferenz für Trusted Language Extensions ... 3412
Aurora-PostgreSQL-Referenz ... 3416
Aurora-PostgreSQL-Kollationen für EBCDIC- und andere Mainframe-Migrationen ... 3416
In Aurora PostgreSQL unterstützte Sortierungen ... 3418
Aurora-PostgreSQL-Funktionsreferenz ... 3419
Aurora-PostgreSQL-Parameter ... 3478
Aurora-PostgreSQL-Warteereignisse ... 3542
Aurora PostgreSQL-Aktualisierungen ... 3574
Identifizieren der Versionen von Amazon Aurora PostgreSQL ... 3574
Aurora PostgreSQL Veröffentlichungen ... 3576
Erweiterungsversionen für Aurora PostgreSQL ... 3577
Upgrade von DB-Clustern von Amazon Aurora PostgreSQL ... 3577
Long-Term Support- (LTS, Langzeit-Support) Versionen nutzen ... 3619
Verwenden von Aurora PostgreSQL Limitless Database ... 3622
Die Architektur von Limitless Database ... 3623
Wichtige Begriffe ... 3625
Tabellentypen ... 3627
Fakturierung ... 3627
Erste Schritte mit Limitless Database ... 3629
Voraussetzungen und Überlegungen zu Aurora PostgreSQL Limitless Database ... 3630
Voraussetzungen ... 3630
Überlegungen ... 3631
Nicht unterstützte Features ... 3634
Voraussetzungen für die Verwendung von Limitless Database ... 3635
Aktivieren von DB-Shard-Gruppenoperationen ... 3635
Erstellen eines DB-Clusters, der Limitless Database verwendet ... 3637
Korrelieren der maximalen Kapazität mit Routern und Shards ... 3637
Erstellen Ihres DB Clusters ... 3640

Arbeiten mit DB-Shard-Gruppen ... 3650
Herstellen einer Verbindung zu Ihrem Aurora PostgreSQL Limitless Database-DB-Cluster 3651
Ermitteln der Anzahl von Routern und Shards in einer DB-Shard-Gruppe ... 3652
Beschreibung von DB-Shard-Gruppen ... 3652
Neustarten einer DB-Shard-Gruppe ... 3653
Ändern der Kapazität einer DB-Shard-Gruppe ... 3654
Unterteilen eines Shards ... 3656
Hinzufügen eines Routers ... 3662
Löschen einer DB-Shard-Gruppe ... 3667
Hinzufügen einer DB-Shard-Gruppe zu einem vorhandenen DB-Cluster von Limitless Database ... 3668
Erstellen von Tabellen in Limitless Database ... 3673
Erstellen von Limitless-Tabellen mithilfe von Variablen ... 3674
Konvertieren von Standardtabellen in Limitless-Tabellen ... 3680
Beispielschemata ... 3685
Laden von Daten in Limitless Database ... 3686
Verwenden des COPY-Befehls in Limitless Database ... 3688
Verwenden des Dienstprogramms zum Laden von Daten für Limitless Database ... 3690
Abfragen einer Limitless Database ... 3720
Single-Shard-Abfragen ... 3721
Verteilte Abfragen ... 3731
Verteiltes Abfrage-Tracing ... 3732
Verteilte Deadlocks ... 3741
Verwalten einer Limitless Database ... 3744
Überlegungen zur Datenbank- und Tabellengröße ... 3744
Zurückgewinnen von Speicherplatz durch Bereinigung ... 3745
Überwachen einer Limitless Database ... 3749
Überwachen von Limitless Database mit CloudWatch ... 3750
Überwachen einer Limitless Database mithilfe von Database Insights ... 3756
Überwachen von Limitless Database mit CloudWatch Logs ... 3769
Überwachen einer Limitless Database mithilfe der erweiterten Überwachung ... 3770
Überwachen einer Limitless Database mithilfe von Performance Insights ... 3771
Überwachen einer Aurora PostgreSQL Limitless Database mithilfe des RDS-Schutzes von GuardDuty ... 3779
Funktionen und Ansichten für Limitless Database ... 3780
Warteereignisse für Limitless Database ... 3806

Sichern und Wiederherstellen von Limitless Database ... 3819
Sicheren eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet ... 3819
Wiederherstellen eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet ... 3822
PostgreSQL-Dienstprogramme zur Sicherung und Wiederherstellung werden nicht unterstützt ... 3825
Aktualisieren von Limitless Database ... 3826
Aktualisieren von DB-Clustern, die Aurora PostgreSQL Limitless Database verwenden ... 3826
Referenz zu Limitless Database ... 3828
DDL-Befehle ... 3828
DDL-Einschränkungen und andere Informationen ... 3837
DML--Befehle ... 3877
DML-Einschränkungen und andere Informationen ... 3880
Variablen ... 3884
DB-Cluster-Parameter ... 3885
Verwenden von Aurora Global Database ... 3886
Übersicht über Aurora Global Database ... 3886
Vorteile von Amazon Aurora Global Database ... 3888
Verfügbarkeit von Regionen und Versionen ... 3889
Einschränkungen von Aurora Global Database ... 3889
Erste Schritte mit Aurora Global Database ... 3892
Konfigurationsanforderungen ... 3893
Erstellen einer globalen Datenbank ... 3894
Hinzufügen eines sekundären Clusters ... 3909
Erstellen eines sekundären Headless-Clusters ... 3914
Erstellen einer globalen Datenbank aus einem Snapshot ... 3917
Verwalten einer Aurora Global Database ... 3918
Verändern einer Aurora Global Database ... 3919
Ändern globaler Datenbankparameter ... 3921
Entfernen eines Clusters aus einer Aurora Global Database ... 3922
Löschen einer Aurora Global Database ... 3925
Tagging bei Aurora Global Database ... 3927
Verbinden mit Aurora Global Database ... 3932
Auswählen des Endpunkts, der Ihren Anwendungsanforderungen entspricht ... 3933
Anzeigen der Endpunkte einer globalen Datenbank ... 3934
Überlegungen zur Verwendung globaler Writer-Endpunkte ... 3938

Verwenden der Schreibweiterleitung in einer Aurora globalen Datenbank 3939
Verwenden der Schreibweiterleitung in Aurora MySQL 3939
Verwenden der Schreibweiterleitung in Aurora PostgreSQL 3961
Verwenden von Umstellung oder Failover in Aurora Global Database 3978
Planen in Bezug auf BCDR 3979
Durchführen von Umstellungen 3980
Wiederherstellen nach einem nicht geplanten Ausfall 3987
Verwalten von RPOs (Aurora PostgreSQL) 3998
Regionsübergreifende Resilienz für sekundäre Cluster 4004
Überwachung einer Aurora Global Database 4005
Überwachung einer Aurora globalen Datenbank mit Performance Insights 4006
Überwachung von globalen Aurora-Datenbanken mithilfe von Datenbank-Aktivitätsstreams 4007
Überwachung der Aurora-MySQL-basierten globalen Datenbanken 4007
Überwachen von Aurora-PostgreSQL-basierten globalen Datenbanken 4011
Blue/Green Bereitstellungen für Aurora Global Database verwenden 4014
Vorteile der Verwendung von Blue/Green Bereitstellungen mit Aurora Global Database 4015
So funktionieren Blue/Green Bereitstellungen mit Aurora Global Database 4015
Verwenden von globalen Aurora-Datenbanken mit anderen AWS-Services 4018
Aktualisieren einer globalen Datenbank von Amazon Aurora 4020
Hauptversions-Upgrades 4020
Unterversion-Upgrades 4021
Amazon-RDS-Proxy 4026
Verfügbarkeit von Regionen und Versionen 4028
Kontingente und Einschränkungen 4028
Einschränkungen für MySQL 4030
PostgreSQL-Beschränkungen 4031
Planen des Verwendungsortes von RDS-Proxy 4033
Konzepte und Terminologie zu RDS-Proxy 4034
Überblick über RDS-Proxy-Konzepte 4035
Verbindungspooling 4036
Sicherheit 4037
Failover 4040
Transaktionen 4041
Erste Schritte mit RDS-Proxy 4042
Einrichten eines Proxy-Netzwerks 4042

Einrichten von Datenbank-Anmeldeinformationen 4048
Konfigurieren der IAM-Authentifizierung 4052
Erstellen eines Proxys 4059
Anzeigen eines Proxys 4072
Verbinden über RDS-Proxy 4074
Verwalten eines RDS-Proxy 4078
Ändern eines RDS-Proxy 4079
Hinzufügen eines Datenbankbenutzers 4087
Umstellung auf IAM-Authentifizierung end-to-end 4088
Überlegungen zu RDS-Proxy-Verbindungen 4092
Vermeiden von Pinning beim RDS-Proxy 4096
Löschen eines RDS-Proxy 4102
Arbeiten mit RDS-Proxy-Endpunkten 4103
Überblick über Proxy-Endpunkte 4104
Limits für Proxy-Endpunkte 4105
Verwenden von Reader-Endpunkten mit Aurora-Clustern 4106
Zugriff auf Aurora über VPCs 4110
Erstellen eines Proxy-Endpunktes 4112
Anzeigen von Proxy-Endpunkten 4115
Ändern eines Proxy-Endpunkts 4117
Löschen eines Proxy-Endpunkts 4118
Überwachen von RDS-Proxy mit CloudWatch 4119
Arbeiten mit RDS-Proxy-Ereignissen 4128
RDS-Proxy-Ereignisse 4129
Beispiele für RDS Proxy 4132
Fehlerbehebung für RDS-Proxy 4134
Überprüfen der Konnektivität für einen Proxy 4135
Häufige Probleme und Lösungen 4137
Fehlerbehebung für RDS-Proxy 4139
Problembehandlung für RDS für PostgreSQL 4141
Verwenden von RDS-Proxy mit AWS CloudFormation 4148
Verwenden von RDS Proxy mit globalen Aurora-Datenbanken 4149
Einschränkungen für RDS Proxy mit globalen Datenbanken 4149
So funktionieren RDS-Proxy-Endpunkte mit globalen Datenbanken 4149
Zero-ETL-Integrationen 4151
Vorteile 4153

Die wichtigsten Konzepte 4153
Einschränkungen 4154
Allgemeine Einschränkungen 4154
Einschränkungen von Aurora MySQL 4155
Einschränkungen bei Aurora PostgreSQL 4156
Einschränkungen für Amazon Redshift 4157
Amazon SageMaker AI Lakehouse-Einschränkungen 4157
Kontingente 4157
Unterstützte Regionen 4157
Erste Schritte mit Null-ETL-Integrationen 4158
Schritt 1: Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe 4159
Schritt 2: Wählen oder Erstellen eines Quell-DB-Clusters 4160
Schritt 3a: Erstellen eines Ziel-Data-Warehouse 4160
Richten Sie eine Integration mit dem ein AWS SDKs 4162
Schritt 3b: Erstellen Sie einen AWS Glue Katalog für die Amazon SageMaker AI Zero-ETL-Integration 4172
Nächste Schritte 4175
Erstellen von Null-ETL-Integrationen mit Amazon Redshift 4176
Voraussetzungen 4176
Erforderliche Berechtigungen 4176
Erstellen von Null-ETL-Integrationen 4179
Verschlüsseln von Integrationen 4184
Nächste Schritte 4186
Erstellen von Null-ETL-Integrationen mit einem Amazon SageMaker-Lakehouse 4186
Voraussetzungen 4187
Erforderliche Berechtigungen 4187
Erstellen von Null-ETL-Integrationen mit einem Amazon SageMaker-Lakehouse 4190
Verschlüsseln von Integrationen 4195
Nächste Schritte 4197
Datenfilterung für Null-ETL-Integrationen 4197
Format eines Datenfilters 4198
Filterlogik 4201
Rangfolge der Filter 4202
Beispiele für Aurora MySQL 4202
Beispiele für Aurora PostgreSQL 4203
Hinzufügen von Datenfiltern 4204

Entfernen von Datenfiltern ... 4206
Daten hinzufügen und abfragen ... 4206
Erstellen einer Zieldatenbank ... 4207
Hinzufügen von Daten zum Quell-DB-Cluster ... 4207
Abfragen Ihrer Aurora-Daten in Amazon Redshift ... 4209
Datentypunterschiede ... 4210
DDL-Vorgänge für Aurora PostgreSQL ... 4223
Anzeigen und Überwachen von Null-ETL-Integrationen ... 4226
Anzeigen von Integrationen ... 4227
Überwachen mithilfe von Systemtabellen ... 4228
Überwachung mit EventBridge ... 4229
Ändern von Null-ETL-Integrationen ... 4229
Löschen von Null-ETL-Integrationen ... 4231
Fehlerbehebung bei Null-ETL-Integrationen ... 4232
Ich kann keine Null-ETL-Integration erstellen ... 4233
Meine Integration steckt im Status `Syncing` fest ... 4233
Meine Tabellen werden nicht in Amazon Redshift repliziert ... 4234
Eine oder mehrere meiner Amazon-Redshift-Tabellen erfordern eine erneute Synchronisation ... 4234
Fehlgeschlagene Integration bei Null-ETL-Integrationen von Lakehouse in Amazon SageMaker AI ... 4238
DDL-Operationen, die nicht abgeschlossen sind, werden in Amazon Redshift angezeigt ... 4239
Verwenden von Aurora Serverless v2 ... 4240
Anwendungsfälle für Aurora Serverless v2 ... 4240
Konvertieren bereitgestellter Workloads ... 4243
Vorteile von Aurora Serverless v2 ... 4243
Funktionsweise von Aurora Serverless v2 ... 4245
Übersicht ... 4245
Cluster-Konfigurationen ... 4247
Capacity (Kapazität) ... 4248
Skalierung ... 4250
Hohe Verfügbarkeit ... 4253
Speicher ... 4254
Konfigurationsparameter ... 4255
Anforderungen und Einschränkungen für Aurora Serverless v2 ... 4255
Verfügbarkeit von Regionen und Versionen ... 4255

Für Cluster, die Aurora Serverless v2 verwenden, muss ein Kapazitätsbereich angegeben sein ... 4256
Einige bereitgestellte Funktionen werden in Aurora Serverless v2 nicht unterstützt ... 4257
Einige Aurora Serverless v2-Aspekte unterscheiden sich von Aurora Serverless v1 ... 4258
Erstellen eines Aurora Serverless v2-DB Clusters ... 4258
Einstellungen ... 4258
Erstellen eines Aurora Serverless v2-DB Clusters ... 4260
Erstellen eines Aurora Serverless v2-Writers ... 4265
Verwalten von Aurora Serverless v2 ... 4266
Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster ... 4267
Überprüfen des Kapazitätsbereichs für Aurora Serverless v2 ... 4274
Hinzufügen eines Aurora Serverless v2-Readers ... 4275
Konvertieren von bereitgestellten Instances in Aurora Serverless v2 ... 4277
Konvertieren von Aurora Serverless v2 in bereitgestellt ... 4278
Anhalten von Aurora Serverless v2-DB-Instances ... 4279
Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader ... 4279
Verwenden von TLS/SSL mit Aurora Serverless v2 ... 4280
Anzeigen von Aurora Serverless v2-Writern und -Readern ... 4282
Protokollierung für Aurora Serverless v2 ... 4284
Performance und Skalierung für Aurora Serverless v2 ... 4291
Auswählen des Kapazitätsbereichs ... 4292
Arbeiten mit Parametergruppen für Aurora Serverless v2 ... 4307
Vermeiden von Fehlern aufgrund von zu wenig Speicher ... 4312
Wichtige CloudWatch-Metriken ... 4314
Überwachung der Aurora Serverless v2-Leistung mit Performance Insights ... 4319
Behebung von Kapazitätsproblemen bei Aurora Serverless v2 ... 4319
Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen ... 4322
Überblick über die die Auto-Pause-Funktion ... 4322
Voraussetzungen und Einschränkungen ... 4324
Aktivieren und Deaktivieren der Auto-Pause-Funktion ... 4325
Funktionsweise der Auto-Pause-Funktion ... 4329
Auto-Pause-Funktion und Aurora-Cluster-Konfigurationen ... 4334
Auto-Pause bei der Überwachung ... 4338
Fehlerbehebung für die Auto-Pause-Funktion ... 4342
Anwendungsdesign für die Auto-Pause-Funktion ... 4344
Migration zu Aurora Serverless v2 ... 4345

Verwenden von Aurora Serverless v2 mit einem vorhandenen Cluster ... 4346
Umstellung von einem bereitgestellten Cluster ... 4350
Aurora Serverless v2 und Aurora Serverless v1 im Vergleich ... 4356
Aktualisieren von Aurora Serverless v1 auf Aurora Serverless v2 ... 4367
Migrieren einer On-Premises-Datenbank zu Aurora Serverless v2 ... 4370
Verwenden von Aurora Serverless v1 ... 4371
Verfügbarkeit von Regionen und Versionen für Aurora Serverless v1 ... 4372
Vorteile von Aurora Serverless v1 ... 4372
Anwendungsfälle für Aurora Serverless v1 ... 4373
Einschränkungen von Aurora Serverless v1 ... 4373
Anforderungen an die Konfiguration von Aurora Serverless v1 ... 4376
Verwenden von TLS/SSL mit Aurora Serverless v1 ... 4377
Unterstützte Verschlüsselungssammlungen für Verbindungen mit Aurora Serverless v1-DB-Clustern ... 4380
Funktionsweise von Aurora Serverless v1 ... 4381
Aurora Serverless v1 architecture ... 4381
Auto Scaling ... 4383
Timeout-Aktion ... 4384
Pausieren und Fortsetzen ... 4386
Bestimmen von max_connections ... 4386
Parametergruppen ... 4389
Protokollierung ... 4393
Wartung ... 4397
Failover ... 4398
-Snapshots ... 4398
Erstellen eines Aurora Serverless v1-DB Clusters ... 4398
Wiederherstellen eines Aurora Serverless v1-DB-Clusters ... 4403
Ändern eines Aurora Serverless v1-DB-Clusters ... 4406
Ändern der Skalierungskonfiguration ... 4407
Durchführen eines Upgrades der Hauptversion ... 4410
Konvertieren von Aurora Serverless v1 in bereitgestellt ... 4412
Überlegungen bei der Konvertierung von einem Aurora Serverless v1-DB-Cluster zu einem bereitgestellten Cluster ... 4414
Manuelles Skalieren der Kapazität des Aurora Serverless v1-DB-Clusters ... 4415
Anzeigen von Aurora Serverless v1-DB-Clustern ... 4418
Überwachung von Aurora Serverless v1-DB-Clustern mit CloudWatch ... 4422

Löschen eines Aurora Serverless v1-DB-Clusters ... 4423
Aurora Serverless v1- und Aurora-Datenbank-Engine-Versionen ... 4426
Aurora MySQL Serverless ... 4427
Aurora PostgreSQL Serverless ... 4428
Automatische Unterversion-Upgrades ... 4428
Verwenden der Daten-API von Amazon RDS ... 4429
Verfügbarkeit von Regionen und Versionen für die Amazon-RDS-Daten-API ... 4430
IPv6 Unterstützung ... 4431
IPv6 Endpunkt-Unterstützung ... 4431
Endpunkte verwenden IPv6 ... 4432
Verwenden mit AWS PrivateLink IPv6 ... 4434
Überlegungen zur Migration ... 4435
Fehlerbehebung bei IPv6 der Konnektivität ... 4435
Einschränkungen ... 4436
Daten-API mit Aurora Serverless v2 oder mit Aurora Serverless v1 – Vergleich ... 4437
Maximale Anzahl von Anforderungen pro Sekunde ... 4437
Aktivieren oder Deaktivieren der Daten-API von Amazon RDS in einer vorhandenen Datenbank ... 4437
CloudTrail-Ereignisse ... 4438
Unterstützung für mehrere Anweisungen ... 4438
Gleichzeitige Anforderungen für dieselbe Transaktions-ID ... 4439
Wie sich BatchExecuteStatement verhält ... 4440
Wie sich ExecuteSQL verhält ... 4441
Wie sich ExecuteStatement verhält ... 4441
Wie sich der Parameter Schema verhält ... 4442
Authentifizierung und Autorisierung ... 4443
Tag-basierte Autorisierung ... 4444
Speichern von Anmeldeinformationen in einem Secrets-Manager-Geheimnis ... 4446
Aktivieren der Daten-API ... 4447
Aktivieren der RDS-Daten-API beim Erstellen einer Datenbank ... 4448
Aktivieren der RDS-Daten-API in einer vorhandenen Datenbank ... 4449
Erstellen eines Amazon-VPC-Endpunkts (PrivateLink) ... 4452
Aufrufen der Daten-API ... 4456
Referenz zu den Operationen ... 4456
Aufrufen über die AWS CLI ... 4461
Aufrufen aus Python-Apps ... 4472

Aufrufen aus Java-Apps ... 4476
Steuern des Timeout-Verhaltens ... 4480
Java-Client-Bibliothek ... 4482
Herunterladen der Java Client-Bibliothek für die Daten-API ... 4483
Java Client-Bibliothek-Beispiele ... 4483
Verarbeiten von Abfrageergebnissen in JSON ... 4485
Abrufen von Abfrageergebnissen im JSON-Format ... 4486
Datentypenzuordnung ... 4486
Fehlerbehebung ... 4487
Beispiele ... 4488
Fehlerbehebung bei Problemen mit der Daten-API ... 4493
Transaktion <transaction_ID> nicht gefunden ... 4493
Packet for Query Is Too Large (Paket für Abfrage zu groß) ... 4494
Database Response Exceeded Size Limit Datenbankantwort überschreitet Größenlimit) ... 4494
HttpEndpointist nicht für Cluster aktiviert <cluster_ID> ... 4495
DatabaseErrorException: Transaction führt immer noch eine Abfrage aus ... 4495
Ausnahme aufgrund eines nicht unterstützten Ergebnisses ... 4496
Keine Unterstützung für mehrere Anweisungen ... 4496
Keine Unterstützung für den Schemaparameter ... 4496
IPv6 Probleme mit der Konnektivität ... 4496
Protokollieren von Daten-API-Aufrufen ... 4497
Arbeiten mit Data API-Informationen in CloudTrail ... 4498
Einschließen und Ausschließen von Daten-API-Ereignissen aus einem CloudTrail-Trail ... 4499
Grundlegendes zu Data API-Protokolldateieinträgen ... 4501
Überwachen von Daten-API-Abfragen ... 4504
So werden RDS-Daten-API-Abfragen in Performance Insights dargestellt ... 4504
Verwenden des Abfrage-Editors ... 4505
Verfügbarkeit des Abfrage-Editors ... 4505
Autorisieren des Zugriffs ... 4505
Ausführen von Abfragen ... 4507
DBQMS-API-Referenz. ... 4511
CreateFavoriteQuery ... 4512
CreateQueryHistory ... 4512
CreateTab ... 4512
DeleteFavoriteQueries ... 4512
DeleteQueryHistory ... 4512

DeleteTab ... 4512
DescribeFavoriteQueries ... 4513
DescribeQueryHistory ... 4513
DescribeTabs ... 4513
GetQueryString ... 4513
UpdateFavoriteQuery ... 4513
UpdateQueryHistory ... 4513
UpdateTab ... 4513
Verwenden von Aurora Machine Learning ... 4514
Verwendung von Aurora Machine Learning mit Aurora MySQL ... 4515
Voraussetzungen für die Verwendung von Aurora Machine Learning ... 4516
Verfügbarkeit von Regionen und Versionen ... 4517
Unterstützte Funktionen und Einschränkungen ... 4518
Einrichten Ihres Aurora-Clusters zur Verwendung von Aurora Machine Learning ... 4519
Verwenden von Amazon Bedrock mit Ihrem DB-Cluster von Aurora MySQL ... 4534
Verwenden von Amazon Comprehend mit Ihrem DB-Cluster von Aurora MySQL ... 4536
Verwenden von SageMaker AI mit Ihrem DB-Cluster von Aurora MySQL ... 4539
Leistungsaspekte ... 4543
Überwachen ... 4545
Verwendung von Aurora Machine Learning mit Aurora PostgreSQL ... 4546
Voraussetzungen für die Verwendung von Aurora Machine Learning ... 4547
Unterstützte Funktionen und Einschränkungen ... 4548
Einrichten Ihres Aurora-DB-Clusters zur Verwendung von Aurora Machine Learning ... 4549
Verwenden von Amazon Bedrock mit Ihrem DB-Cluster von Aurora PostgreSQL ... 4562
Verwenden von Amazon Comprehend mit Ihrem DB-Cluster von Aurora PostgreSQL ... 4564
SageMaker KI mit Ihrem Aurora PostgreSQL-DB-Cluster verwenden ... 4566
Exportieren von Daten nach Amazon S3 für das SageMaker KI-Modelltraining (Fortgeschritten) ... 4571
Leistungsaspekte ... 4572
Überwachen ... 4578
Codebeispiele ... 4580
Grundlagen ... 4591
Hello Aurora ... 4592
Kennenlernen der Grundlagen ... 4602
Aktionen ... 4772
Szenarien ... 4978

Leihbibliothek-REST-API erstellen 4979
Erstellen Sie einen Tracker für Aurora-Serverless-Arbeitsaufgaben 4980
Bewährte Methoden mit Aurora 4985
Grundlegende Anleitungen für Amazon Aurora 4985
RAM-Empfehlungen für DB-Instances 4986
AWS-Datenbanktreiber 4987
Überwachen von Amazon Aurora 4987
Arbeiten mit DB-Parametergruppen und DB-Cluster-Parametergruppen 4988
Video zu bewährten Amazon-Aurora-Methoden 4988
Durchführen eines Aurora-Machbarkeitsnachweises 4989
Übersicht über einen Aurora-Machbarkeitsnachweis 4989
1. Identifizieren Ihrer Ziele 4990
2. Verständnis der Workload-Eigenschaften 4991
3. Durchführen einer Übung mit der Konsole oder der CLI 4992
Durchführen einer Übung mit der Konsole 4992
Durchführen einer Übung mit der AWS CLI 4993
4. Erstellen Ihres Aurora-Clusters 4994
5. Einrichten Ihres Schemas 4995
6. Importieren Ihrer Daten 4996
7. Portieren Ihres SQL-Codes 4997
8. Angeben der Konfigurationseinstellungen 4998
9. Verbinden mit Aurora 4999
10. Ausführen Ihrer Workload 5001
11. Messen der Leistung 5002
12. Ausführen der Aurora-Hochverfügbarkeit 5005
13. Weitere Vorgehensweisen 5007
Sicherheit 5010
Datenbankauthentifizierung 5012
Passwortauthentifizierung 5014
IAM-Datenbankauthentifizierung 5014
Kerberos-Authentifizierung 5014
Passwortverwaltung mit Aurora und Secrets Manager 5016
Verfügbarkeit von Regionen und Versionen 5016
Einschränkungen 5016
Übersicht 5017
Vorteile 5018

Berechtigungen ... 5018
Erzwingen der Aurora-Verwaltung ... 5019
Verwalten des Passworts für einen DB-Cluster ... 5020
Rotieren von Secrets für -DB-Cluster ... 5025
Anzeigen von Details zu Secrets für DB-Cluster ... 5027
Datenschutz ... 5030
Datenverschlüsselung ... 5031
Richtlinie für den Datenverkehr zwischen Netzwerken ... 5062
Identity and Access Management ... 5064
Zielgruppe ... 5064
Authentifizierung mit Identitäten ... 5065
Verwalten des Zugriffs mit Richtlinien ... 5068
Funktionsweise von Amazon Aurora mit IAM ... 5070
Beispiele für identitätsbasierte Richtlinien ... 5078
Von AWS verwaltete Richtlinien ... 5101
Richtlinienaktualisierungen ... 5108
Serviceübergreifende Confused-Deputy-Prävention ... 5121
IAM-Datenbankauthentifizierung ... 5123
Fehlersuche ... 5176
Protokollierung und Überwachung ... 5178
Compliance-Validierung ... 5182
Ausfallsicherheit ... 5183
Backup und Backup ... 5183
Replikation ... 5184
Failover ... 5184
Sicherheit der Infrastruktur ... 5185
Sicherheitsgruppen ... 5185
Öffentliche Zugänglichkeit ... 5185
VPC-Endpunkte (AWS PrivateLink) ... 5187
Überlegungen ... 1181
Verfügbarkeit ... 1181
Erstellen eines Schnittstellen-VPC-Endpunkts ... 1181
Erstellen einer VPC-Endpunktrichtlinie ... 1181
Bewährte Methoden für die Gewährleistung der Sicherheit ... 5191
Zugriffskontrolle mit Sicherheitsgruppen ... 5192
Überblick über VPC-Sicherheitsgruppen ... 5192

Sicherheitsgruppenszenario ... 5193
Erstellen einer VPC-Sicherheitsgruppe ... 5195
Verknüpfen mit einem DB-Cluster ... 5196
Berechtigungen von Hauptbenutzerkonten ... 5196
Serviceverknüpfte Rollen ... 5199
Berechtigungen von serviceverknüpften Rollen für Amazon Aurora ... 5199
Serviceverknüpfte Rollenberechtigungen für Amazon RDS Beta ... 5203
Serviceverknüpfte Rolle für Amazon RDS Preview ... 5204
Verwenden von Amazon Aurora mit Amazon VPC ... 5206
Arbeiten mit einer DB-Cluster in einer VPC ... 5206
Szenarien für den Zugriff auf eine DB-Cluster in einer VPC ... 5225
Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4) ... 5232
Tutorial: Erstellen einer VPC zur Verwendung mit einer DB-einem DB-Cluster (Dual-Stack-Modus) ... 5240
Kontingente und Beschränkungen ... 5252
Kontingente in Amazon Aurora ... 5252
Benennungseinschränkungen in Amazon Aurora ... 5258
Amazon Aurora-Größenbeschränkungen ... 5259
Fehlerbehebung ... 5261
Verbindung zur DB-Instance kann nicht hergestellt werden ... 5261
Testen der DB-Instanceverbindung ... 5264
Fehlerbehebung bei der Verbindungsauthentifizierung ... 5265
Sicherheitsprobleme ... 5265
Fehlermeldung „Failed to retrieve account attributes, certain console functions may be impaired (Fehler beim Abrufen von Kontoattributen, bestimmte Konsolenfunktionen können beeinträchtigt sein).“ ... 5265
Zurücksetzen des Besitzerpassworts der DB-Instance ... 5265
Ausfall oder Neustart einer DB-Instance ... 5266
Parameteränderungen wirken sich nicht aus ... 5267
Probleme mit freisetzbarem Speicher in Aurora ... 5267
Aurora MySQL-Replikationsprobleme ... 5268
Diagnose und Lösung bei Verzögerungen zwischen Read Replicas (Lesereplikaten) ... 5269
Diagnose und Lösung eines Fehlers bei einer MySQL Read Replica ... 5271
Fehler „Replication stopped (Replikation gestoppt)“ ... 5273
Die Replikation des Read Replica kann die Metadatenstruktur nicht initialisieren ... 5273
Amazon RDS-API-Referenz ... 5275

Verwenden der Abfrage-API 5275
Abfrageparameter 5275
Authentifizierung von Abfrageanforderungen 5276
Fehlerbehebung bei Anwendungen 5276
Fehler bei Abrufen 5276
Tipps zur Problembehebung 5277
Dokumentverlauf 5278
AWS Glossar 5387
..... 5388

Was ist Amazon Aurora?

Amazon Aurora (Aurora) ist eine vollständig verwaltete, mit MySQL und PostgreSQL kompatible relationale Datenbank-Engine. Sie wissen bereits, wie MySQL und PostgreSQL die Geschwindigkeit und Zuverlässigkeit von kommerziellen High-End-Datenbanken mit der Einfachheit und Kosteneffizienz von Open-Source-Datenbanken verbinden. Der Code, die Tools und Anwendungen, die Sie heute mit Ihren bestehenden MySQL- und PostgreSQL-Datenbanken verwenden, können auch mit Aurora verwendet werden. Bei manchen Workloads kann Aurora einen bis zu fünfmal höheren Durchsatz als MySQL und einen bis zu dreimal höheren Durchsatz als PostgreSQL liefern, ohne dass die meisten Ihrer bestehenden Anwendungen geändert werden müssen.

Aurora umfasst ein hochleistungsfähiges Speichersubsystem. Die mit MySQL und PostgreSQL kompatiblen Datenbank-Engines werden angepasst, um diesen schnellen verteilten Speicher zu nutzen. Der zugrunde liegende Speicher wächst bei Bedarf automatisch. Ein Aurora-Cluster-Volume kann auf eine maximale Größe von 128 tebibytes (TiB) anwachsen. Aurora automatisiert und standardisiert auch das Clustering und die Replikation von Datenbanken, die normalerweise zu den schwierigsten Aspekten der Datenbankkonfiguration und -verwaltung gehören.

Aurora ist Teil des verwalteten Datenbankservices Amazon Relational Database Service (Amazon RDS). Amazon RDS vereinfacht das Einrichten, Betreiben und Skalieren einer relationalen Datenbank in der Cloud. Wenn Sie noch keine Erfahrung mit Amazon RDS haben, beachten Sie die Informationen im Amazon Relational Database Service Benutzerhandbuch. Weitere Informationen zu den verschiedenen Datenbankoptionen, die in Amazon Web Services verfügbar sind, finden Sie unter Choosing the right database for your organization on AWS.

Themen

- Amazon-RDS-Modell der geteilten Verantwortung
- Funktionsweise von Amazon Amazon Aurora mit Amazon RDS
- Amazon-Aurora-DB-Cluster
- Amazon-Aurora-Versionen
- Regionen und Availability Zones
- Unterstützte Funktionen in Amazon Aurora by AWS-Region und der Aurora-DB-Engine
- Amazon-Aurora-Endpunktverbindungen
- DB-Instance-Klassen von Amazon Aurora

- Amazon-Aurora-Speicher
- Amazon Aurora-Zuverlässigkeit
- Amazon Aurora-Sicherheit
- Hohe Verfügbarkeit für Amazon Aurora
- Replikation mit Amazon Aurora
- Abrechnung von DB-Instances für Aurora

Amazon-RDS-Modell der geteilten Verantwortung

Amazon RDS ist für das Hosting der Softwarekomponenten und der Infrastruktur von DB-Instances und DB-Clustern verantwortlich. Sie sind für die Abfrageoptimierung verantwortlich, d. h. für den Prozess der Anpassung von SQL-Abfragen, um die Leistung zu verbessern. Die Abfrageleistung hängt erheblich vom Datenbankdesign, der Datengröße, der Datenverteilung, der Anwendungs-Workload und den Abfragemustern ab, die stark variieren können. Überwachung und Optimierung sind hochgradig individualisierte Prozesse, für die Sie für Ihre RDS-Datenbanken verantwortlich sind. Sie können Erkenntnisse zur Amazon-RDS-Leistung und andere Tools verwenden, um problematische Abfragen zu identifizieren.

Funktionsweise von Amazon Amazon Aurora mit Amazon RDS

Die folgenden Punkte veranschaulichen die Beziehung zwischen Amazon Aurora und den in Amazon RDS verfügbaren standardmäßigen MySQL- und PostgreSQL-Engines:

- Sie wählen Aurora MySQL oder Aurora PostgreSQL als DB-Engine-Option aus, wenn Sie über Amazon RDS neue Datenbankserver einrichten.
- Aurora nutzt die bekannten Funktionen von Amazon Relational Database Service (Amazon RDS) für Verwaltung und Administration. Aurora verwendet die Amazon-RDS-AWS-Managementkonsole-Schnittstelle, AWS CLI-Befehle und API-Operationen, um routinemäßige Datenbankaufgaben wie Bereitstellung, Patching, Sicherung, Wiederherstellung, Fehlererkennung und Reparatur zu erledigen.
- Aurora-Verwaltungsvorgänge umfassen in der Regel komplette Cluster von Datenbankservern, die durch Replikation synchronisiert werden, und keine einzelnen Datenbank-Instances. Durch das automatische Clustering, die automatische Replikation und Speicherzuweisung können selbst Ihre größten MySQL- und PostgreSQL-Bereitstellungen einfach und kosteneffizient eingerichtet, betrieben und skaliert werden.

- Durch Erstellen und Wiederherstellen von Snapshots oder Einrichtung einer unidirektionalen Replikation können Sie Daten aus Amazon RDS für MySQL und Amazon RDS für PostgreSQL in Aurora einbringen. Sie können Push-Button-Migrationstools verwenden, um Ihre vorhandenen RDS für MySQL- und RDS für PostgreSQL-Anwendungen in Aurora zu konvertieren.

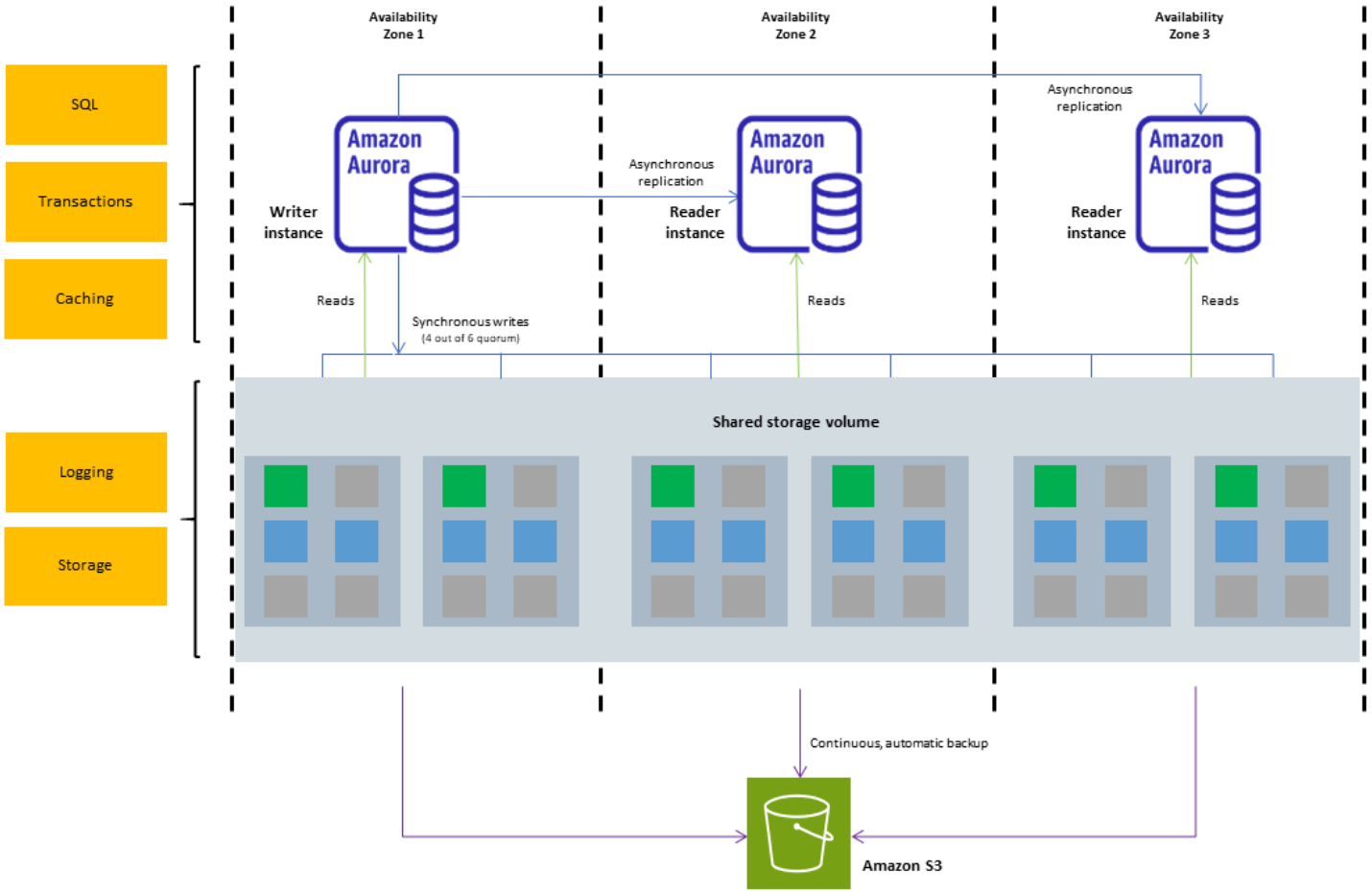
Bevor Sie Amazon Aurora verwenden, führen Sie die Schritte unter Einrichten Ihrer Umgebung für Amazon Aurora aus und sehen Sie sich anschließend im Abschnitt Amazon-Aurora-DB-Cluster die Konzepte und Funktionen von Aurora an.

Amazon-Aurora-DB-Cluster

Ein Amazon Aurora-DB-Cluster besteht aus einer oder mehreren DB-Instances und einem Cluster-Volume, das die Daten für diese DB-Instances verwaltet. Ein Aurora-Cluster-Volume ist ein virtuelles Datenbankspeicher-Volume, das sich über mehrere Availability Zones erstreckt, wobei jede Availability Zone über eine Kopie der DB-Cluster-Daten verfügt. Zwei Typen von DB-Instances bilden einen Aurora-DB-Cluster:

- Primäre (Schreiber-)DB-Instance – Unterstützt Lese- und Schreiboperationen und führt alle Datenänderungen im Cluster-Volume durch. Jeder Aurora-DB-Cluster verfügt über eine primäre DB-Instance.
- Aurora-Replikat (Leser-DB-Instance) – Stellt eine Verbindung zu demselben Speichervolume wie die primäre DB-Instance her und unterstützt nur Lesevorgänge. Jeder Aurora-DB-Cluster kann zusätzlich zur primären DB-Instance bis zu 15 Aurora-Replicas besitzen. Sorgen Sie für eine hohe Verfügbarkeit, indem Sie die Aurora-Replikate in separaten Availability Zones platzieren. Aurora führt ein automatisches Failover auf ein Aurora-Replikat durch, falls die primäre DB-Instance nicht mehr verfügbar ist. Sie können die Failover-Priorität für Aurora-Replicas festlegen. Aurora-Replicas können auch schreibgeschützte Workloads von der primären DB-Instance auslagern.

Das folgende Diagramm veranschaulicht die Beziehung zwischen dem Cluster-Volume, der Schreiber-DB-Instance und Leser-DB-Instances in einem Aurora-DB-Cluster.



> **Note**
>
> Die obigen Informationen gelten für alle Aurora-DB-Cluster – bereitgestellt, Parallelabfrage, Aurora Global Database, Aurora Serverless, Aurora-MySQL-kompatibel und Aurora-PostgreSQL-kompatibel.

Der Aurora-DB-Cluster veranschaulicht die Trennung von Rechenkapazität und Speicher. Beispielsweise ist eine Aurora-Konfiguration mit nur einer einzigen DB-Instance immer noch ein Cluster, da das zugrunde liegende Speichervolumen mehrere Speicherknoten umfasst, die auf mehrere Availability Zones (AZs) verteilt sind.

Ein-/Ausgabe (I/O)-Operationen in Aurora-DB-Clustern werden auf die gleiche Weise gezählt, unabhängig davon, ob sie sich auf einer Writer- oder Reader-DB-Instance befinden. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.

Amazon-Aurora-Versionen

Mit Amazon Aurora können Sie die unterstützte relationale Datenbank-Engine wählen, die Ihren Anwendungsanforderungen am besten entspricht und gleichzeitig die Kompatibilität mit den zugrunde liegenden Engines gewährleistet. Aurora verwendet den Datenbank-Engine-Code von unterstützten Engines wieder, sodass Sie vorhandene Fähigkeiten, Tools und Bibliotheken für diese Engines nutzen können. Wenn Sie einen Cluster erstellen, geben Sie die Version der Amazon-Aurora-Datenbank-Engine an, die Sie verwenden möchten. Die von Ihnen gewählte Version bestimmt die Kompatibilität und die verfügbaren Funktionen.

Diese Dokumentation erklärt die Gemeinsamkeiten und Unterschiede zwischen Amazon Aurora und den entsprechenden Datenbank-Engines. Anhand dieser Informationen können Sie bestimmen, welche Softwareversion Sie auswählen sollten, und wie Sie die verfügbaren Funktionen und Bugfixes in jeder Version überprüfen können. Anhand dieser Referenz können Sie auch eine geeignete Upgrade-Frequenz und einen Plan für Ihr Upgrade festlegen.

Unterstützte Datenbank-Engines für Amazon-Aurora-Datenbank-Cluster

Die folgenden relationalen Datenbanken sind in Amazon Aurora verfügbar. Aurora verwendet Code wieder und behält die Kompatibilität mit den zugrunde liegenden DB-Engines bei. Aurora verfügt jedoch über einzigartige Versionsnummern, Veröffentlichungszyklen und Zeitpläne für die Einstellung der Version. Jede neue Aurora-Version enthält Versionshinweise, in denen die neuen Funktionen, Korrekturen und anderen Änderungen und Verbesserungen aufgeführt sind, die für jede Version gelten.

Aurora-Datenbank	Benutzerhandbuch	Verfügbare Versionen	Versionshinweise
Amazon Aurora MySQL-Compatible Edition	Arbeiten mit Amazon Aurora MySQL	Datenbank-Engine-Updates für Amazon Aurora MySQL	Versionshinweise für Aurora MySQL
Amazon Aurora PostgreSQL-Compatible Edition	Arbeiten mit Amazon Aurora PostgreSQL	Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL	Versionshinweise für Aurora PostgreSQL

Angeben der Amazon-Aurora-Datenbankversion für Ihren Datenbank-Cluster

Wenn Sie einen neuen DB-Cluster mit der Operation Datenbank erstellen in der AWS-Managementkonsole, der AWS CLI oder mit der `CreateDBCluster`-API-Operation erstellen, können Sie jede derzeit verfügbare Aurora-Datenbankversion (Haupt- und Nebenversion) angeben.

Informationen zum Erstellen von Aurora-Clustern finden Sie unter Erstellen eines Amazon Aurora-DB Clusters. Informationen zum Ändern der Version eines vorhandenen Aurora-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

> **Note**
>
> Nicht jede Aurora-Datenbankversion ist in jeder AWS-Region verfügbar. Weitere Informationen zu Regionen und den in jeder AWS-Region verfügbaren Versionen finden Sie unter Regionen und Availability Zones und Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken.

Amazon-Aurora-Versionsverwaltung

Amazon-Aurora-Versionen unterscheiden sich von den Upstream-Community-Datenbanken, mit denen sie kompatibel sind. Um Ihnen zu helfen, die Anwendungskompatibilität aufrechtzuerhalten und die neuesten DB-Engine-Funktionen zu nutzen, werden in den folgenden Abschnitten die Konventionen zur Aurora-Versionsverwaltung und die Zuordnung der Aurora-Versionen zu ihren jeweiligen Community-Datenbanken erläutert.

Eine Liste der relationalen Datenbanken, die in Amazon Aurora verfügbar sind, finden Sie unter Unterstützte Datenbank-Engines für Amazon-Aurora-Datenbank-Cluster.

Unterschiede in den Versionsnummern zwischen Community-Datenbanken und Aurora

Jede Amazon-Aurora-Version ist mit einer bestimmten Version der zugehörigen Community-Datenbank kompatibel. Sie können die Community-Version Ihrer Datenbank mit der `version`-Funktion und die Aurora-Version mit der `aurora_version`-Funktion finden.

Die folgenden Beispiele zeigen, wie Sie die Community-Version Ihrer Datenbank für Aurora MySQL und Aurora PostgreSQL finden.

Aurora MySQL

Die `version`-Funktion gibt die Community-Version Ihrer Datenbank für Aurora MySQL zurück.

```
mysql> select version();
```

Ausgabebeispiel

```
+------------------+
|   version()      |
+------------------+
|  8.0.32          |
+------------------+
```

Und die `aurora_version`-Funktion gibt die Aurora-Version zurück:

```
mysql> select aurora_version(), @@aurora_version;
```

Ausgabebeispiel

```
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

Aurora PostgreSQL

Die `version`-Funktion gibt die Community-Version Ihrer Datenbank für Aurora PostgreSQL zurück.

```
postgres=> select version();
```

Ausgabebeispiel

```
-----------------------------------------------------------------------------
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)
```

Und die `aurora_version`-Funktion gibt die Aurora-Version zurück:

```
postgres=> select aurora_version();
```

Ausgabebeispiel

```
aurora_version
----------------
3.2.2
```

Weitere Informationen erhalten Sie unter Überprüfen von Aurora MySQL-Versionen mit SQL und Identifizieren der Versionen von Amazon Aurora PostgreSQL.

Standard-Amazon-Aurora-Versionen

Die Standardversion ist die Version, die Aurora automatisch für die Datenbankerstellung oder das Upgrade auswählt, wenn Sie keine Ziel-Engine-Version manuell angeben. Der folgende Befehl zeigt beispielsweise die Standard-Engine-Version für Aurora PostgreSQL (einschließlich Beispielausgabe).

```
aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
    --output text

16.4
```

Jede Hauptversion hat eine entsprechende Standard-Nebenversion. Daher ist die Standard-Nebenversion für Aurora PostgreSQL 16 die Version 16.n, wobei sich die Versionsnummer n ändert, wenn Aurora neue Standard-Nebenversionen veröffentlicht. In der Regel veröffentlicht Aurora zwei Standard-Nebenversionen für jede Hauptversion pro Jahr. Das folgende Bash-Shell-Skript zeigt die Standard-Nebenversionen für eine Reihe von Aurora-PostgreSQL-Hauptversionen (einschließlich Beispielausgabe).

```
for major in 16 15 14 13 12 11; do
  echo -n "Default for Aurora PostgreSQL major version $major: "
  aws rds describe-db-engine-versions \
    --engine aurora-postgresql \
    --engine-version "$major" \
    --default-only \
    --query 'DBEngineVersions[0].EngineVersion' \
```

```
    --output text
done

Default for Aurora PostgreSQL major version 16: 16.4
Default for Aurora PostgreSQL major version 15: 15.8
Default for Aurora PostgreSQL major version 14: 14.13
Default for Aurora PostgreSQL major version 13: 13.16
Default for Aurora PostgreSQL major version 12: 12.20
Default for Aurora PostgreSQL major version 11: 11.21
```

Wenn Sie automatische Nebenversions-Upgrades für Ihren Aurora-DB-Cluster aktivieren, verwendet Aurora entweder die Standard-Nebenversion oder eine neuere Nebenversion für eine bestimmte Hauptversion. Wenn die Standard-Nebenversion für Aurora PostgreSQL 15 beispielsweise 15.8 ist und die neuere Version 15.10 ebenfalls zur Verfügung steht, kann Aurora automatisch entweder auf 15.8 oder 15.10 aktualisieren.

Amazon-Aurora-Hauptversionen

Aurora-Versionen verwenden das Schema *major*.*minor*.*patch*. Eine Aurora-Hauptversion bezieht sich auf die MySQL- oder PostgreSQL-Community-Hauptversion, mit der Aurora kompatibel ist. Die Hauptversionen von Aurora MySQL und Aurora PostgreSQL stehen mindestens bis zum Ende des Lebenszyklus der Community für die entsprechende Community-Version zur Verfügung. Gegen eine Gebühr können Sie eine Hauptversion auch nach Ablauf des Standard-Supports von Aurora weiter ausführen. Weitere Informationen dazu finden Sie unter Amazon RDS Extended Support mit Amazon Aurora und Amazon Aurora – Preise.

Weitere Informationen zu Hauptversionen und dem Veröffentlichungskalender für Aurora MySQL und Aurora PostgreSQL finden Sie auf den folgenden Seiten in den jeweiligen Versionshinweisen:

- Veröffentlichungskalender für Aurora-MySQL-Hauptversionen
- Veröffentlichungskalender für Aurora-PostgreSQL-Hauptversionen

Sie können auch Informationen zu den Supportdaten für die Engine-Hauptversionen anzeigen, indem Sie den AWS CLI-Befehl describe-db-major-engine-versions ausführen oder die RDS-API-Operation DescribeDBMajorEngineVersions verwenden.

> **Note**
>
> Der erweiterte Amazon-RDS-Support für Aurora-MySQL-Version 2 beginnt am 1. November 2024, wird Ihnen jedoch erst am 1. Dezember 2024 in Rechnung gestellt.

Zwischen dem 1. und 30. November 2024 sind alle DB-Cluster von Aurora-MySQL-Version 2 durch den erweiterten Amazon-RDS-Support abgedeckt. Weitere Informationen finden Sie unter Erweiterter Amazon-RDS-Support für ausgewählte Aurora-Versionen.

Wie lange bleiben Amazon-Aurora-Hauptversionen verfügbar?

Die Hauptversionen von Amazon Aurora stehen mindestens bis zum Ende des Lebenszyklus der Community für die entsprechende Community-Version zur Verfügung. Sie können die Daten des Ablaufs des Standard-Supports für Aurora verwenden, um Ihre Test- und Upgrade-Zyklen zu planen. Diese Daten stellen das früheste Datum dar, an dem ein Upgrade auf eine neuere Version erforderlich sein könnte. Weitere Informationen zu den Daten finden Sie unter Amazon-Aurora-Hauptversionen.

Bevor Aurora Sie bittet, auf eine neuere Hauptversion zu aktualisieren und Ihnen bei der Planung hilft, erhalten Sie mindestens 12 Monate im Voraus eine Erinnerung. Die Erinnerungen kommunizieren die folgenden Hinweise zum Upgrade-Prozess.

- Den Zeitpunkt bestimmter Meilensteine
- Die Auswirkungen auf Ihre DB-Cluster
- Empfohlene Aktionen

Es wird empfohlen, Ihre Anwendungen mit neuen Datenbankversionen gründlich zu testen, bevor Sie Ihren Cluster auf eine neue Hauptversion aktualisieren.

Wenn die Hauptversion das Ende des Standard-Supports von Aurora erreicht hat, wird jeder DB-Cluster, auf dem noch die frühere Version ausgeführt wird, während eines geplanten Wartungsfensters automatisch auf eine Version mit erweitertem Support aktualisiert. Es können Gebühren für erweiterten Support anfallen. Weitere Informationen zum erweiterten Amazon-RDS-Support finden Sie unter Verwenden des erweiterten Amazon-RDS-Supports.

Amazon-Aurora-Nebenversionen

Aurora-Versionen verwenden das Schema *major*.*minor*.*patch*. Eine Aurora-Nebenversion bietet inkrementelle Community- und Aurora-spezifische Verbesserungen des Services, zum Beispiel neue Funktionen und Fehlerkorrekturen.

Weitere Informationen zu Nebenversionen und dem Veröffentlichungskalender für Aurora MySQL und Aurora PostgreSQL finden Sie auf den folgenden Seiten in den jeweiligen Versionshinweisen:

- Veröffentlichungskalender für Aurora-MySQL-Nebenversionen
- Veröffentlichungskalender für Aurora-PostgreSQL-Nebenversionen

In den folgenden Abschnitten werden Details zur Frequenz und Lebensdauer beschrieben, die Sie bei Aurora-Nebenversionen erwarten können.

Themen

- Wie oft werden Amazon-Aurora-Nebenversionen veröffentlicht?
- Wie lange bleiben Amazon-Aurora-Nebenversionen verfügbar

Wie oft werden Amazon-Aurora-Nebenversionen veröffentlicht?

Im Allgemeinen werden Amazon-Aurora-Nebenversionen vierteljährlich veröffentlicht. Der Release-Zeitplan kann variieren, um zusätzliche Funktionen oder Korrekturen aufzunehmen.

Wie lange bleiben Amazon-Aurora-Nebenversionen verfügbar

In der Regel macht Aurora jede Amazon-Aurora-Nebenversion einer bestimmten Hauptversion für mindestens 12 Monate verfügbar. Am Ende dieses Zeitraums kann Aurora Ihre Datenbank automatisch auf die Standard-Nebenversion oder eine spätere Version aktualisieren. Aurora startet das Upgrade während des geplanten Wartungsfensters für jeden DB-Cluster, auf dem die frühere Nebenversion ausgeführt wird.

In einigen Fällen kann Aurora eine Nebenversion einer bestimmten Hauptversion früher als in der üblichen 12-Monats-Frist ersetzen. Gründe können kritische Sicherheitsprobleme oder das Datum des Support-Endes für eine Hauptversion sein.

Bevor Sie mit automatischen Upgrades von Nebenversionen beginnen, die sich dem Ende ihrer Lebensdauer nähern, erinnert Aurora Sie in der Regel drei Monate im Voraus daran. Aurora gibt die folgenden Hinweise zum Upgrade-Prozess.

- Den Zeitpunkt bestimmter Meilensteine
- Die Auswirkungen auf Ihre DB-Cluster
- Empfohlene Aktionen

Benachrichtigungen mit einer Vorlaufzeit von weniger als drei Monaten beschreiben kritische Angelegenheiten wie Sicherheitsprobleme, die ein schnelleres Handeln erfordern.

Wenn die Einstellung Kleinere Versions-Upgrades automatisch aktivieren aktiviert ist, erhalten Sie eine Erinnerung, aber keine RDS-Ereignisbenachrichtigung. Aurora aktualisiert Ihre Datenbank innerhalb eines Wartungsfensters, nachdem die vorgeschriebene Upgrade-Frist abgelaufen ist.

Wenn die Einstellung Kleinere Versions-Upgrades automatisch aktivieren nicht aktiviert ist, erhalten Sie eine Erinnerung und ein Ereignis des DB-Clusters von Amazon RDS mit der Kategorie `maintenance` und der ID `RDS-EVENT-0156`. Aurora aktualisiert Ihre Datenbank innerhalb des nächsten Wartungsfensters.

Beachten Sie, dass nach dem Ende des Standard-Supports von Aurora für eine Nebenversion keine weiteren Patchversionen für diese Nebenversion veröffentlicht werden. Um kritische Bugfixes oder CVEs zu erhalten, müssen Sie ein Upgrade auf eine Nebenversion mit Standard-Support durchführen.

Weitere Informationen zu automatischen Nebenversions-Upgrades finden Sie unter Automatische Nebenversions-Upgrades für Aurora-DB-Cluster.

Amazon-Aurora-Patchversionen

Aurora-Versionen verwenden das Schema *`major.minor.patch`*. Eine Aurora-Patchversion enthält wichtige Fehlerkorrekturen, die einer Nebenversion nach ihrer ersten Veröffentlichung hinzugefügt wurden (z. B. Aurora MySQL 3.04.0, 3.04.1, ..., 3.04.3). Während jede neue Nebenversion neue Aurora-Funktionen bietet, werden neue Patch-Versionen innerhalb einer bestimmten Nebenversion hauptsächlich verwendet, um wichtige Probleme zu beheben.

Weitere Informationen zu Patch-Vorgängen finden Sie unter Warten eines Amazon Aurora-DB-Clusters.

Upgrade von Amazon-Aurora-DB-Clustern

Mit Amazon Aurora können Sie Upgrades für Ihre DB-Cluster steuern und testen. Amazon Aurora bietet Optionen für automatische Nebenversions-Upgrades, manuelle Upgrade-Steuerung, erforderliche Upgrades und Pre-Upgrade-Tests. Sie können Ihre Cluster mit der aktuellen Nebenversion auf dem neuesten Stand halten, unkritische Upgrades zurückstellen, Upgrades bei schwerwiegenden Problemen erzwingen und das Upgrade-Verhalten in Nicht-Produktionsumgebungen überprüfen. In den folgenden Abschnitten wird beschrieben, wie Aurora-DB-Cluster-Upgrades mithilfe dieser Funktionen verwaltet und getestet werden.

Automatische Nebenversions-Upgrades für Aurora

Durch automatische Nebenversions-Upgrades wird Ihre Datenbank regelmäßig auf die neuesten Datenbank-Engine-Versionen aktualisiert. Das Upgrade beinhaltet jedoch möglicherweise nicht immer die aktuelle Datenbank-Engine-Version. Wenn Sie Ihre Datenbanken zu bestimmten Zeiten auf bestimmten Versionen belassen müssen, empfehlen wir Ihnen, ein manuelles Upgrade auf die Datenbankversionen durchzuführen, die Sie gemäß Ihrem erforderlichen Zeitplan benötigen. Bei kritischen Sicherheitsproblemen oder wenn für eine Version das Datum des Support-Endes erreicht ist, kann Amazon Aurora ein Nebenversions-Upgrade durchführen, auch wenn Sie die Option Kleinere Versions-Upgrades automatisch aktivieren nicht aktiviert haben. Weitere Informationen finden Sie in der Upgrade-Dokumentation zu Ihrer spezifischen Datenbank-Engine.

Siehe Upgrade der Nebenversion oder der Patch-Ebene eines Aurora MySQL-DB-Clusters und Durchführen eines Nebenversions-Upgrades.

Sie können mit Aurora-Nebenversionen auf dem Laufenden bleiben, indem Sie für jede DB-Instance im Aurora-Cluster das automatische Upgrade der Nebenversion aktivieren. Aurora führt das automatische Upgrade nur durch, wenn alle DB-Instances in Ihrem Cluster diese Einstellung aktiviert haben.

Wenn Kleinere Versions-Upgrades automatisch aktivieren für Ihren DB-Cluster auf Ja gesetzt ist, führt Aurora automatisch ein Upgrade auf die Standard-Nebenversion oder auf eine neuere Nebenversion durch. Wenn beispielsweise die Standard-Nebenversion für Aurora PostgreSQL 15 die Version 15.8 ist und Version 15.10 existiert, könnte das Ziel für das automatische Upgrade entweder 15.8 oder 15.10 sein.

Normalerweise plant Aurora zweimal im Jahr automatische Upgrades für DB-Cluster, bei denen das automatische Nebenversions-Upgrade aktiviert ist. Diese Upgrades werden während des Wartungsfensters durchgeführt, das Sie für den Cluster angeben. Weitere Informationen finden Sie unter Automatische Nebenversions-Upgrades für Aurora-DB-Cluster.

Automatische Upgrades von Nebenversionen werden im Voraus über ein Amazon-RDS-DB-Clusterereignis mit der Kategorie `maintenance` und der ID `RDS-EVENT-0156` kommuniziert. Weitere Informationen finden Sie unter Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora.

Manuelles Steuern von Upgrades Ihrer DB-Cluster auf neue Versionen

Wenn Sie die Einstellung Kleinere Versions-Upgrades automatisch aktivieren aktiviert haben, aktualisiert Aurora Ihren DB-Cluster automatisch auf die Standard-Nebenversion oder eine neuere

Nebenversion. Normalerweise plant Aurora zweimal im Jahr automatische Upgrades für DB-Cluster, bei denen die Einstellung Kleinere Versions-Upgrades automatisch aktivieren aktiviert ist. Diese Upgrades werden während kundenspezifischer Wartungsfenster gestartet.

Deaktivieren Sie zum Ausschalten der automatischen Nebenversions-Upgrades Kleinere Versions-Upgrades automatisch aktivieren auf einer beliebigen DB-Instance innerhalb eines Aurora-Clusters. Aurora führt nur dann ein automatisches Nebenversions-Upgrade durch, wenn alle DB-Instances in Ihrem Cluster die Einstellung aktiviert haben.

> **Note**
>
> Bei obligatorischen Upgrades wie dem Ende der Lebensdauer einer Nebenversion aktualisiert Aurora den DB-Cluster, auch wenn die Einstellung Kleinere Versions-Upgrades automatisch aktivieren deaktiviert ist. Sie erhalten eine Erinnerung, aber keine RDS-Ereignisbenachrichtigung. Aurora aktualisiert Ihren Cluster innerhalb eines Wartungsfensters, nachdem die vorgeschriebene Upgrade-Frist abgelaufen ist.

Da Upgrades von Hauptversionen ein gewisses Kompatibilitätsrisiko beinhalten, werden sie nicht automatisch ausgeführt. Sie müssen diese initiieren, außer im Fall der Einstellung einer Hauptversion. Es wird empfohlen, Ihre Anwendungen mit neuen Datenbankversionen gründlich zu testen, bevor Sie Ihren Cluster auf eine Hauptversion aktualisieren.

Weitere Informationen zum Aktualisieren eines DB-Clusters auf eine neue Aurora-Hauptversion finden Sie unter Upgrade von Amazon Aurora MySQL-DB-Clustern und Upgrade von DB-Clustern von Amazon Aurora PostgreSQL.

Erforderliche Amazon-Aurora-Upgrades

Bei bestimmten kritischen Bugfixes führt Aurora möglicherweise ein verwaltetes Upgrade auf eine neuere Patch-Stufe innerhalb derselben Nebenversion durch. In diesem Fall führt Aurora ein Upgrade Ihres Clusters durch, auch wenn Kleinere Versions-Upgrades automatisch aktivieren deaktiviert ist. Zuvor kommuniziert Aurora den detaillierten Upgrade-Prozess. Details umfassen das Timing bestimmter Meilensteine, die Auswirkungen auf Ihre DB-Cluster und empfohlene Aktionen. Solche verwalteten Upgrades werden automatisch innerhalb des Wartungsfensters des Clusters durchgeführt.

Testen Ihres DB-Clusters mit einer neuen Aurora-Version vor dem Upgrade

Sie können den Upgrade-Prozess und die Funktionsweise der neuen Version mit Ihrer Anwendung und Ihrer Workload testen. Verwenden Sie eine der folgenden Methoden:

- Klonen Sie Ihren Cluster mit der Funktion zum schnellen Klonen von Amazon-Aurora-Datenbanken. Führen Sie das Upgrade und alle Tests nach dem Upgrade auf dem neuen Cluster durch.
- Wiederherstellung aus einem Cluster-Snapshot, um einen neuen Aurora-Cluster zu erstellen. Sie können einen Cluster-Snapshot selbst aus einem vorhandenen Aurora-Cluster erstellen. Aurora erstellt außerdem automatisch periodische Snapshots für jeden deiner Cluster. Anschließend können Sie ein Versionsupdate für den neuen Cluster initiieren. Sie können mit der aktualisierten Kopie des Clusters experimentieren, bevor Sie entscheiden, ob Sie den ursprünglichen Cluster aktualisieren möchten.

Weitere Informationen zu diesen Methoden zum Erstellen neuer Cluster zum Testen finden Sie unter Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster und Erstellen eines DB-Cluster-Snapshots.

Support für Amazon-Aurora-Versionen

Wenn Ihr Amazon Aurora DB komplexe Abhängigkeiten von einem bestimmten Verhalten der Datenbank-Engine aufweist, empfehlen wir Ihnen, vor dem Upgrade auf neuere Datenbank-Engine-Versionen umfangreiche Tests durchzuführen. Es gibt Langzeit-Support-Optionen, sodass Sie Ihre DB-Cluster auf ausgewählten Aurora-Engine-Versionen verwalten können, auch wenn diese durch neuere Versionen ersetzt wurden. In den folgenden Abschnitten werden die Langzeit-Support-Optionen für Ihre Aurora-DB-Cluster erläutert.

Langzeit-Support für ausgewählte Amazon-Aurora-Nebenversionen

Für jede Aurora-Hauptversion werden bestimmte Nebenversionen als LTS-Versionen (Long-Term-Support, Langzeit-Support) festgelegt und für mindestens drei Jahre zur Verfügung gestellt. Das heißt, mindestens eine Nebenversion pro Hauptversion wird länger als die typischen 12 Monate zur Verfügung gestellt. Aurora erinnert Sie in der Regel sechs Monate vor Ablauf dieser Frist daran. Aurora teilt die folgenden Hinweise zum Upgrade-Prozess mit.

- Den Zeitpunkt bestimmter Meilensteine
- Die Auswirkungen auf Ihre DB-Cluster

- Empfohlene Aktionen

Benachrichtigungen mit einer Vorlaufzeit von weniger als sechs Monaten kommunizieren kritische Angelegenheiten wie Sicherheitsprobleme, die ein schnelleres Handeln erfordern.

LTS-Nebenversionen enthalten nur wichtige Fehlerkorrekturen (über Patch-Versionen). Eine LTS-Version enthält keine neuen Funktionen, die nach ihrer Einführung veröffentlicht wurden. Einmal im Jahr werden DB-Cluster, die auf einer LTS-Nebenversion ausgeführt werden, auf die neueste Patch-Version der LTS-Version gepatcht. Aurora führt Patches für Ihre Cluster durch, sodass Sie von kumulativen Sicherheits- und Stabilitätskorrekturen profitieren. Wenn kritische Bugfixes angewendet werden müssen, kann Aurora eine LTS-Nebenversion häufiger patchen, z. B. für die Sicherheit.

> **Note**
>
> Um für die Dauer ihres Lebenszyklus auf einer LTS-Nebenversion zu bleiben, stellen Sie sicher, dass Sie das automatische Upgrade der Nebenversion für Ihre DB-Instances deaktivieren. Um zu vermeiden, dass Ihr DB-Cluster automatisch von der LTS-Nebenversion aktualisiert wird, deaktivieren Sie das Kontrollkästchen Automatische Aktualisierung von Unterversionen aktivieren für alle DB-Instances in Ihrem Aurora-Cluster.

Die Versionsnummern aller Aurora LTS-Versionen finden Sie unter Aurora MySQL Long-Term Support- (LTS, Langzeit-Support) Versionen und Verwenden einer Aurora PostgreSQL Long-Term Support (LTS, Langzeit-Support)-Version.

Erweiterter Amazon-RDS-Support für ausgewählte Aurora-Versionen

Mit dem erweiterten Amazon-RDS-Support können Sie Ihre Datenbank auf einer Haupt-Engine-Version gegen Aufpreis über das Ende des Aurora-Standard-Supports hinaus ausführen. Während der Gültigkeit des erweiterten RDS-Supports stellt Amazon RDS Patches für kritische und hohe CVEs bereit, wie in den CVSS-Schweregraden der National Vulnerability Database (NVD) definiert. Weitere Informationen finden Sie unter Amazon RDS Extended Support mit Amazon Aurora.

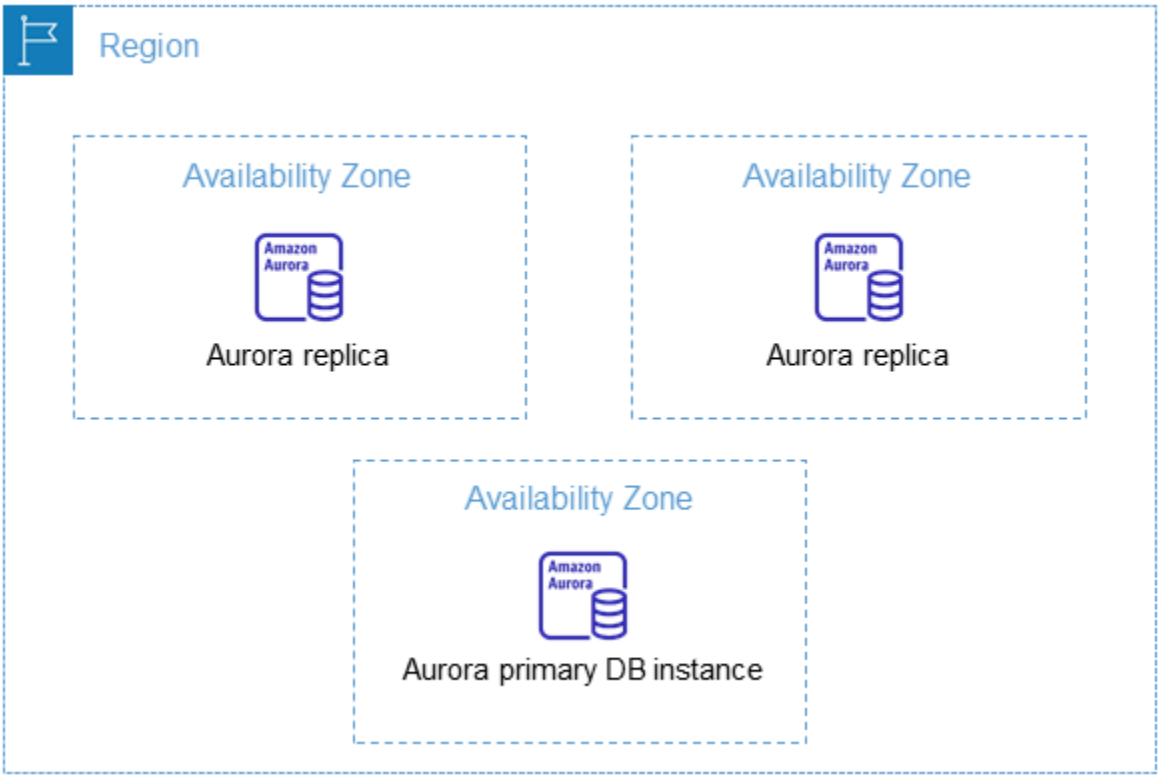
Der erweiterte RDS-Support ist nur für bestimmte Aurora-Versionen verfügbar. Weitere Informationen finden Sie unter Amazon-Aurora-Hauptversionen.

Regionen und Availability Zones

Amazon Cloud Computing-Ressourcen werden an mehreren Standorten weltweit gehostet. Diese Standorte bestehen aus AWS Regionen und Availability Zones. Jede AWS -Region ist ein separater geografischer Bereich. Jede AWS Region hat mehrere isolierte Standorte, die als Availability Zones bezeichnet werden.

> **Note**
>
> Informationen zur Suche nach den Availability Zones für eine AWS Region finden Sie unter Beschreiben Sie Ihre Availability Zones in der EC2 Amazon-Dokumentation.

Amazon betreibt state-of-the-art hochverfügbare Rechenzentren. Obwohl selten, können Fehler auftreten, die sich auf die Verfügbarkeit von DB-Instances auswirken, die sich am selben Speicherort befinden. Wenn Sie alle Ihre DB-Instances an einem Ort hosten, der von einem solchen Ausfall betroffen ist, ist keine Ihrer DB-Instances verfügbar.



Es ist wichtig, sich daran zu erinnern, dass jede AWS Region völlig unabhängig ist. Jede von Ihnen initiierte Amazon RDS-Aktivität (z. B. das Erstellen von Datenbank-Instances

oder das Auflisten verfügbarer Datenbank-Instances) wird nur in Ihrer aktuellen AWS Standardregion ausgeführt. Die AWS Standardregion kann in der Konsole oder durch Einstellen der `AWS_DEFAULT_REGION`Umgebungsvariablen geändert werden. Oder sie kann überschrieben werden, indem der `--region` Parameter mit der AWS Command Line Interface ()AWS CLI verwendet wird. Weitere Informationen finden Sie unter Konfigurieren der AWS Command Line Interface, insbesondere in den Abschnitten zu Umgebungsvariablen und zu den Befehlszeilenoptionen.

Amazon RDS unterstützt spezielle AWS Regionen namens AWS GovCloud (US). Diese wurden entwickelt, damit US-Regierungsbehörden und Kunden vertrauliche Workloads in die Cloud verschieben können. Die Regionen AWS GovCloud (US) gehen auf die spezifischen regulatorischen und Compliance-Anforderungen der US-Regierung ein. Weitere Informationen finden Sie unter Was ist AWS GovCloud (US)?

Um eine Amazon RDS-DB-Instance in einer bestimmten AWS Region zu erstellen oder mit ihr zu arbeiten, verwenden Sie den entsprechenden regionalen Service-Endpunkt.

> **Note**
>
> Aurora unterstützt keine Local Zones.

AWS Regionen

Jede AWS Region ist so konzipiert, dass sie von den anderen AWS Regionen isoliert ist. Dieser Entwurf sorgt für die größtmögliche Fehlertoleranz und Stabilität.

Wenn Sie Ihre Ressourcen anzeigen, sehen Sie nur die Ressourcen, die mit der von Ihnen angegebenen AWS Region verknüpft sind. Das liegt daran, dass AWS Regionen voneinander isoliert sind und wir Ressourcen nicht automatisch regionsübergreifend AWS replizieren.

Verfügbarkeit in Regionen

Wenn Sie mit einem Aurora-DB-Cluster über die Befehlszeilenschnittstelle oder API-Vorgänge arbeiten, stellen Sie sicher, dass Sie den regionalen Endpunkt angeben.

Themen

- Aurora MySQL-Verfügbarkeit in Regionen
- Aurora PostgreSQL-Verfügbarkeit in Regionen

Aurora MySQL-Verfügbarkeit in Regionen

Die folgende Tabelle zeigt die AWS Regionen, in denen Aurora MySQL derzeit verfügbar ist, und den Endpunkt für jede Region.

Name der Region	Region	Endpunkt	Protocol (Protokoll)	
USA Ost (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
USA Ost (Nord-Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
USA West (Nordkalifornien)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
USA West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
Afrika (Kapstadt)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
Asien-Pazifik (Hongkong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asien-Pazifik (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Asien-Pazifik (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
Asien-Pazifik (Malaysia)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS
Asien-Pazifik (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS
Asien-Pazifik (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
Asien-Pazifik (Neuseeland)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS
Asien-Pazifik (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
Asien-Pazifik (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)	
Asien-Pazifik (Singapur)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
Asien-Pazifik (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
Asien-Pazifik (Taipeh)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS	
Asien-Pazifik (Thailand)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS	
Asien-Pazifik (Tokio)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
Kanada (Zentral)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
Kanada West (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
Europa (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
Europa (Irland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Europa (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
Europa (Mailand)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
Europa (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
Europa (Spanien)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
Europa (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
Europa (Zürich)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
Mexiko (Zentral)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS
Naher Osten (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
Naher Osten (VAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)	
Südamerika (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud (US-Ost)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Aurora PostgreSQL-Verfügbarkeit in Regionen

Die folgende Tabelle zeigt die AWS Regionen, in denen Aurora PostgreSQL derzeit verfügbar ist, und den Endpunkt für jede Region.

Name der Region	Region	Endpunkt	Protocol (Protokoll)	
USA Ost (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
USA Ost (Nord-Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
USA West (Nordkalifornien)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	

Name der Region	Region	Endpunkt	Protocol (Protokoll)
USA West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
Afrika (Kapstadt)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
Asien-Pazifik (Hongkong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
Asien-Pazifik (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
Asien-Pazifik (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
Asien-Pazifik (Malaysia)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS
Asien-Pazifik (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Asien-Pazifik (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
Asien-Pazifik (Neuseeland)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS
Asien-Pazifik (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
Asien-Pazifik (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
Asien-Pazifik (Singapur)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
Asien-Pazifik (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
Asien-Pazifik (Taipeh)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS
Asien-Pazifik (Thailand)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Asien-Pazifik (Tokio)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
Kanada (Zentral)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
Kanada West (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
Europa (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
Europa (Irland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
Europa (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
Europa (Mailand)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
Europa (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
Europa (Spanien)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
Europa (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Europa (Zürich)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
Mexiko (Zentral)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS
Naher Osten (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
Naher Osten (VAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS
Südamerika (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-Ost)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS

Availability Zones

Eine Availability Zone ist ein isolierter Standort in einer bestimmten AWS-Region. Jede Region verfügt über mehrere Availability Zones (AZ), die eine hohe Verfügbarkeit für die Region bieten. Eine

AZ wird durch den AWS Regionalcode gefolgt von einer Buchstabenkennung (z. B.`us-east-1a`) identifiziert. Wenn Sie die VPC und Subnetze erstellen, anstatt die Standard-VPC zu verwenden, definieren Sie jedes Subnetz in einer bestimmten AZ. Wenn Sie einen Aurora-DB-Cluster erstellen, erstellt Aurora die primäre Instance in einem der Subnetze in der DB-Subnetzgruppe der VPC. Auf diese Weise wird diese Instance mit einer von Aurora ausgewählten AZ verknüpft.

Jeder Aurora-DB-Cluster hostet Kopien seines Speichers in drei separaten, von Aurora automatisch AZs ausgewählten Kopien aus der Subnetzgruppe AZs in Ihrer DB-Subnetzgruppe. Jede DB-Instance im Cluster muss sich in einer dieser drei AZs Instanzen befinden.

Wenn Sie eine DB-Instance in Ihrem Cluster erstellen, wählt Aurora automatisch eine geeignete AZ für diese Instance, wenn Sie keine AZ angeben.

Verwenden Sie den describe-availability-zones EC2 Amazon-Befehl wie folgt, um die Availability Zones innerhalb der angegebenen Region zu beschreiben, die für Ihr Konto aktiviert sind.

```
aws ec2 describe-availability-zones --region region-name
```

Wenn Sie beispielsweise die Availability Zones in der Region USA Ost (Nord-Virginia) (us-east-1) beschreiben möchten, die für Ihr Konto aktiviert sind, führen Sie den folgenden Befehl aus:

```
aws ec2 describe-availability-zones --region us-east-1
```

Informationen dazu, wie Sie die AZ angeben, wenn Sie einen Cluster erstellen oder Instances hinzufügen, finden Sie unter Konfigurieren des Netzwerks für den DB-Cluster.

Lokale Zeitzone für Amazon Aurora-DB-Cluster

Standardmäßig ist die Zeitzone für ein Amazon Aurora-DB-Cluster auf die Universal Time Coordinated (UTC) eingestellt. Sie können die Zeitzone für Instances in Ihrem DB-Cluster stattdessen auf die lokale Zeitzone Ihrer Anwendung festlegen.

Legen Sie den Zeitzonenparameter auf einen der unterstützten Werte fest, um die lokale Zeitzone für einen DB-Cluster anzugeben. Diesen Parameter legen Sie in der Cluster-Parametergruppe für Ihren DB-Cluster fest.

- Für Aurora MySQL lautet der Name dieses Parameters `time_zone`. Informationen zu bewährten Methoden zum Festlegen des Parameters `time_zone` finden Sie unter Optimierung von Zeitstempeloperationen.
- Für Aurora PostgreSQL lautet der Name dieses Parameters `timezone`.

Wenn Sie den Parameter für ein DB-Cluster setzen, wird bei allen Instances in diesem DB-Cluster die neue lokale Zeitzone eingestellt. In einigen Fällen verwenden andere Aurora-DB-Cluster möglicherweise dieselbe Cluster-Parametergruppe. In diesem Fall wird für alle Instances in diesen DB-Clustern die neue lokale Zeitzone eingestellt. Weitere Informationen zu Parametern für Cluster-Ebenen finden Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter.

Nachdem Sie die lokale Zeitzone eingestellt haben, werden alle neuen Verbindungen zur Datenbank die Änderung reflektieren. In einigen Fällen haben Sie möglicherweise offene Verbindungen mit Ihrer Datenbank, wenn Sie die lokale Zeitzone ändern. In diesem Fall sehen Sie die aktualisierte lokale Zeitzone nicht, nachdem Sie die Verbindung schließen und eine neue Verbindung öffnen.

Wenn Sie regionsübergreifend AWS replizieren, verwenden der Replikationsquell-DB-Cluster und das Replikat unterschiedliche Parametergruppen. Parametergruppen sind für eine AWS Region einzigartig. Damit Sie dieselbe lokale Zeitzone für jede Instance verwenden können, müssen Sie den Zeitzonenparameter in den Parametergruppen für die Replikationsquelle und für das Replikat festlegen.

Wenn Sie ein DB-Cluster aus einem DB-Cluster-Snapshot wiederherstellen, wird die lokale Zeitzone auf die Zeitzone UTC eingestellt. Sie können die Zeitzone auf Ihre lokale Zeitzone einstellen, nachdem die Wiederherstellung abgeschlossen ist. In einigen Fällen führen Sie möglicherweise eine zeitpunktbezogen Wiederherstellung eines DB-Clusters durch. In diesem Fall entspricht die lokale Zeitzone für den wiederhergestellten DB-Cluster der Zeitzoneneinstellung aus der Parametergruppe des wiederhergestellten DB-Clusters.

In der folgenden Tabelle sind einige Werte aufgeführt, die Sie für Ihre lokale Zeitzone festlegen können. Um alle verfügbaren Zeitzonen aufzulisten, können Sie die folgenden SQL-Abfragen verwenden:

- Aurora MySQL: `select * from mysql.time_zone_name;`
- Aurora PostgreSQL: `select * from pg_timezone_names;`

> **Note**
>
> Für einige Zeitzonen können Zeitwerte für bestimmte Datumsbereiche falsch gemeldet werden, worauf in der Tabelle hingewiesen wird. Für Zeitzonen in Australien wird die Zeitzonenabkürzung mit einem veralteten Wert zurückgegeben, worauf in der Tabelle hingewiesen wird.

Zeitzone	Hinweise
`Africa/Harare`	Diese Zeitzoneneinstellung kann vom 28. Februar 1903 21:49:40 GMT bis zum 28. Februar 1903 21:55:48 GMT falsche Werte zurückgeben.
`Africa/Monrovia`	
`Africa/Nairobi`	Diese Zeitzoneneinstellung kann vom 31. Dezember 1939 21:30:00 GMT bis zum 31. Dezember 1959 21:15:15 GMT falsche Werte zurückgeben.
`Africa/Windhoek`	
`America/Bogota`	Diese Zeitzoneneinstellung kann vom 23. November 1914 04:56:16 GMT bis zum 23. November 1914 04:56:20 GMT falsche Werte zurückgeben.
`America/Caracas`	
`America/Chihuahua`	
`America/Cuiaba`	
`America/Denver`	
`America/Fortaleza`	In einigen Fällen wird bei einem DB-Cluster in der Region Südamerika (São Paulo) die Zeit für eine kürzlich geänderte Zeitzone Brasiliens nicht korrekt angezeigt. Setzen Sie in diesem Fall den Zeitzonenparameter des DB-Clusters auf `America/Fortaleza` zurück.
`America/Guatemala`	
`America/Halifax`	Diese Zeitzoneneinstellung kann vom 27. Oktober 1918 05:00:00 GMT bis zum 31. Oktober 1918 05:00:00 GMT falsche Werte zurückgeben.
`America/Manaus`	Wenn sich Ihr DB-Cluster in der Zeitzone Südamerika (Cuiaba) befindet und die erwartete Zeit für die kürzlich geänderte Zeitzone in Brasilien nicht korrekt angezeigt wird, setzen Sie den Zeitzonenparameter des DB-Clusters auf `America/Manaus` zurück.

Zeitzone	Hinweise
America/Matamoros	
America/Monterrey	
America/Montevideo	
America/Noronha	
America/Phoenix	
America/Tijuana	
Asia/Ashgabat	
Asia/Baghdad	
Asia/Baku	
Asia/Bangkok	
Asia/Beirut	
Asia/Calcutta	
Asia/Kabul	
Asia/Karachi	
Asia/Kathmandu	
Asia/Muscat	Diese Zeitzoneneinstellung kann vom 31. Dezember 1919 20:05:36 GMT bis zum 31. Dezember 1919 20:05:40 GMT falsche Werte zurückgeben.
Asia/Riyadh	Diese Zeitzoneneinstellung kann vom 13. März 1947 20:53:08 GMT bis zum 31. Dezember 1949 20:53:08 GMT falsche Werte zurückgeben.

Zeitzone	Hinweise
Asia/Seoul	Diese Zeitzoneneinstellung kann vom 30. November 1904 15:30:00 GMT bis zum 07. September 1945 15:00:00 GMT falsche Werte zurückgeben.
Asia/Shanghai	Diese Zeitzoneneinstellung kann vom 31. Dezember 1927 15:54:08 GMT bis zum 02. Juni 1940 16:00:00 GMT falsche Werte zurückgeben.
Asia/Singapore	
Asia/Taipei	Diese Zeitzoneneinstellung kann vom 30. September 1937 16:00:00 GMT bis zum 29. September 1979 15:00:00 GMT falsche Werte zurückgeben.
Asia/Tehran	
Asia/Tokyo	Diese Zeitzoneneinstellung kann vom 30. September 1937 15:00:00 GMT bis zum 31. Dezember 1937 15:00:00 GMT falsche Werte zurückgeben.
Asia/Ulaanbaatar	
Atlantic/Azores	Diese Zeitzoneneinstellung kann vom 24. Mai 1911 01:54:32 GMT bis zum 01. Januar 1912 01:54:32 GMT falsche Werte zurückgeben.
Australia/Adelaide	Die Abkürzung für diese Zeitzone wird als CST anstatt als ACDT/ACST zurückgegeben.
Australia/Brisbane	Die Abkürzung für diese Zeitzone wird als EST anstatt als AEDT/AEST zurückgegeben.
Australia/Darwin	Die Abkürzung für diese Zeitzone wird als CST anstatt als ACDT/ACST zurückgegeben.
Australia/Hobart	Die Abkürzung für diese Zeitzone wird als EST anstatt als AEDT/AEST zurückgegeben.
Australia/Perth	Die Abkürzung für diese Zeitzone wird als WST statt als AWST AWDT/ zurückgegeben.

Zeitzone	Hinweise
Australia/Sydney	Die Abkürzung für diese Zeitzone wird als EST anstatt als AEDT/AEST zurückgegeben.
Brazil/East	
Canada/Saskatchewan	Diese Zeitzoneneinstellung kann vom 27. Oktober 1918 08:00:00 GMT bis zum 31. Oktober 1918 08:00:00 GMT falsche Werte zurückgeben.
Europe/Amsterdam	
Europe/Athens	
Europe/Dublin	
Europe/Helsinki	Diese Zeitzoneneinstellung kann vom 30. April 1921 22:20:08 GMT bis zum 30. April 1921 22:20:11 GMT falsche Werte zurückgeben.
Europe/Paris	
Europe/Prague	
Europe/Sarajevo	
Pacific/Auckland	
Pacific/Guam	
Pacific/Honolulu	Diese Zeitzoneneinstellung kann vom 21. Mai 1933 11:30:00 GMT bis zum 30. September 1945 11:30:00 GMT falsche Werte zurückgeben.
Pacific/Samoa	Diese Zeitzoneneinstellung kann vom 01. Januar 1911 11:22:48 GMT bis zum 01. Januar 1950 11:30:00 GMT falsche Werte zurückgeben.
US/Alaska	
US/Central	

Zeitzone	Hinweise
US/Eastern	
US/East-Indiana	
US/Pacific	
UTC	

Unterstützte Funktionen in Amazon Aurora by AWS-Region und der Aurora-DB-Engine

Aurora MySQL- und PostgreSQL-kompatible Datenbank-Engines unterstützen mehrere Amazon-RDS- und Amazon Aurora-Funktionen und Optionen. Der Support variiert zwischen bestimmten Versionen der einzelnen Datenbank-Engines und in allen AWS-Regionen. Um die Versionsunterstützung und Verfügbarkeit der Aurora-Datenbank-Engine für eine bestimmte Funktion zu ermitteln AWS-Region, können Sie die folgenden Abschnitte verwenden.

Einige dieser Funktionen sind haben nur Aurora-Fähigkeiten. Beispielsweise werden Aurora Serverless, globale Aurora-Datenbanken und die Unterstützung für die Integration mit AWS Machine Learning-Services von nicht Amazon RDS unterstützt. Andere Funktionen, wie zum Beispiel Amazon-RDS-Proxy, werden von Amazon Aurora und Amazon RDS unterstützt.

Unterstützte Regionen und DB-Engines

- Tabellenkonventionen
- Unterstützte Regionen und Aurora-DB-Engines für Blau/Grün-Bereitstellungen
- Unterstützte Regionen und Aurora-DB-Engines für Cluster-Speicherkonfigurationen
- Unterstützte Regionen und Aurora-DB-Engines für Datenbankaktivitäts-Streams
- Unterstützte Regionen und Aurora-DB-Engines zum Exportieren von Cluster-Daten nach Amazon S3
- Unterstützte Regionen und Aurora-DB-Engines zum Exportieren von Snapshot-Daten nach Amazon S3
- Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken
- Unterstützte Regionen und Aurora-DB-Engines für die IAM-Datenbankauthentifizierung
- Unterstützte Regionen und Aurora-DB-Engines für die Kerberos-Authentifizierung
- Unterstützte Regionen und DB-Engines für Aurora Machine Learning
- Unterstützte Regionen und Aurora-DB-Engines für Performance Insights
- Unterstützte Regionen und Aurora-DB-Engines für Null-ETL-Integrationen
- Unterstützte Regionen und Aurora-DB-Engines für Amazon-RDS-Proxy
- Unterstützte Regionen und Aurora-DB-Engines für die Integration von Secrets Manager
- Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2
- Aurora Serverless v1

- Unterstützte Regionen und Aurora-DB-Engines für RDS-Daten-API
- Unterstützte Regionen und Aurora-DB-Engines für Zero-Downtime-Patching (ZDP)
- Unterstützte Regionen in Aurora PostgreSQL Limitless Database
- Unterstützte Regionen und DB-Engines für native Features der Aurora-Engine

Tabellenkonventionen

In den Tabellen werden die folgenden Muster verwendet, um Versionsnummern und den Grad der Unterstützung anzugeben:

- Version x.y – Die spezifische Version allein wird unterstützt.
- Version x.y und höher – Die angegebene Version und alle höheren Unterversionen der jeweiligen Hauptversion werden unterstützt. Zum Beispiel bedeutet „Version 10.11 und höher“, dass die Versionen 10.11, 10.11.1 und 10.12 unterstützt werden.

Unterstützte Regionen und Aurora-DB-Engines für Blau/Grün-Bereitstellungen

Bei einer blue/green Bereitstellung wird eine Produktionsdatenbankumgebung in eine separate, synchronisierte Staging-Umgebung kopiert. Mithilfe von Amazon RDS Blue/Green Deployments können Sie Änderungen an der Datenbank in der Staging-Umgebung vornehmen, ohne die Produktionsumgebung zu beeinträchtigen. Sie können beispielsweise die Haupt- oder Nebenversion der DB-Engine aktualisieren, Datenbankparameter ändern oder Schemaänderungen in der Staging-Umgebung vornehmen. Wenn Sie bereit sind, können Sie die Staging-Umgebung zur neuen Produktionsdatenbankumgebung hochstufen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

Blau/Grün-Bereitstellungen mit Aurora MySQL

Die Blue/Green Bereitstellungsfunktion ist für alle Versionen von Aurora MySQL insgesamt verfügbar AWS-Regionen, einschließlich Aurora MySQL-Cluster, die als Aurora Global Database konfiguriert sind.

Blau/Grün-Bereitstellungen mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für Blue/Green Bereitstellungen mit Aurora PostgreSQL verfügbar, einschließlich Aurora PostgreSQL-Cluster, die als Aurora Global Database konfiguriert sind.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Alle AWS-Regionen	Version 17.4 und höher	Version 16.1 und höher	Version 15.4 und höher	Version 14.9 und höher	Version 13.12 und höher	Version 12.16 und höher	Version 11.21 und höher

Unterstützte Regionen und Aurora-DB-Engines für Cluster-Speicherkonfigurationen

Amazon Aurora bietet zwei Speicherkonfigurationen für DB-Cluster, Aurora I/O-Optimized und Aurora Standard. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.

Aurora I/O-Optimized

Aurora I/O-Optimized ist in allen AWS-Regionen für die folgenden Amazon-Aurora-Versionen verfügbar:

- Aurora MySQL Version 3.03.1 und höher
- Aurora-PostgreSQL-Versionen 17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher sowie 13.10 und höher

Aurora Standard

Aurora Standard ist in allen AWS-Regionen für alle Aurora-MySQL- und Aurora-PostgreSQL-Versionen verfügbar.

Unterstützte Regionen und Aurora-DB-Engines für Datenbankaktivitäts-Streams

Durch die Verwendung von Datenbank-Aktivitätsstreams in Aurora können Sie Alarme für die Prüfung von Aktivitäten in Ihrer Aurora-Datenbank überwachen und einstellen. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.

Datenbankaktivitäts-Streams werden im Hinblick auf die folgenden Funktionen nicht unterstützt:

- Aurora Serverless v1
- Aurora Serverless v2
- Babelfish für Aurora PostgreSQL

Themen

- Datenbank-Aktivitätsstreams mit Aurora MySQL
- Datenbank-Aktivitätsstreams mit Aurora PostgreSQL

Datenbank-Aktivitätsstreams mit Aurora MySQL

Die folgenden Regionen und Engine-Versionen sind für Datenbankaktivitäts-Streams mit Aurora MySQL verfügbar.

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
USA Ost (Ohio)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
USA West (Nordkalifornien)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
USA West (Oregon)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Afrika (Kapstadt)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Hongkong)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Hyderabad)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Asien-Pazifik (Jakarta)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Melbourne)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Mumbai)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Seoul)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Singapur)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Sydney)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Kanada (Zentral)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Kanada West (Calgary)	Nicht verfügbar	Nicht verfügbar
China (Peking)	Nicht verfügbar	Nicht verfügbar
China (Ningxia)	Nicht verfügbar	Nicht verfügbar
Europa (Frankfurt)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Europa (Irland)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Europa (London)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Europa (Milan)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Europa (Paris)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Europa (Spain)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Europa (Stockholm)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Europa (Zürich)	Nicht verfügbar	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar
Naher Osten (VAE)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher
Südamerika (São Paulo)	Alle verfügbaren Versionen	Aurora Version 2.11 und höher

Datenbank-Aktivitätsstreams mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für Datenbankaktivitäts-Streams mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
USA Ost (Nord-Virginia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
USA Ost (Ohio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
USA West (Nordkalifornien)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
USA West (Oregon)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Afrika (Kapstadt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Hongkong)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Hyderabad)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Jakarta)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	
Asien-Pazifik (Melbourne)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Mumbai)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Seoul)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Singapur)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Sydney)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Kanada (Zentral)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Kanada West (Calgary)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
China (Peking)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
China (Ningxia)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Europa (Frankfurt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Europa (Irland)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (London)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Europa (Milan)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Europa (Paris)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Europa (Spain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (Stockholm)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Europa (Zürich)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher
Naher Osten (VAE)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Südamerika (São Paulo)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Alle verfügbaren Versionen	Version 11.9 sowie Version 11.13 und höher

Unterstützte Regionen und Aurora-DB-Engines zum Exportieren von Cluster-Daten nach Amazon S3

Sie können DB-Cluster-Daten von Aurora in einen Amazon-S3-Bucket exportieren. Nachdem die Daten exportiert wurden, können Sie die exportierten Daten direkt mit Tools wie Amazon Athena oder Amazon Redshift Spectrum analysieren. Weitere Informationen finden Sie unter Exportieren von DB-Cluster-Daten nach Amazon S3.

Der Export von Clusterdaten nach S3 ist in folgenden AWS-Regionen verfügbar:

- USA Ost (Nord-Virginia)
- USA Ost (Ohio)
- USA West (Nordkalifornien)
- USA West (Oregon)
- Asien-Pazifik (Hongkong)
- Asien-Pazifik (Hyderabad)
- Asien-Pazifik (Jakarta)
- Asien-Pazifik (Melbourne)
- Asien-Pazifik (Mumbai)
- Asien-Pazifik (Neuseeland)
- Asien-Pazifik (Osaka)
- Asien-Pazifik (Seoul)

- Asien-Pazifik (Singapur)
- Asien-Pazifik (Sydney)
- Asien-Pazifik (Tokio)
- Canada (Central)
- Kanada West (Calgary)
- China (Ningxia)
- Europe (Frankfurt)
- Europa (Irland)
- Europa (London)
- Europa (Paris)
- Europa (Spain)
- Europa (Stockholm)
- Europa (Zürich)
- Israel (Tel Aviv)
- Naher Osten (VAE)
- Südamerika (São Paulo)

Themen

- Exportieren von Clusterdaten nach S3 mit Aurora MySQL
- Exportieren von Clusterdaten nach S3 mit Aurora PostgreSQL

Exportieren von Clusterdaten nach S3 mit Aurora MySQL

Alle derzeit verfügbaren Aurora-MySQL-Engine-Versionen unterstützen den Export von DB-Cluster-Daten nach Amazon S3. Weitere Informationen über Versionen finden Sie in den Versionshinweisen für Aurora MySQL.

Exportieren von Clusterdaten nach S3 mit Aurora PostgreSQL

Alle derzeit verfügbaren Aurora-PostgreSQL-Engine-Versionen unterstützen den Export von DB-Cluster-Daten nach Amazon S3. Weitere Informationen zu Versionen finden Sie unter Versionshinweise für Aurora PostgreSQL.

Unterstützte Regionen und Aurora-DB-Engines zum Exportieren von Snapshot-Daten nach Amazon S3

Sie können DB-Cluster-Snapshot-Daten von Aurora in einen Amazon-S3-Bucket exportieren. Sie können manuelle Snapshots und automatisierte System-Snapshots exportieren. Nachdem die Daten exportiert wurden, können Sie die exportierten Daten direkt mit Tools wie Amazon Athena oder Amazon Redshift Spectrum analysieren. Weitere Informationen finden Sie unter Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3.

Das Exportieren von Snapshots nach S3 ist in allen Versionen verfügbar, AWS-Regionen mit Ausnahme der folgenden:

- Asien-Pazifik (Malaysia)
- Asien-Pazifik (Neuseeland)
- Asien-Pazifik (Taipeh)
- Asien-Pazifik (Thailand)
- Mexiko (Zentral)
- AWS GovCloud (US-Ost)
- AWS GovCloud (US-West)

Themen

- Exportieren von Snapshot-Daten nach S3 mit Aurora MySQL
- Exportieren von Snapshot-Daten nach S3 hat Aurora PostgreSQL

Exportieren von Snapshot-Daten nach S3 mit Aurora MySQL

Alle derzeit verfügbaren Aurora-MySQL-Engine-Versionen unterstützen den Export von DB-Cluster-Snapshot-Daten nach Amazon S3. Weitere Informationen über Versionen finden Sie in den Versionshinweisen für Aurora MySQL.

Exportieren von Snapshot-Daten nach S3 hat Aurora PostgreSQL

Alle derzeit verfügbaren Aurora-PostgreSQL-Engine-Versionen unterstützen den Export von DB-Cluster-Snapshot-Daten nach Amazon S3. Weitere Informationen zu Versionen finden Sie unter Versionshinweise für Aurora PostgreSQL.

Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken

Eine globale Aurora-Datenbank ist eine einzelne Datenbank, die sich über mehrere AWS-Regionen Datenbanken erstreckt und globale Lesevorgänge mit geringer Latenz und eine Notfallwiederherstellung nach einem Ausfall in der gesamten Region ermöglicht. Sie bietet integrierte Fehlertoleranz für Ihre Bereitstellung, da sich die DB-Instance nicht auf eine einzelne AWS-Region, sondern auf mehrere Regionen und verschiedene Availability Zones stützt. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.

Themen

- Globale Aurora-Datenbanken mit Aurora MySQL
- Globale Aurora-Datenbanken unterstützen Aurora PostgreSQL

Globale Aurora-Datenbanken mit Aurora MySQL

Die folgenden Regionen und Engine-Versionen sind für globale Aurora-Datenbanken mit Aurora MySQL verfügbar.

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Version 3.01.0 und höher	Version 2.07.0 und höher
USA Ost (Ohio)	Version 3.01.0 und höher	Version 2.07.0 und höher
USA West (Nordkalifornien)	Version 3.01.0 und höher	Version 2.07.0 und höher
USA West (Oregon)	Version 3.01.0 und höher	Version 2.07.0 und höher
Afrika (Kapstadt)	Version 3.01.0 und höher	Version 2.07.1 und höher
Asien-Pazifik (Hongkong)	Version 3.01.0 und höher	Version 2.07.1 und höher
Asien-Pazifik (Hyderabad)	Version 3.02.0 und höher	Version 2.11.2 und höher
Asien-Pazifik (Jakarta)	Version 3.01.0 und höher	Version 2.07.6 und höher
Asien-Pazifik (Malaysia)	Version 3.04.0 und höher	Version 2.07.6 und höher
Asien-Pazifik (Melbourne)	Version 3.03.0 und höher	Version 2.07.6 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Asien-Pazifik (Mumbai)	Version 3.01.0 und höher	Version 2.07.0 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 3.01.0 und höher	Version 2.07.3 und höher
Asien-Pazifik (Seoul)	Version 3.01.0 und höher	Version 2.07.0 und höher
Asien-Pazifik (Singapur)	Version 3.01.0 und höher	Version 2.07.0 und höher
Asien-Pazifik (Sydney)	Version 3.01.0 und höher	Version 2.07.0 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 3.01.0 und höher	Version 2.07.0 und höher
Kanada (Zentral)	Version 3.01.0 und höher	Version 2.07.0 und höher
Kanada West (Calgary)	Version 3.01.0 und höher	Version 2.07.0 und höher
China (Peking)	Version 3.01.0 und höher	Version 2.07.2 und höher
China (Ningxia)	Version 3.01.0 und höher	Version 2.07.2 und höher
Europa (Frankfurt)	Version 3.01.0 und höher	Version 2.07.0 und höher
Europa (Irland)	Version 3.01.0 und höher	Version 2.07.0 und höher
Europa (London)	Version 3.01.0 und höher	Version 2.07.0 und höher
Europa (Milan)	Version 3.01.0 und höher	Version 2.07.1 und höher
Europa (Paris)	Version 3.01.0 und höher	Version 2.07.0 und höher
Europa (Spain)	Version 3.02.0 und höher	Version 2.07.6 und höher
Europa (Stockholm)	Version 3.01.0 und höher	Version 2.07.0 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Europa (Zürich)	Version 3.02.0 und höher	Version 2.07.6 und höher
Israel (Tel Aviv)	Version 3.02.0 und höher	Version 2.07.6 und höher
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Version 3.01.0 und höher	Version 2.07.1 und höher
Naher Osten (VAE)	Version 3.02.0 und höher	Version 2.07.6 und höher
Südamerika (São Paulo)	Version 3.01.0 und höher	Version 2.07.1 und höher
AWS GovCloud (US-Ost)	Version 3.01.0 und höher	Version 2.07.0 und höher
AWS GovCloud (US-West)	Version 3.01.0 und höher	Version 2.07.0 und höher

Globale Aurora-Datenbanken unterstützen Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für globale Aurora-Datenbanken mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
USA Ost (Nord-Virginia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
USA Ost (Ohio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
							11.13 und höher
USA West (Nordkalifornien)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
USA West (Oregon)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Afrika (Kapstadt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Hongkong)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Hyderabad)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Jakarta)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Malaysia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Melbourne)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Mumbai)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Seoul)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Singapur)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Sydney)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Sydney)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Kanada (Zentral)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Kanada West (Calgary)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
China (Peking)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
China (Ningxia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Frankfurt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Irland)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (London)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Milan)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Paris)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Spain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (Stockholm)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Zürich)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Israel (Tel Aviv)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Naher Osten (VAE)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Südamerika (São Paulo)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
AWS GovCloud (US-Ost)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
AWS GovCloud (US-West)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Unterstützte Regionen und Aurora-DB-Engines für die IAM-Datenbankauthentifizierung

Mit der IAM-Datenbankauthentifizierung in Aurora können Sie Ihren DB-Cluster mithilfe der Datenbankauthentifizierung von AWS Identity and Access Management (IAM) authentifizieren. Mit dieser Authentifizierungsmethode benötigen Sie kein Passwort, um eine Verbindung mit einem DB-Cluster herzustellen. Stattdessen verwenden Sie ein Authentifizierungstoken. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung.

Themen

- IAM-Datenbank-Authentifizierung mit Aurora MySQL
- IAM-Datenbankauthentifizierung mit Aurora PostgreSQL

IAM-Datenbank-Authentifizierung mit Aurora MySQL

Die IAM-Datenbankauthentifizierung mit Aurora MySQL ist in allen Regionen für die folgenden Versionen verfügbar:

- Aurora MySQL 3 – alle verfügbaren Versionen
- Aurora MySQL 2 – alle verfügbaren Versionen

IAM-Datenbankauthentifizierung mit Aurora PostgreSQL

Die IAM-Datenbankauthentifizierung mit Aurora PostgreSQL ist in allen Regionen für die folgenden Versionen verfügbar:

- Aurora PostgreSQL 17 – alle verfügbaren Versionen
- Aurora PostgreSQL 16 – alle verfügbaren Versionen
- Aurora PostgreSQL 15 – alle verfügbaren Versionen
- Aurora PostgreSQL 14 – alle verfügbaren Versionen
- Aurora PostgreSQL 13 – alle verfügbaren Versionen
- Aurora PostgreSQL 12 – alle verfügbaren Versionen
- Aurora PostgreSQL 11 – alle verfügbaren Versionen

Unterstützte Regionen und Aurora-DB-Engines für die Kerberos-Authentifizierung

Durch die Verwendung der Kerberos-Authentifizierung mit Aurora können Sie die externe Authentifizierung von Datenbankbenutzern mit Kerberos und Microsoft Active Directory unterstützen. Die Verwendung von Kerberos und Active Directory bietet die Vorteile des Single Sign-Ons und der zentralisierten Authentifizierung von Datenbankbenutzern. Kerberos und Active Directory sind mit AWS Directory Service for Microsoft Active Directory, einer Funktion von verfügbar. Directory Service Weitere Informationen finden Sie unter Kerberos-Authentifizierung.

Themen

- Kerberos-Authentifizierung mit Aurora MySQL
- Kerberos-Authentifizierung mit Aurora PostgreSQL
- Active Directory (AD)-Sicherheitsgruppen mit Aurora PostgreSQL

Kerberos-Authentifizierung mit Aurora MySQL

Die folgenden Regionen und Engine-Versionen sind für die Kerberos-Authentifizierung mit Aurora MySQL verfügbar.

Region	Aurora-MySQL-Version 3
USA Ost (Nord-Virginia)	Version 3.03.0 und höher
USA Ost (Ohio)	Version 3.03.0 und höher
USA West (Nordkalifornien)	Version 3.03.0 und höher
USA West (Oregon)	Version 3.03.0 und höher
Afrika (Kapstadt)	Nicht verfügbar
Asien-Pazifik (Hongkong)	Nicht verfügbar
Asien-Pazifik (Jakarta)	Nicht verfügbar
Asien-Pazifik (Malaysia)	Nicht verfügbar

Region	Aurora-MySQL-Version 3
Asien-Pazifik (Melbourne)	Nicht verfügbar
Asien-Pazifik (Mumbai)	Version 3.03.0 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar
Asien-Pazifik (Osaka)	Nicht verfügbar
Asien-Pazifik (Seoul)	Version 3.03.0 und höher
Asien-Pazifik (Singapur)	Version 3.03.0 und höher
Asien-Pazifik (Sydney)	Version 3.03.0 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 3.03.0 und höher
Kanada (Zentral)	Version 3.03.0 und höher
Kanada West (Calgary)	Nicht verfügbar
China (Peking)	Version 3.03.0 und höher
China (Ningxia)	Version 3.03.0 und höher
Europa (Frankfurt)	Version 3.03.0 und höher
Europa (Irland)	Version 3.03.0 und höher
Europa (London)	Version 3.03.0 und höher
Europa (Milan)	Nicht verfügbar
Europa (Paris)	Version 3.03.0 und höher
Europa (Spain)	Nicht verfügbar

Region	Aurora-MySQL-Version 3
Europa (Stockholm)	Version 3.03.0 und höher
Europa (Zürich)	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar
Mexiko (Zentral)	Nicht verfügbar
Middle East (Bahrain)	Nicht verfügbar
Naher Osten (VAE)	Nicht verfügbar
Südamerika (São Paulo)	Version 3.03.0 und höher
AWS GovCloud (US-Ost)	Version 3.03.0 und höher
AWS GovCloud (US-West)	Version 3.03.0 und höher

Kerberos-Authentifizierung mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für die Kerberos-Authentifizierung mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
USA Ost (Nord-Virginia)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA Ost (Ohio)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA West	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
(Nordkalifornien)							
USA West (Oregon)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Afrika (Kapstadt)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Hongkong)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Hyderabad)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Jakarta)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Melbourne)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Mumbai)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Seoul)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Singapur)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Sydney)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Tokio)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Kanada (Zentral)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Kanada West (Calgary)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
China (Peking)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
China (Ningxia)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Frankfurt)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Irland)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (London)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Milan)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Paris)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Spain)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (Stockholm)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Zürich)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Israel (Tel Aviv)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Naher Osten (VAE)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Südamerika (São Paulo)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
AWS GovCloud (US-Ost)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
AWS GovCloud (US-West)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Active Directory (AD)-Sicherheitsgruppen mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für ActiveDirectory Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
USA Ost (Nord-Virginia)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA Ost (Ohio)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA West (Nordkalifornien)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA West (Oregon)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Afrika (Kapstadt)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Hongkong)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Hyderabad)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Jakarta)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Melbourne)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Mumbai)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Seoul)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Singapur)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Sydney)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Kanada (Zentral)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Kanada West (Calgary)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
China (Peking)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
China (Ningxia)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (Frankfurt)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Irland)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (London)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Milan)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Paris)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Spain)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Stockholm)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Zürich)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Israel (Tel Aviv)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Naher Osten (VAE)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Südamerika (São Paulo)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
AWS GovCloud (US-Ost)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
AWS GovCloud (US-West)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Unterstützte Regionen und DB-Engines für Aurora Machine Learning

Durch die Verwendung von Amazon Aurora Machine Learning können Sie Ihren Aurora-DB-Cluster je nach Ihren Anforderungen in einen der folgenden AWS Machine-Learning-Services integrieren. Sie alle unterstützen spezifische Anwendungsfälle für Machine Learning.

Amazon Bedrock ist ein vollständig verwalteter Dienst, der führende Basismodelle von KI-Unternehmen über eine API verfügbar macht und zudem Entwickler-Tools zum Erstellen und Skalieren von auf generativer KI basierenden Anwendungen bereitstellt.

Amazon Comprehend ist ein verwalteter Dienst für die natürliche Sprachverarbeitung (NLP), der verwendet wird, um Informationen aus Dokumenten zu extrahieren. Durch die Verwendung von Aurora Machine Learning mit Amazon Comprehend können Sie die Stimmung von Text in Ihren Datenbanktabellen bestimmen.

SageMaker KI ist ein Service für maschinelles Lernen mit vollem Funktionsumfang. Datenwissenschaftler verwenden Amazon SageMaker AI, um Modelle für maschinelles Lernen

für eine Vielzahl von Inferenzaufgaben zu erstellen, zu trainieren und zu testen, z. B. zur Betrugserkennung. Durch die Verwendung von Aurora Machine Learning mit SageMaker KI können Datenbankentwickler die SageMaker KI-Funktionalität im SQL-Code aufrufen.

Nicht alle AWS-Regionen unterstützen alle Dienste für maschinelles Lernen. Nur bestimmte AWS-Regionen unterstützen Aurora Machine Learning und bieten daher Zugriff auf diese Dienste von einem Aurora-DB-Cluster aus. Der Integrationsprozess für Aurora Machine Learning ist außerdem je nach Datenbank-Engine unterschiedlich. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Machine Learning.

Themen

- Aurora Machine Learning mit Aurora MySQL
- Aurora Machine Learning mit Aurora PostgreSQL

Aurora Machine Learning mit Aurora MySQL

Amazon Bedrock wird nur auf Aurora MySQL Version 3.06 und höher unterstützt. Informationen zur Verfügbarkeit von Regionen für Amazon Bedrock finden Sie unter Modell-Support nach AWS-Region im Amazon-Bedrock-Benutzerhandbuch.

Aurora Machine Learning mit Amazon Comprehend und Amazon SageMaker AI wird für Aurora MySQL unterstützt, wie in der AWS-Regionen Tabelle aufgeführt. Sie müssen nicht nur Ihre Version von Aurora MySQL verfügbar haben, sondern auch den Dienst unterstützen, den Sie verwenden möchten. AWS-Region Eine Liste, AWS-Regionen wo Amazon SageMaker AI verfügbar ist, finden Sie unter Amazon SageMaker AI-Endpunkte und Kontingente in der Allgemeine Amazon Web Services-Referenz. Eine Liste, AWS-Regionen wo Amazon Comprehend verfügbar ist, finden Sie unter Amazon Comprehend Endpoints and Quotas in der. Allgemeine Amazon Web Services-Referenz

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Version 3.01.0 und höher	Version 2.07 und höher
USA Ost (Ohio)	Version 3.01.0 und höher	Version 2.07 und höher
USA West (Nordkalifornien)	Version 3.01.0 und höher	Version 2.07 und höher
USA West (Oregon)	Version 3.01.0 und höher	Version 2.07 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Afrika (Kapstadt)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Hongkong)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Hyderabad)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Jakarta)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Malaysia)	Version 3.04.0 und höher	Nicht verfügbar
Asien-Pazifik (Melbourne)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Mumbai)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 3.01.0 und höher	Version 2.07.3 und höher
Asien-Pazifik (Seoul)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Singapur)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Sydney)	Version 3.01.0 und höher	Version 2.07 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 3.01.0 und höher	Version 2.07 und höher
Kanada (Zentral)	Version 3.01.0 und höher	Version 2.07 und höher
Kanada West (Calgary)	Version 3.01.0 und höher	Version 2.07 und höher
China (Peking)	Version 3.01.0 und höher	Version 2.07 und höher
China (Ningxia)	Version 3.01.0 und höher	Version 2.07 und höher
Europa (Frankfurt)	Version 3.01.0 und höher	Version 2.07 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Europa (Irland)	Version 3.01.0 und höher	Version 2.07 und höher
Europa (London)	Version 3.01.0 und höher	Version 2.07 und höher
Europa (Milan)	Nicht verfügbar	Nicht verfügbar
Europa (Paris)	Version 3.01.0 und höher	Version 2.07 und höher
Europa (Spain)	Version 3.01.0 und höher	Version 2.07 und höher
Europa (Stockholm)	Version 3.01.0 und höher	Version 2.07 und höher
Europa (Zürich)	Version 3.01.0 und höher	Version 2.07 und höher
Israel (Tel Aviv)	Version 3.01.0 und höher	Version 2.07 und höher
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Version 3.01.0 und höher	Version 2.07 und höher
Naher Osten (VAE)	Version 3.01.0 und höher	Version 2.07 und höher
Südamerika (São Paulo)	Version 3.01.0 und höher	Version 2.07 und höher
AWS GovCloud (US-Ost)	Version 3.01.0 und höher	Version 2.07 und höher
AWS GovCloud (US-West)	Version 3.01.0 und höher	Version 2.07 und höher

Aurora Machine Learning mit Aurora PostgreSQL

Informationen zur Versionsunterstützung für Amazon Bedrock auf Aurora PostgreSQL finden Sie unter Verwenden von Aurora PostgreSQL als Wissensdatenbank für Amazon Bedrock.

Aurora Machine Learning mit Amazon Comprehend und Amazon SageMaker AI wird für Aurora PostgreSQL unterstützt, wie in der Tabelle AWS-Regionen aufgeführt. Sie müssen nicht nur Ihre Version von Aurora PostgreSQL verfügbar haben, sondern auch den AWS-Region Dienst unterstützen, den Sie verwenden möchten. Eine Liste, AWS-Regionen wo Amazon SageMaker AI verfügbar ist, finden Sie unter Amazon SageMaker AI-Endpunkte und Kontingente in der Allgemeine

Amazon Web Services-Referenz. Eine Liste, AWS-Regionen wo Amazon Comprehend verfügbar ist, finden Sie unter Amazon Comprehend Endpoints and Quotas in der. Allgemeine Amazon Web Services-Referenz

Die folgenden Regionen und Engine-Versionen sind für Aurora Machine Learning mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
USA Ost (Nord-Virginia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
USA Ost (Ohio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
USA West (Nordkalifornien)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
USA West (Oregon)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Afrika (Kapstadt)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Hongkong)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Hyderabad)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Jakarta)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Malaysia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Melbourne)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Mumbai)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Neuseeland)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Osaka)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Seoul)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Singapur)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Sydney)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Taipeh)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Thailand)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Asien-Pazifik (Tokio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Kanada (Zentral)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Kanada West (Calgary)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
China (Peking)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
China (Ningxia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Europa (Frankfurt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Europa (Irland)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (London)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Europa (Milan)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Europa (Paris)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Europa (Spain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Europa (Stockholm)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Europa (Zürich)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Israel (Tel Aviv)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Naher Osten (VAE)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
Südamerika (São Paulo)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher
AWS GovCloud (US-Ost)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud (US-West)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.3 und höher	Version 12.4 und höher	Version 11.9, 11.12 und höher

Unterstützte Regionen und Aurora-DB-Engines für Performance Insights

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.

Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.

Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.

Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-

Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

Performance-Insights erweitert die vorhandenen Amazon-RDS-Überwachungsfunktionen, um die Leistung Ihrer Datenbank zu veranschaulichen und zu analysieren. Mit dem Performance-Insights-Dashboard können Sie die Datenbanklast Ihrer DB-Instance von Amazon RDS visualisieren und die Last nach Wartezeiten, SQL-Anweisungen, Hosts oder Benutzern filtern. Weitere Informationen finden Sie unter Übersicht über Performance-Insights für Amazon Aurora.

Informationen zur Unterstützung von Performance-Insights-Funktionen nach Region, DB-Engine und Instance-Klasse finden Sie unter DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance-Insights-Funktionen.

Themen

- Performance-Insights mit Aurora MySQL
- Performance-Insights mit Aurora PostgreSQL
- Performance-Insights mit Aurora Serverless

Performance-Insights mit Aurora MySQL

> **Note**
>
> Die Unterstützung der Engine-Version ist für Performance-Insights mit Aurora MySQL anders, wenn Sie parallele Abfragen aktiviert haben. Weitere Informationen zu parallelen Abfragen finden Sie unter Parallel Query für Amazon Aurora MySQL.

Themen

- Performance-Insights mit Aurora MySQL und deaktivierter paralleler Abfragefunktion
- Performance-Insights mit Aurora MySQL und aktivierter paralleler Abfragefunktion

Performance-Insights mit Aurora MySQL und deaktivierter paralleler Abfragefunktion

Die folgenden Regionen und Engine-Versionen sind für Performance Insights mit Aurora MySQL und deaktivierter Parallelabfragefunktion verfügbar.

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Alle Versionen	Alle Versionen
USA Ost (Ohio)	Alle Versionen	Alle Versionen
USA West (Nordkalifornien)	Alle Versionen	Alle Versionen
USA West (Oregon)	Alle Versionen	Alle Versionen
Afrika (Kapstadt)	Alle Versionen	Alle Versionen
Asien-Pazifik (Hongkong)	Alle Versionen	Alle Versionen
Asien-Pazifik (Hyderabad)	Alle Versionen	Alle Versionen
Asien-Pazifik (Jakarta)	Alle Versionen	Alle Versionen
Asien-Pazifik (Malaysia)	Alle Versionen	Alle Versionen
Asien-Pazifik (Melbourne)	Alle Versionen	Alle Versionen
Asien-Pazifik (Mumbai)	Alle Versionen	Alle Versionen
Asien-Pazifik (Neuseeland)	Alle Versionen	Alle Versionen
Asien-Pazifik (Osaka)	Alle Versionen	Alle Versionen
Asien-Pazifik (Seoul)	Alle Versionen	Alle Versionen
Asien-Pazifik (Singapur)	Alle Versionen	Alle Versionen
Asien-Pazifik (Sydney)	Alle Versionen	Alle Versionen
Asien-Pazifik (Taipeh)	Alle Versionen	Alle Versionen
Asien-Pazifik (Thailand)	Alle Versionen	Alle Versionen
Asien-Pazifik (Tokio)	Alle Versionen	Alle Versionen
Kanada (Zentral)	Alle Versionen	Alle Versionen

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Kanada West (Calgary)	Alle Versionen	Alle Versionen
China (Peking)	Alle Versionen	Alle Versionen
China (Ningxia)	Alle Versionen	Alle Versionen
Europa (Frankfurt)	Alle Versionen	Alle Versionen
Europa (Irland)	Alle Versionen	Alle Versionen
Europa (London)	Alle Versionen	Alle Versionen
Europa (Milan)	Alle Versionen	Alle Versionen
Europa (Paris)	Alle Versionen	Alle Versionen
Europa (Spain)	Alle Versionen	Alle Versionen
Europa (Stockholm)	Alle Versionen	Alle Versionen
Europa (Zürich)	Alle Versionen	Alle Versionen
Israel (Tel Aviv)	Alle Versionen	Alle Versionen
Mexiko (Zentral)	Alle Versionen	Alle Versionen
Middle East (Bahrain)	Alle Versionen	Alle Versionen
Naher Osten (VAE)	Alle Versionen	Alle Versionen
Südamerika (São Paulo)	Alle Versionen	Alle Versionen
AWS GovCloud (US-Ost)	Alle Versionen	Alle Versionen
AWS GovCloud (US-West)	Alle Versionen	Alle Versionen

Performance-Insights mit Aurora MySQL und aktivierter paralleler Abfragefunktion

Die folgenden Regionen und Engine-Versionen sind für Performance Insights mit Aurora MySQL und aktivierter Parallelabfragefunktion verfügbar.

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Nicht verfügbar	Version 2.09.0 und höher
USA Ost (Ohio)	Nicht verfügbar	Version 2.09.0 und höher
USA West (Nordkalifornien)	Nicht verfügbar	Version 2.09.0 und höher
USA West (Oregon)	Nicht verfügbar	Version 2.09.0 und höher
Afrika (Kapstadt)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Hongkong)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Hyderabad)	Nicht verfügbar	Alle Versionen
Asien-Pazifik (Jakarta)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Malaysia)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Melbourne)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Mumbai)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Osaka)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Seoul)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Singapur)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Sydney)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Version 2.09.0 und höher
Asien-Pazifik (Thailand)	Nicht verfügbar	Version 2.09.0 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Asien-Pazifik (Tokio)	Nicht verfügbar	Version 2.09.0 und höher
Kanada (Zentral)	Nicht verfügbar	Version 2.09.0 und höher
Kanada West (Calgary)	Nicht verfügbar	Version 2.09.0 und höher
China (Peking)	Nicht verfügbar	Version 2.09.0 und höher
China (Ningxia)	Nicht verfügbar	Version 2.09.0 und höher
Europa (Frankfurt)	Nicht verfügbar	Version 2.09.0 und höher
Europa (Irland)	Nicht verfügbar	Version 2.09.0 und höher
Europa (London)	Nicht verfügbar	Version 2.09.0 und höher
Europa (Milan)	Nicht verfügbar	Version 2.09.0 und höher
Europa (Paris)	Nicht verfügbar	Version 2.09.0 und höher
Europa (Spain)	Nicht verfügbar	Version 2.09.0 und höher
Europa (Stockholm)	Nicht verfügbar	Version 2.09.0 und höher
Europa (Zürich)	Nicht verfügbar	Version 2.09.0 und höher
Israel (Tel Aviv)	Nicht verfügbar	Version 2.09.0 und höher
Mexiko (Zentral)	Nicht verfügbar	Version 2.09.0 und höher
Middle East (Bahrain)	Nicht verfügbar	Version 2.09.0 und höher
Naher Osten (VAE)	Nicht verfügbar	Version 2.09.0 und höher
Südamerika (São Paulo)	Nicht verfügbar	Version 2.09.0 und höher
AWS GovCloud (US-Ost)	Nicht verfügbar	Version 2.09.0 und höher
AWS GovCloud (US-West)	Nicht verfügbar	Version 2.09.0 und höher

Performance-Insights mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für Performance Insights mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
USA Ost (Nord-Virginia)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA Ost (Ohio)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA West (Nordkalifornien)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
USA West (Oregon)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Afrika (Kapstadt)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Hongkong)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
(Hyderabad)								
Asien-Pazifik (Jakarta)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Malaysia)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Melbourne)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Mumbai)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Neuseeland)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Osaka)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Seoul)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
Asien-Pazifik (Singapur)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Sydney)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Taipeh)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Thailand)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Asien-Pazifik (Tokio)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Kanada (Zentral)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Kanada West (Calgary)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
China (Peking)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
China (Ningxia)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
Europa (Frankfurt)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Irland)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (London)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Milan)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Paris)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Spain)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Stockholm)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Europa (Zürich)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Israel (Tel Aviv)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Mexiko (Zentral)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
Middle East (Bahrain)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Naher Osten (VAE)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
Südamerika (São Paulo)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
AWS GovCloud (US-Ost)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen
AWS GovCloud (US-West)	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen	Alle Versionen

Performance-Insights mit Aurora Serverless

Aurora Serverless v2 unterstützt Performance Insights für alle MySQL-kompatiblen und PostgreSQL-kompatiblen Versionen. Wir empfehlen, die Mindestkapazität auf mindestens 2 Aurora-Kapazitätseinheiten (ACUs) festzulegen.

Aurora Serverless v1 unterstützt Performance Insights nicht.

Unterstützte Regionen und Aurora-DB-Engines für Null-ETL-Integrationen

Bei Null-ETL-Integrationen von Amazon Aurora handelt es sich um eine vollständig verwaltete Lösung, die Transaktionsdaten in Amazon Redshift oder Amazon SageMaker verfügbar macht,

nachdem diese in einen Aurora-Cluster geschrieben wurden. Weitere Informationen finden Sie unter Zero-ETL-Integrationen.

Themen

- Null-ETL-Integrationen in Aurora MySQL
- Null-ETL-Integrationen in Aurora PostgreSQL

Null-ETL-Integrationen in Aurora MySQL

Die folgenden Regionen und Engine-Versionen sind für Null-ETL-Integrationen von Aurora MySQL in Amazon Redshift und Amazon SageMaker verfügbar.

Region	Null-ETL-Integration in Amazon Redshift für Aurora MySQL Version 3	Null-ETL-Integration in Amazon SageMaker für Aurora MySQL Version 3
USA Ost (Nord-Virginia)	Version 3.05.2 und höher	Version 3.05.2 und höher
USA Ost (Ohio)	Version 3.05.2 und höher	Version 3.05.2 und höher
USA West (Nordkalifornien)	Version 3.05.2 und höher	Nicht verfügbar
USA West (Oregon)	Version 3.05.2 und höher	Version 3.05.2 und höher
Afrika (Kapstadt)	Version 3.05.2 und höher	Nicht verfügbar
Asien-Pazifik (Hongkong)	Version 3.05.2 und höher	Version 3.05.2 und höher
Asien-Pazifik (Hyderabad)	Version 3.05.2 und höher	Nicht verfügbar
Asien-Pazifik (Jakarta)	Version 3.05.2 und höher	Nicht verfügbar
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Melbourne)	Version 3.05.2 und höher	Nicht verfügbar
Asien-Pazifik (Mumbai)	Version 3.05.2 und höher	Nicht verfügbar
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar

Region	Null-ETL-Integration in Amazon Redshift für Aurora MySQL Version 3	Null-ETL-Integration in Amazon SageMaker für Aurora MySQL Version 3
Asien-Pazifik (Osaka)	Version 3.05.2 und höher	Nicht verfügbar
Asien-Pazifik (Seoul)	Version 3.05.2 und höher	Version 3.05.2 und höher
Asien-Pazifik (Singapur)	Version 3.05.2 und höher	Version 3.05.2 und höher
Asien-Pazifik (Sydney)	Version 3.05.2 und höher	Version 3.05.2 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 3.05.2 und höher	Version 3.05.2 und höher
Kanada (Zentral)	Version 3.05.2 und höher	Version 3.05.2 und höher
Kanada West (Calgary)	Version 3.05.2 und höher	Nicht verfügbar
China (Peking)	Version 3.05.2 und höher	Nicht verfügbar
China (Ningxia)	Version 3.05.2 und höher	Nicht verfügbar
Europa (Frankfurt)	Version 3.05.2 und höher	Version 3.05.2 und höher
Europa (Irland)	Version 3.05.2 und höher	Version 3.05.2 und höher
Europa (London)	Version 3.05.2 und höher	Version 3.05.2 und höher
Europa (Milan)	Version 3.05.2 und höher	Nicht verfügbar
Europa (Paris)	Version 3.05.2 und höher	Nicht verfügbar
Europa (Spain)	Version 3.05.2 und höher	Nicht verfügbar
Europa (Stockholm)	Version 3.05.2 und höher	Version 3.05.2 und höher
Europa (Zürich)	Version 3.05.2 und höher	Nicht verfügbar

Region	Null-ETL-Integration in Amazon Redshift für Aurora MySQL Version 3	Null-ETL-Integration in Amazon SageMaker für Aurora MySQL Version 3
Israel (Tel Aviv)	Version 3.05.2 und höher	Nicht verfügbar
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Version 3.05.2 und höher	Nicht verfügbar
Naher Osten (VAE)	Version 3.05.2 und höher	Nicht verfügbar
Südamerika (São Paulo)	Version 3.05.2 und höher	Version 3.05.2 und höher
AWS GovCloud (USA-Ost)	Nicht verfügbar	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar	Nicht verfügbar

Null-ETL-Integrationen in Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für Aurora PostgreSQL Zero-ETL-Integrationen mit Amazon Redshift und verfügbar. Amazon Sagemaker

Region	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 16	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 16
USA Ost (Ohio)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
USA Ost (Nord-Virginia)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher

Region	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 16	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 16
USA West (Nordkalifornien)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
USA West (Oregon)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Afrika (Kapstadt)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Asien-Pazifik (Hongkong)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Asien-Pazifik (Hyderabad)	Version 17.4 und höher	Version 16.4 und höher	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Jakarta)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Asien-Pazifik (Melbourne)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar		
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar		
Asien-Pazifik (Mumbai)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar

Region	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 16	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 16
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar		
Asien-Pazifik (Osaka)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Asien-Pazifik (Seoul)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Asien-Pazifik (Singapur)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Asien-Pazifik (Sydney)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar		
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar		
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar		
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar		
Asien-Pazifik (Tokio)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher

Region	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 16	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 16
Kanada (Zentral)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Kanada West (Calgary)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
China (Peking)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
China (Ningxia)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Europa (Frankfurt)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Europa (Irland)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Europa (London)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Europa (Milan)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Europa (Paris)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Europa (Spain)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar

Region	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 17	Zero-ETL-Integration mit Amazon Redshift für Aurora PostgreSQL Version 16	Zero-ETL-Integration mit Amazon SageMaker für Aurora PostgreSQL Version 16
Europa (Stockholm)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
Europa (Zürich)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Israel (Tel Aviv)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar		
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar		
Middle East (Bahrain)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Naher Osten (VAE)	Version 17.4 und höher	Nicht verfügbar	Version 16.4 und höher	Nicht verfügbar
Südamerika (São Paulo)	Version 17.4 und höher	Version 17.4 und höher	Version 16.4 und höher	Version 16.4 und höher
AWS GovCloud (US-Ost)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Unterstützte Regionen und Aurora-DB-Engines für Amazon-RDS-Proxy

Amazon-RDS-Proxy ist ein vollständig verwalteter, hochverfügbarer Datenbank-Proxy, der Anwendungen durch Bündelung und gemeinsame Nutzung etablierter Datenbankverbindungen skalierbarer macht. Weitere Informationen zu RDS-Proxy finden Sie unter Amazon-RDS-Proxy für Aurora.

Themen

- Amazon-RDS-Proxy mit Aurora MySQL
- Amazon-RDS-Proxy mit Aurora PostgreSQL

Amazon-RDS-Proxy mit Aurora MySQL

Die folgenden Regionen und Engine-Versionen sind für RDS-Proxy mit Aurora MySQL verfügbar.

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
USA Ost (Ohio)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
USA West (Nordkalifornien)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
USA West (Oregon)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Afrika (Kapstadt)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Hongkong)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Hyderabad)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Asien-Pazifik (Jakarta)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Malaysia)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Melbourne)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Mumbai)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Seoul)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Singapur)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Sydney)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Asien-Pazifik (Tokio)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Kanada (Zentral)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Kanada West (Calgary)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
China (Peking)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
China (Ningxia)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (Frankfurt)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (Irland)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (London)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (Milan)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (Paris)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (Spain)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (Stockholm)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Europa (Zürich)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Israel (Tel Aviv)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Mexiko (Zentral)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Middle East (Bahrain)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Naher Osten (VAE)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
Südamerika (São Paulo)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
AWS GovCloud (USA-Ost)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher
AWS GovCloud (US-West)	Version 3.01.0 und höher	Version 2.07 sowie Version 2.11 und höher

Amazon-RDS-Proxy mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für RDS-Proxy mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
USA Ost (Nord-Virginia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
USA Ost (Ohio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
							11.13 und höher
USA West (Nordkalifornien)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
USA West (Oregon)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Afrika (Kapstadt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Hongkong)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Hyderabad)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Jakarta)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Malaysia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Melbourne)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Mumbai)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Seoul)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Singapur)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Sydney)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asien-Pazifik (Thailand)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Asien-Pazifik (Tokio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Kanada (Zentral)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Kanada West (Calgary)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
China (Peking)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
China (Ningxia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Frankfurt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Irland)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (London)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Milan)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Paris)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Spain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Europa (Stockholm)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Europa (Zürich)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Israel (Tel Aviv)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Mexiko (Zentral)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Middle East (Bahrain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Naher Osten (VAE)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
Südamerika (São Paulo)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher
AWS GovCloud (US-Ost)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud (US-West)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.4 und höher	Version 12.8 und höher	Version 11.9 sowie Version 11.13 und höher

Unterstützte Regionen und Aurora-DB-Engines für die Integration von Secrets Manager

Mit AWS Secrets Manager können Sie hartkodierte Anmeldeinformationen im Code (einschließlich Datenbankpasswörter) durch einen API-Aufruf an Secrets Manager ersetzen und das Secret programmgesteuert abrufen. Weitere Informationen zu Secrets Manager finden Sie im Benutzerhandbuch für AWS Secrets Manager.

Sie können angeben, dass Amazon Aurora das Hauptbenutzerpasswort in Secrets Manager für einen Aurora-DB-Cluster verwaltet. Aurora generiert das Passwort, speichert es in Secrets Manager und wechselt es regelmäßig. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager.

Die Integration von Secrets Manager ist in allen AWS-Regionen verfügbar.

Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2

Aurora Serverless v2 ist eine On-Demand-Funktion für die automatische Skalierung, die als kostengünstiger Ansatz für die Ausführung intermittierender oder unvorhersehbarer Workloads auf Amazon Aurora entwickelt wurde. Die Kapazität wird je nach Bedarf Ihrer Anwendungen hoch- oder herunterskaliert. Die Skalierung ist schneller und stärker granular als mit Aurora Serverless v1. Mit Aurora Serverless v2 kann jeder Cluster eine Writer-DB-Instance und mehrere Reader-DB-Instances enthalten. Sie können Aurora Serverless v2 und traditionelle bereitgestellte DB-Instances innerhalb desselben Clusters kombinieren. Weitere Informationen finden Sie unter Verwenden von Aurora Serverless v2.

Themen

- Aurora Serverless v2 mit Aurora MySQL
- Aurora Serverless v2 mit Aurora PostgreSQL

Aurora Serverless v2 mit Aurora MySQL

Die folgenden Regionen und Engine-Versionen sind für Aurora Serverless v2 mit Aurora MySQL verfügbar.

Region	Aurora-MySQL-Version 3
USA Ost (Nord-Virginia)	Version 3.02.0 und höher
USA Ost (Ohio)	Version 3.02.0 und höher
USA West (Nordkalifornien)	Version 3.02.0 und höher
USA West (Oregon)	Version 3.02.0 und höher
Afrika (Kapstadt)	Version 3.02.0 und höher
Asien-Pazifik (Hongkong)	Version 3.02.0 und höher
Asien-Pazifik (Hyderabad)	Version 3.02.3 und höher
Asien-Pazifik (Jakarta)	Version 3.02.0 und höher
Asien-Pazifik (Malaysia)	Versionen 3.04.3, 3.05.2, 3.06.1, 3.07.1 und höher
Asien-Pazifik (Melbourne)	Version 3.02.3 und höher
Asien-Pazifik (Mumbai)	Version 3.02.0 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 3.02.0 und höher
Asien-Pazifik (Seoul)	Version 3.02.0 und höher

Region	Aurora-MySQL-Version 3
Asien-Pazifik (Singapur)	Version 3.02.0 und höher
Asien-Pazifik (Sydney)	Version 3.02.0 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar
Asien-Pazifik (Thailand)	Versionen 3.04.3 und höher, 3.08.0 und höher
Asien-Pazifik (Tokio)	Version 3.02.0 und höher
Kanada (Zentral)	Version 3.02.0 und höher
Kanada West (Calgary)	Version 3.04.0 und höher
China (Peking)	Version 3.02.2 und höher
China (Ningxia)	Version 3.02.2 und höher
Europa (Frankfurt)	Version 3.02.0 und höher
Europa (Irland)	Version 3.02.0 und höher
Europa (London)	Version 3.02.0 und höher
Europa (Milan)	Version 3.02.0 und höher
Europa (Paris)	Version 3.02.0 und höher
Europa (Spain)	Version 3.02.3 und höher
Europa (Stockholm)	Version 3.02.0 und höher
Europa (Zürich)	Version 3.02.3 und höher
Israel (Tel Aviv)	Versionen 3.02.3 und höher, 3.03.1 und höher
Mexiko (Zentral)	Nicht verfügbar
Middle East (Bahrain)	Version 3.02.0 und höher

Region	Aurora-MySQL-Version 3
Naher Osten (VAE)	Version 3.02.3 und höher
Südamerika (São Paulo)	Version 3.02.0 und höher
AWS GovCloud (US-Ost)	Version 3.02.2 und höher
AWS GovCloud (US-West)	Version 3.02.2 und höher

Die oberen und unteren ACU-Kapazitätsgrenzen für Aurora Serverless v2 können je nach Engine-Version variieren. Details hierzu finden Sie unter Kapazität von Aurora Serverless v2.

Aurora Serverless v2 mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für Aurora Serverless v2 mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
USA Ost (Nord-Virginia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
USA Ost (Ohio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
USA West (Nordkalifornien)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
USA West (Oregon)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Afrika (Kapstadt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Asien-Pazifik (Hongkong)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Asien-Pazifik (Hyderabad)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.6 und höher	Version 13.9 und höher
Asien-Pazifik (Jakarta)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Asien-Pazifik (Malaysia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.4 und höher	Version 14.6, 14.9 und höher	Version 13.9, 13.12 und höher
Asien-Pazifik (Melbourne)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.6 und höher	Version 13.9 und höher
Asien-Pazifik (Mumbai)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Asien-Pazifik (Seoul)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Asien-Pazifik (Singapur)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Asien-Pazifik (Sydney)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Asien-Pazifik (Thailand)	Version 17.4 und höher	Version 16.4 und höher	Version 15.8 und höher	Version 14.13 und höher	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Kanada (Zentral)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Kanada West (Calgary)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.6, 14.8 und höher	Version 13.9, 13.11 und höher
China (Peking)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
China (Ningxia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Europa (Frankfurt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Europa (Irland)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Europa (London)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Europa (Milan)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Europa (Paris)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Europa (Spain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.6 und höher	Version 13.9 und höher
Europa (Stockholm)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Europa (Zürich)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.6 und höher	Version 13.9 und höher
Israel (Tel Aviv)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.6 und höher	Version 13.9 und höher
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
Naher Osten (VAE)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.6 und höher	Version 13.9 und höher
Südamerika (São Paulo)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
AWS GovCloud (US-Ost)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher
AWS GovCloud (US-West)	Version 17.4 und höher	Version 16.1 und höher	Version 15.2 und höher	Version 14.3 und höher	Version 13.6 und höher

Die oberen und unteren ACU-Kapazitätsgrenzen für Aurora Serverless v2 können je nach Engine-Version variieren. Details hierzu finden Sie unter Kapazität von Aurora Serverless v2.

Aurora Serverless v1

> **Important**
>
> AWS hat den end-of-life Termin bekannt gegeben fürAurora Serverless v1: 31. März 2025. Wir empfehlen dringend, alle DB-Cluster mit Aurora Serverless v1 vor diesem Datum auf Aurora Serverless v2 zu aktualisieren. Das Upgrade kann eine Änderung der Hauptversionsnummer der Datenbank-Engine beinhalten. Daher ist es wichtig, diese Umstellung vor dem Datum zu planen, zu testen und zu implementieren. end-of-life Ab dem 8. Januar 2025 können Kunden keine neuen Aurora Serverless v1 Cluster oder Instances mehr mit der AWS-Managementkonsole oder der CLI erstellen. Informationen über den Migrationsprozess finden Sie unter Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2.
>
> Aurora Serverless v2 skaliert schneller und stärker granular. Aurora Serverless v2 bietet außerdem mehr Kompatibilität mit anderen Aurora-Features wie Reader-DB-Instances. Weitere Informationen über Aurora Serverless v2 finden Sie unter Verwenden von Aurora Serverless v2.

Aurora Serverless v1 ist eine On-Demand-Funktion für die automatische Skalierung, die als kostengünstiger Ansatz für die Ausführung intermittierender oder unvorhersehbarer Workloads auf Amazon Aurora entwickelt wurde. Der Service kann automatisch starten, herunterfahren und die Kapazität je nach Bedarf Ihrer Anwendungen hoch- oder herunterskalieren. Dabei wird eine einzelne DB-Instance in jedem Cluster verwendet. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Serverless v1.

Themen

- Aurora Serverless v1 mit Aurora MySQL
- Aurora Serverless v1 mit Aurora PostgreSQL

Aurora Serverless v1 mit Aurora MySQL

Die folgenden Regionen und Engine-Versionen sind für Aurora Serverless v1 mit Aurora MySQL verfügbar.

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Nicht verfügbar	Version: 2.11.4
USA Ost (Ohio)	Nicht verfügbar	Version 2.11.4
USA West (Nordkalifornien)	Nicht verfügbar	Version 2.11.4
USA West (Oregon)	Nicht verfügbar	Version 2.11.4
Afrika (Kapstadt)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Hongkong)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Hyderabad)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Jakarta)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Melbourne)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Mumbai)	Nicht verfügbar	Version 2.11.4
Asien-Pazifik (Osaka)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Seoul)	Nicht verfügbar	Version 2.11.4
Asien-Pazifik (Singapur)	Nicht verfügbar	Version 2.11.4
Asien-Pazifik (Sydney)	Nicht verfügbar	Version 2.11.4
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Nicht verfügbar	Version 2.11.4
Kanada (Zentral)	Nicht verfügbar	Version 2.11.4
Kanada West (Calgary)	Nicht verfügbar	Nicht verfügbar
China (Peking)	Nicht verfügbar	Nicht verfügbar

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
China (Ningxia)	Nicht verfügbar	Version 2.11.4
Europa (Frankfurt)	Nicht verfügbar	Version 2.11.4
Europa (Irland)	Nicht verfügbar	Version 2.11.4
Europa (London)	Nicht verfügbar	Version 2.11.4
Europa (Milan)	Nicht verfügbar	Nicht verfügbar
Europa (Paris)	Nicht verfügbar	Version 2.11.4
Europa (Spain)	Nicht verfügbar	Nicht verfügbar
Europa (Stockholm)	Nicht verfügbar	Nicht verfügbar
Europa (Zürich)	Nicht verfügbar	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Nicht verfügbar	Nicht verfügbar
Naher Osten (VAE)	Nicht verfügbar	Nicht verfügbar
Südamerika (São Paulo)	Nicht verfügbar	Nicht verfügbar
AWS GovCloud (US-Ost)	Nicht verfügbar	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar	Nicht verfügbar

Aurora Serverless v1 mit Aurora PostgreSQL

Die folgenden Regionen und Engine-Versionen sind für Aurora Serverless v1 mit Aurora PostgreSQL verfügbar.

Region	Aurora PostgreSQL 13
USA Ost (Nord-Virginia)	Version 13.12

Region	Aurora PostgreSQL 13
USA Ost (Ohio)	Version 13.12
USA West (Nordkalifornien)	Version 13.12
USA West (Oregon)	Version 13.12
Afrika (Kapstadt)	Nicht verfügbar
Asien-Pazifik (Hongkong)	Nicht verfügbar
Asien-Pazifik (Hyderabad)	Nicht verfügbar
Asien-Pazifik (Jakarta)	Nicht verfügbar
Asien-Pazifik (Malaysia)	Nicht verfügbar
Asien-Pazifik (Melbourne)	Nicht verfügbar
Asien-Pazifik (Mumbai)	Version 13.12
Asien-Pazifik (Osaka)	Nicht verfügbar
Asien-Pazifik (Seoul)	Version 13.12
Asien-Pazifik (Singapur)	Version 13.12
Asien-Pazifik (Sydney)	Version 13.12
Asien-Pazifik (Thailand)	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 13.12
Kanada (Zentral)	Version 13.12
Kanada West (Calgary)	Nicht verfügbar
China (Peking)	Nicht verfügbar
China (Ningxia)	Nicht verfügbar

Region	Aurora PostgreSQL 13
Europa (Frankfurt)	Version 13.12
Europa (Irland)	Version 13.12
Europa (London)	Version 13.12
Europa (Milan)	Nicht verfügbar
Europa (Paris)	Version 13.12
Europa (Spain)	Nicht verfügbar
Europa (Stockholm)	Nicht verfügbar
Europa (Zürich)	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar
Middle East (Bahrain)	Nicht verfügbar
Naher Osten (VAE)	Nicht verfügbar
Südamerika (São Paulo)	Nicht verfügbar
AWS GovCloud (US-Ost)	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar

Unterstützte Regionen und Aurora-DB-Engines für RDS-Daten-API

RDS-Daten-API (Daten-API) bietet eine Web-Services-Schnittstelle zu einem DB-Cluster von Amazon Aurora. Anstatt Datenbankverbindungen von Clientanwendungen aus zu verwalten, können Sie SQL-Befehle für einen HTTPS-Endpunkt ausführen. Weitere Informationen finden Sie unter Verwenden der Daten-API von Amazon RDS.

Themen

- Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellt
- Daten-API mit Aurora MySQL Serverless v2 und bereitgestellt

- Daten-API mit Aurora PostgreSQL Serverless v1
- Daten-API mit Aurora MySQL Serverless v1

Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellt

Die folgenden Regionen und Engine-Versionen sind für Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellten DB-Clustern verfügbar.

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
USA Ost (Nord-Virginia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
USA Ost (Ohio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
USA West (Nordkalifornien)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
USA West (Oregon)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Afrika (Kapstadt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Hongkong)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Hyderabad)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Jakarta)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Asien-Pazifik (Malaysia)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Melbourne)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Mumbai)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Seoul)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Singapur)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Sydney)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Thailand)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Asien-Pazifik (Tokio)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Kanada (Zentral)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Kanada West (Calgary)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
China (Peking)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
China (Ningxia)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Europa (Frankfurt)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Europa (Irland)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Europa (London)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Europa (Milan)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Europa (Paris)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Europa (Spain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Europa (Stockholm)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Europa (Zürich)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Region	Aurora PostgreSQL 17	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Middle East (Bahrain)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Mexiko (Zentral)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
Naher Osten (VAE)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
Südamerika (São Paulo)	Version 17.4 und höher	Version 16.1 und höher	Version 15.3 und höher	Version 14.8 und höher	Version 13.11 und höher
AWS GovCloud (US-Ost)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar	Nicht verfügbar

Daten-API mit Aurora MySQL Serverless v2 und bereitgestellt

Die folgenden Regionen und Engine-Versionen sind für Daten-API mit Aurora MySQL Serverless v2 und bereitgestellten DB-Clustern verfügbar.

Region	Aurora-MySQL-Version 3
USA Ost (Ohio)	Version 3.07 und höher
USA Ost (Nord-Virginia)	Version 3.07 und höher
USA West (Nordkalifornien)	Version 3.07 und höher
USA West (Oregon)	Version 3.07 und höher

Region	Aurora-MySQL-Version 3
Afrika (Kapstadt)	Version 3.07 und höher
Asien-Pazifik (Hongkong)	Version 3.07 und höher
Asien-Pazifik (Hyderabad)	Nicht verfügbar
Asien-Pazifik (Jakarta)	Version 3.07 und höher
Asien-Pazifik (Malaysia)	Version 3.07 und höher
Asien-Pazifik (Melbourne)	Nicht verfügbar
Asien-Pazifik (Mumbai)	Version 3.07 und höher
Asien-Pazifik (Neuseeland)	Nicht verfügbar
Asien-Pazifik (Osaka)	Version 3.07 und höher
Asien-Pazifik (Seoul)	Version 3.07 und höher
Asien-Pazifik (Singapur)	Version 3.07 und höher
Asien-Pazifik (Sydney)	Version 3.07 und höher
Asien-Pazifik (Taipeh)	Nicht verfügbar
Asien-Pazifik (Thailand)	Version 3.07 und höher
Asien-Pazifik (Tokio)	Version 3.07 und höher
Kanada (Zentral)	Version 3.07 und höher
Kanada West (Calgary)	Nicht verfügbar
China (Peking)	Nicht verfügbar
China (Ningxia)	Nicht verfügbar
Europa (Frankfurt)	Version 3.07 und höher

Region	Aurora-MySQL-Version 3
Europa (Irland)	Version 3.07 und höher
Europa (London)	Version 3.07 und höher
Europa (Milan)	Version 3.07 und höher
Europa (Paris)	Version 3.07 und höher
Europa (Spain)	Version 3.07 und höher
Europa (Stockholm)	Version 3.07 und höher
Europa (Zürich)	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar
Mexiko (Zentral)	Nicht verfügbar
Middle East (Bahrain)	Version 3.07 und höher
Naher Osten (VAE)	Version 3.07 und höher
Südamerika (São Paulo)	Version 3.07 und höher
AWS GovCloud (US-Ost)	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar

Daten-API mit Aurora PostgreSQL Serverless v1

Die folgenden Regionen und Engine-Versionen sind für die Daten-API mit Aurora PostgreSQL Serverless v1 verfügbar.

Region	Aurora PostgreSQL 13	Aurora PostgreSQL 11
USA Ost (Nord-Virginia)	Version 13.9	Version 11.18
USA Ost (Ohio)	Version 13.9	Version 11.18

Region	Aurora PostgreSQL 13	Aurora PostgreSQL 11
USA West (Nordkalifornien)	Version 13.9	Version 11.18
USA West (Oregon)	Version 13.9	Version 11.18
Afrika (Kapstadt)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Hongkong)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Hyderabad)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Jakarta)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Malaysia)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Melbourne)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Mumbai)	Version 13.9	Version 11.18
Asien-Pazifik (Osaka)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Seoul)	Version 13.9	Version 11.18
Asien-Pazifik (Singapur)	Version 13.9	Version 11.18
Asien-Pazifik (Sydney)	Version 13.9	Version 11.18
Asien-Pazifik (Thailand)	Nicht verfügbar	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 13.9	Version 11.18
Kanada (Zentral)	Version 13.9	Version 11.18
China (Peking)	Nicht verfügbar	Nicht verfügbar
China (Ningxia)	Nicht verfügbar	Nicht verfügbar
Europa (Frankfurt)	Version 13.9	Version 11.18
Europa (Irland)	Version 13.9	Version 11.18

Region	Aurora PostgreSQL 13	Aurora PostgreSQL 11
Europa (London)	Version 13.9	Version 11.18
Europa (Milan)	Nicht verfügbar	Nicht verfügbar
Europa (Paris)	Version 13.9	Version 11.18
Europa (Spain)	Version 13.9	Version 11.18
Europa (Stockholm)	Nicht verfügbar	Nicht verfügbar
Europa (Zürich)	Nicht verfügbar	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar	Nicht verfügbar
Middle East (Bahrain)	Nicht verfügbar	Nicht verfügbar
Naher Osten (VAE)	Nicht verfügbar	Nicht verfügbar
Südamerika (São Paulo)	Nicht verfügbar	Nicht verfügbar
AWS GovCloud (US-Ost)	Nicht verfügbar	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar	Nicht verfügbar

Daten-API mit Aurora MySQL Serverless v1

Die folgenden Regionen und Engine-Versionen sind für die Daten-API mit Aurora MySQL Serverless v1 verfügbar.

Region	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Version 2.11.3
USA Ost (Ohio)	Version 2.11.3
USA West (Nordkalifornien)	Version 2.11.3
USA West (Oregon)	Version 2.11.3

Region	Aurora-MySQL-Version 2
Afrika (Kapstadt)	Nicht verfügbar
Asien-Pazifik (Hongkong)	Nicht verfügbar
Asien-Pazifik (Hyderabad)	Nicht verfügbar
Asien-Pazifik (Jakarta)	Nicht verfügbar
Asien-Pazifik (Malaysia)	Nicht verfügbar
Asien-Pazifik (Melbourne)	Nicht verfügbar
Asien-Pazifik (Mumbai)	Version 2.11.3
Asien-Pazifik (Osaka)	Nicht verfügbar
Asien-Pazifik (Seoul)	Version 2.11.3
Asien-Pazifik (Singapur)	Version 2.11.3
Asien-Pazifik (Sydney)	Version 2.11.3
Asien-Pazifik (Thailand)	Nicht verfügbar
Asien-Pazifik (Tokio)	Version 2.11.3
Kanada (Zentral)	Version 2.11.3
Kanada West (Calgary)	Nicht verfügbar
China (Peking)	Nicht verfügbar
China (Ningxia)	Version 2.11.3
Europa (Frankfurt)	Version 2.11.3
Europa (Irland)	Version 2.11.3
Europa (London)	Version 2.11.3

Region	Aurora-MySQL-Version 2
Europa (Milan)	Nicht verfügbar
Europa (Paris)	Version 2.11.3
Europa (Spain)	Version 2.11.3
Europa (Stockholm)	Nicht verfügbar
Europa (Zürich)	Nicht verfügbar
Israel (Tel Aviv)	Nicht verfügbar
Middle East (Bahrain)	Nicht verfügbar
Naher Osten (VAE)	Nicht verfügbar
Südamerika (São Paulo)	Nicht verfügbar
AWS GovCloud (US-Ost)	Nicht verfügbar
AWS GovCloud (US-West)	Nicht verfügbar

Unterstützte Regionen und Aurora-DB-Engines für Zero-Downtime-Patching (ZDP)

Das Durchführen von Upgrades für Aurora-DB-Cluster beinhaltet die Möglichkeit eines Ausfalls, wenn die Datenbank heruntergefahren wird und während sie aktualisiert wird. Wenn Sie das Upgrade starten, während die Datenbank ausgelastet ist, verlieren Sie standardmäßig alle Verbindungen und Transaktionen, die der DB-Cluster verarbeitet. Wenn Sie warten, bis die Datenbank im Leerlauf ist, um das Upgrade durchzuführen, müssen Sie möglicherweise lange warten.

Die Funktion Zero-Downtime-Patching (ZDP) versucht, Client-Verbindungen während eines Aurora-Upgrades auf Best-Effort-Basis aufrechtzuerhalten. Wenn das ZDP erfolgreich abgeschlossen wird, werden Anwendungssitzungen bewahrt und die Datenbank-Engine wird während der laufenden Aktualisierung neu gestartet. Der Neustart der Datenbank-Engine kann zu einem Abfall des Durchsatzes von einigen Sekunden bis 1 Minute führen.

Detaillierte Informationen zu den Bedingungen und Engine-Versionen, unter denen ZDP für Aurora-MySQL-Upgrades verfügbar ist, finden Sie unter Verwendung von Zero-Downtime-Patching (Patchen ohne Ausfallzeiten).

Detaillierte Informationen zu den Bedingungen und Engine-Versionen, unter denen ZDP für Aurora-PostgreSQL-Upgrades verfügbar ist, finden Sie unter Nebenversions-Upgrades und Zero-Downtime-Patching.

Unterstützte Regionen in Aurora PostgreSQL Limitless Database

Amazon Aurora PostgreSQL Limitless Database bietet automatisierte horizontale Skalierung zur Verarbeitung von Millionen von Schreibtransaktionen pro Sekunde, und verwaltet Petabyte an Daten. Gleichzeitig bleibt die Einfachheit des Betriebs in einer einzigen Datenbank erhalten. Mit Aurora PostgreSQL Limitless Database können Sie sich auf die Entwicklung umfangreicher Anwendungen konzentrieren, ohne zur Unterstützung Ihrer Workloads komplexe Lösungen für die Skalierung Ihrer Daten über mehrere DB-Instances hinweg erstellen und instandhalten zu müssen.

Weitere Informationen finden Sie unter Verwenden von Amazon Aurora PostgreSQL Limitless Database.

Aurora PostgreSQL Limitless Database ist in allen AWS-Regionen außer für Asien-Pazifik (Taipeh) verfügbar.

Unterstützte Regionen und DB-Engines für native Features der Aurora-Engine

Aurora-Datenbank-Engines unterstützen auch zusätzliche Funktionen speziell für Aurora. Einige Engine-native Funktionen werden möglicherweise nur begrenzt unterstützt oder haben eingeschränkte Berechtigungen für eine bestimmte Aurora-DB-Engine, -Version oder Region.

Themen

- Engine-native Funktionen für Aurora MySQL
- Engine-native Funktionen für Aurora PostgreSQL

Engine-native Funktionen für Aurora MySQL

Im Folgenden sind die Engine-nativen Funktionen für Aurora MySQL aufgeführt.

- Erweitertes Auditing

- Backtrack
- Fehlersimulationsabfragen
- Clusterinterne Schreibweiterleitung
- Parallele Abfrage

Engine-native Funktionen für Aurora PostgreSQL

Im Folgenden sind die Engine-native Funktionen für Aurora PostgreSQL aufgeführt.

- Babelfish
- Fehlersimulationsabfragen
- Abfrageplanverwaltung

Amazon-Aurora-Endpunktverbindungen

Bei Amazon Aurora kommt in der Regel keine einzelne Instance, sondern ein Cluster von DB-Instances zum Einsatz. Jede Verbindung wird von einer bestimmten DB-Instance verarbeitet. Wenn Sie eine Verbindung zu einem Aurora-Cluster herstellen, verweisen der von Ihnen angegebene Hostname und Port auf einen zwischengeschalteten Handler, der als Endpunkt bezeichnet wird. Aurora verwendet den Endpunktmechanismus, um diese Verbindungen zu abstrahieren. Sie müssen daher für den Ausgleich und das Umleiten von Verbindungen nicht alle Hostnamen fest codieren oder Ihre eigene Logik schreiben, wenn einige DB-Instances nicht verfügbar sind.

Bei bestimmten Aurora-Aufgaben führen verschiedene Instances oder Gruppen von Instances unterschiedliche Rollen aus. Beispielsweise verarbeitet die primäre Instance alle Data Definition Language (DDL)- und Data Manipulation Language (DML)-Anweisungen. Der schreibgeschützte Abfragedatenverkehr wird von bis zu 15 Aurora-Replicas verarbeitet.

Themen

- Typen von Aurora-Endpunkten
- Anzeigen der Endpunkte für einen Aurora-Cluster
- Funktionsweise von Aurora-Endpunkten mit hoher Verfügbarkeit
- Cluster-Endpunkte für Amazon Aurora
- Reader-Endpunkte für Amazon Aurora

- Instance-Endpunkte für Amazon Aurora
- Benutzerdefinierte Endpunkte für Amazon Aurora

Typen von Aurora-Endpunkten

Ihrem Anwendungsfall entsprechend können Sie mit Endpunkten jede Verbindung der entsprechenden Instance oder Gruppe von Instances zuordnen. Zum Ausführen von DDL-Anweisungen können Sie beispielsweise eine Verbindung mit der Instance herstellen, die als primäre Instance fungiert. Zum Durchführen von Abfragen können Sie eine Verbindung mit dem Leser-Endpunkt herstellen. Aurora führt hierbei automatisch einen Verbindungsausgleich unter allen Aurora-Replikaten aus. Bei Clustern mit DB-Instances unterschiedlicher Kapazitäten oder Konfigurationen besteht die Möglichkeit, eine Verbindung mit benutzerdefinierten Endpunkten herzustellen, die mit unterschiedlichen Untergruppen von DB-Instances verknüpft sind. Zur Diagnose und Optimierung können Sie eine Verbindung mit einem spezifischen Instance-Endpunkt herstellen, um die Details einer bestimmten DB-Instance zu untersuchen.

Ein Endpunkt wird als eine Aurora-spezifische URL dargestellt, die eine Hostadresse und einen Port enthält. In einem Aurora-DB-Cluster stehen die folgenden Endpunkt-Typen zur Verfügung:

Cluster-Endpunkt

Stellen Sie eine Verbindung zur primären Instance Ihres Clusters her, um Anwendungen zu entwickeln und zu testen und Transformationen wie `INSERT`-Anweisungen und DDL-, DML- und ETL-Operationen durchzuführen. Suchen Sie den Standort des Cluster-Endpunkts mithilfe der AWS-Managementkonsole, der AWS CLI oder der Amazon-RDS-API, wie unter Anzeigen der Endpunkte für einen Aurora-Cluster beschrieben.

Weitere Informationen über Cluster-Endpunkte finden Sie unter Cluster-Endpunkte für Amazon Aurora.

Leser-Endpunkt

Führen Sie Abfragen durch. Aurora führt automatisch einen Verbindungsausgleich zwischen allen Aurora-Replikaten durch. Suchen Sie den Standort des Leser-Endpunkts mithilfe der AWS-Managementkonsole, der AWS CLI oder der Amazon-RDS-API, wie unter Anzeigen der Endpunkte für einen Aurora-Cluster beschrieben.

Weitere Informationen über Leser-Endpunkte finden Sie unter Reader-Endpunkte für Amazon Aurora.

Instance-Endpunkt

Untersuchen Sie Details zu einer bestimmten DB-Instance zur Diagnose oder Optimierung. Sie finden den Standort des Instance-Endpunkts für jede Ihrer Instances nur in der AWS-Managementkonsole auf der Instance-Detailseite Ihrer Instance.

Weitere Informationen über Instance-Endpunkte finden Sie unter Instance-Endpunkte für Amazon Aurora.

Benutzerdefinierter Endpunkt

Stellen Sie eine Verbindung zu verschiedenen Teilmengen von DB-Instances auf dem DB-Cluster her. Dies ist nützlich, wenn Sie in Ihrem DB-Cluster über unterschiedliche Instance-Kapazitäten und -Konfigurationen verfügen. Suchen Sie die benutzerdefinierten Endpunkt-Standorte mithilfe der AWS-Managementkonsole, der AWS CLI oder der Amazon-RDS-API, wie unter Anzeigen der Endpunkte für einen Aurora-Cluster beschrieben.

Weitere Informationen über benutzerdefinierte Endpunkte finden Sie unter Benutzerdefinierte Endpunkte für Amazon Aurora.

Schreiber-Endpunkt von Aurora Global Database

Aurora Global Database verfügt über einen speziellen Endpunkt, der denselben Zweck erfüllt wie der Cluster-Endpunkt eines eigenständigen Aurora-Clusters. Er verarbeitet sowohl Schreib- als auch Leseanfragen. Wenn ein sekundärer Cluster aufgrund einer Umstellung oder eines Failovers zum neuen primären Cluster wird, stellt Aurora diesen Endpunkt automatisch so um, dass er auf den Cluster-Endpunkt des neuen primären Clusters in der anderen AWS-Region verweist. Auf diese Weise müssen Sie die AWS-Region nicht in der Verbindungszeichenfolge für Ihre Anwendung codieren und Sie müssen die Verbindungszeichenfolge nicht ändern, wenn sich das Layout der globalen Datenbank ändert. Aurora erstellt diesen Endpunkt, wenn Sie eine Aurora Global Database einrichten, indem Sie beispielsweise Region hinzufügen für einen Aurora-Cluster in der AWS-Managementkonsole auswählen.

Weitere Informationen darüber, wie Sie diesen Endpunkttyp mit Aurora Global Database verwenden können, finden Sie unter Verbinden mit Amazon Aurora Global Database.

Anzeigen der Endpunkte für einen Aurora-Cluster

Sie können den Standort des Instance-Endpunkts zwar nur auf der Instance-Detailseite in der AWS-Managementkonsole finden, aber Sie können die Konsole, die AWS CLI oder die Amazon-RDS-API verwenden, um die Standorte von Cluster-, Leser- und benutzerdefinierten Endpunkten zu finden.

Console

In der AWS-Managementkonsole können Sie den Cluster-Endpunkt, den Leser-Endpunkt und alle benutzerdefinierten Endpunkte auf der Instance-Detailseite Ihres Clusters anzeigen. Der Instance-Endpunkt wird auf der Detailseite der jeweiligen Instance angezeigt. Fügen Sie beim Herstellen der Verbindung dem auf der Detailseite angezeigten Namen des Endpunkts die zugewiesene Portnummer (mit einem Doppelpunkt hinter der Nummer) an.

AWS CLI

Mit der AWS CLI finden Sie die Schreiber-, die Leser- und alle benutzerdefinierten Endpunkte in der Ausgabe des Befehls describe-db-cluster. Der folgende Befehl zeigt beispielsweise die Endpunktattribute für alle Cluster in Ihrer aktuellen AWS-Region an.

```
aws rds describe-db-clusters --query '*[].
{Endpoint:Endpoint,ReaderEndpoint:ReaderEndpoint,CustomEndpoints:CustomEndpoints}'
```

Amazon RDS API

Mit der Amazon-RDS-API rufen Sie die Endpunkte ab, indem Sie die Operation DescribeDBClusterEndpoints aufrufen.

Funktionsweise von Aurora-Endpunkten mit hoher Verfügbarkeit

Verwenden Sie im Fall von Clustern, bei denen eine hohe Verfügbarkeit von großer Bedeutung ist, den Cluster-Endpunkt für Lese-/Schreib- oder Allzweck-Verbindungen und den Leser-Endpunkt für schreibgeschützte Verbindungen. Die Schreiber- und Leser-Endpunkte verwalten das Failover von DB-Instances besser als Instance-Endpunkte. Im Gegensatz zu den Instance-Endpunkten ändern die Schreiber- und Leser-Endpunkte automatisch, mit welcher DB-Instance sie eine Verbindung herstellen, wenn eine DB-Instance in Ihrem Cluster nicht mehr verfügbar ist. Weitere Informationen über Cluster- und Leser-Endpunkte finden Sie unter Cluster-Endpunkte für Amazon Aurora und Reader-Endpunkte für Amazon Aurora.

Wenn die primäre DB-Instance eines DB-Clusters ausfällt, führt Aurora automatisch ein Failover zu einer neuen primären DB-Instance durch. Dies geschieht entweder durch Hochstufung einer vorhandenen Aurora-Replica zu einer neuen primären DB-Instance oder durch Erstellen einer neuen primären DB-Instance. Bei einem Failover können Sie den Cluster-Endpunkt verwenden, um die Verbindung zu der neu hochgestuften oder erstellten primären DB-Instance wiederherzustellen. Sie haben auch die Möglichkeit, den Leser-Endpunkt zu verwenden, um eine Verbindung zu einem der

Aurora-Replikate im DB-Cluster wiederherzustellen. Während eines Failovers leitet der Reader-Endpunkt Verbindungen möglicherweise für kurze Zeit an die neue primäre DB-Instance eines DB-Clusters weiter, nachdem ein Aurora-Replica zur neuen primären DB-Instance hochgestuft wurde.

Wenn Sie Ihre Anwendungslogik so entwickeln, dass Verbindungen zu Instance-Endpunkten verwaltet werden können, ist es möglich, den daraus resultierenden Satz an verfügbaren DB-Instances im DB-Cluster manuell oder programmgesteuert zu ermitteln. Verwenden Sie den AWS CLI-Befehl describe-db-clusters oder die DescribeDBClusters-RDS-API-Operation, um die DB-Cluster- und Leser-Endpunkte, die DB-Instances, deren Leserstatus sowie deren Hochstufungsebenen zu finden. Sie können dann die Instance-Klassen nach einem Failover bestätigen und eine Verbindung mit einem geeigneten Instance-Endpunkt herstellen.

Weitere Informationen zu Failovers finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.

Weitere Informationen über Hochverfügbarkeit in Amazon Aurora finden Sie unter Hohe Verfügbarkeit für Amazon Aurora.

Cluster-Endpunkte für Amazon Aurora

Ein Cluster-Endpunkt (oder Writer-Endpunkt) für einen Aurora-DB-Cluster stellt eine Verbindung zur aktuellen primären DB-Instance für diesen DB-Cluster her. Nur dieser Endpunkt kann Schreibvorgänge wie z. B. DDL-Anweisungen durchführen. Aus diesem Grund sollten Sie eine Verbindung zum Cluster-Endpunkt herstellen, wenn Sie zum ersten Mal einen Cluster einrichten oder Ihr Cluster nur eine DB-Instance enthält.

Jeder Aurora-DB-Cluster verfügt über einen Cluster-Endpunkt und eine primäre DB-Instance.

Sie verwenden den Cluster-Endpunkt für alle Schreibvorgänge auf dem DB-Cluster, darunter Einfügungs-, Aktualisierungs- und Löschvorgänge sowie DDL-Änderungen. Sie können den Cluster-Endpunkt auch für Lesevorgänge nutzen, beispielsweise Abfragen.

Der Cluster-Endpunkt bietet Failover-Support für Lese-/Schreibverbindungen zum DB-Cluster. Wenn die aktuelle primäre DB-Instance eines DB-Clusters ausfällt, führt Aurora automatisch ein Failover zu einer neuen primären DB-Instance durch. Während eines Failovers bedient der DB-Cluster weiterhin Verbindungsanfragen von der neuen primären DB-Instance an den Cluster-Endpunkt mit minimaler Serviceunterbrechung.

Das folgende Beispiel zeigt einen Cluster-Endpunkt für einen Aurora MySQL-DB-Cluster.

```
mydbcluster.cluster-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Jeder Aurora-Cluster verfügt über einen integrierten Cluster-Endpunkt, dessen Name zusammen mit anderen Attributen durch Aurora verwaltet wird. Sie können einen solchen Endpunkt nicht erstellen, löschen oder ändern.

Den Cluster-Endpunkt verwenden Sie beim Verwalten Ihres Clusters, bei Extraktions-, Transformations- und Ladevorgängen (ETL) oder beim Entwickeln und Testen von Anwendungen. Der Cluster-Endpunkt stellt eine Verbindung zur primären Instance des Clusters her. Die primäre Instance ist die einzige DB-Instance, mit der Sie Tabellen und Indizes erstellen sowie `INSERT`-Anweisungen und andere DDL- und DML-Operationen ausführen können.

Die physische IP-Adresse, auf die der Cluster-Endpunkt verweist, wird geändert, wenn der Failover-Mechanismus für den Cluster eine neue DB-Instance zur primären Instance für Lese-Schreib-Operationen hochstuft. Wenn Sie irgendeine Form von Verbindungspooling oder eine andere Art von Multiplexing verwenden, müssen Sie möglicherweise zwischengespeicherte DNS-Informationen löschen oder die entsprechende Time-to-Live reduzieren. Hierdurch wird sichergestellt, dass Sie keine Lese-Schreib-Verbindung mit einer DB-Instance herstellen, die nicht mehr verfügbar ist oder nach einem Failover schreibgeschützt ist.

Reader-Endpunkte für Amazon Aurora

Ein Reader-Endpunkt für einen Aurora-DB-Cluster bietet Verbindungsausgleichsunterstützung für schreibgeschützte Verbindungen zum DB-Cluster. Verwenden Sie den Leser-Endpunkt für Lesevorgänge, beispielsweise Abfragen. Durch die Verarbeitung dieser Anweisungen auf den schreibgeschützten Aurora-Replikaten reduziert dieser Endpunkt den Overhead für die primäre Instance. Er hilft dem Cluster auch, die Kapazität für gleichzeitige `SELECT`-Abfragen zu skalieren, und dies proportional zur Anzahl der Aurora-Replikate im Cluster. Jeder Aurora-DB-Cluster verfügt über einen Reader-Endpunkt.

Wenn der Cluster mindestens ein Aurora-Replikat enthält, gleicht der Reader-Endpunkt jede Verbindungsanforderung zwischen den Aurora-Replikaten aus. In diesem Fall können Sie nur schreibgeschützte Anweisungen wie `SELECT` in dieser Sitzung ausführen. Wenn der Cluster nur eine primäre Instance und keine Aurora-Replikate enthält, stellt der Reader-Endpunkt eine Verbindung zur primären Instance her. In diesem Fall können Sie Schreibvorgänge über den Endpunkt ausführen.

Das folgende Beispiel zeigt einen Leser-Endpunkt für einen Aurora MySQL-DB-Cluster.

```
mydbcluster.cluster-ro-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Sie nutzen den Reader-Endpunkt für schreibgeschützte Verbindungen Ihres Aurora-Clusters. Der Endpunkt nutzt einen Verbindungsausgleichsmechanismus, um Ihren Cluster bei der Verarbeitung abfrageintensiver Workloads zu unterstützen. Der Reader-Endpunkt ist der Endpunkt, den Sie Anwendungen zur Verfügung stellen, die Berichterstattung oder andere schreibgeschützte Operationen auf dem Cluster ausführen.

Der Reader-Endpunkt sorgt für den Ausgleich der Verbindungen zu verfügbaren Aurora-Replikaten in einem Aurora-DB-Cluster. Es wird kein Ausgleich für individuelle Abfragen durchgeführt. Wenn Sie zur Verteilung der Workload-Auslastung durch Lesevorgänge für einen DB-Cluster einen Ausgleich für jede Abfrage durchführen möchten, öffnen Sie für jede Abfrage eine neue Verbindung zum Reader-Endpunkt.

Jeder Aurora-Cluster verfügt über einen integrierten Cluster-Endpunkt verfügt, dessen Name zusammen mit anderen Attributen durch Aurora verwaltet wird. Sie können einen solchen Endpunkt nicht erstellen, löschen oder ändern.

Wenn Ihr Cluster nur ein primäres Ziel (Instance oder DB-Shard-Gruppe) und keine Aurora-Replikate enthält, stellt der Reader-Endpunkt eine Verbindung zur primären Instance her. In diesem Fall können Sie Schreibvorgänge über diesen Endpunkt ausführen.

> **Tip**
>
> Über RDS-Proxy können Sie zusätzliche schreibgeschützte Endpunkte für einen Aurora-Cluster erstellen. Diese Endpunkte führen den gleichen Verbindungsausgleich wie der Aurora-Reader-Endpunkt aus. Anwendungen können sich schneller mit den Proxy-Endpunkten wieder verbinden als der Aurora-Reader-Endpunkt, wenn Reader-Instances nicht verfügbar sind. Die Proxy-Endpunkte können auch andere Proxy-Funktionen wie Multiplexing nutzen. Weitere Informationen finden Sie unter Verwenden von Reader-Endpunkten mit Aurora-Clustern.

Instance-Endpunkte für Amazon Aurora

Ein Instance-Endpunkt stellt innerhalb eines Aurora-Clusters eine Verbindung zu einer spezifischen DB-Instance her. Jede DB-Instance in einem DB-Cluster hat einen eigenen, spezifischen Instance-Endpunkt. Daher ist ein Instance-Endpunkt für die aktuelle primäre DB-Instance des DB-Clusters vorhanden und es steht ein Instance-Endpunkt für jedes der Aurora-Replicas im DB-Cluster zur Verfügung.

Der Instance-Endpunkt bietet direkte Kontrolle über Verbindungen zum DB-Cluster für Szenarien, in denen die Verwendung des Cluster-Endpunkts oder des Leser-Endpunkts möglicherweise nicht angemessen ist. Beispiel: Ihre Client-Anwendung erfordert möglicherweise einen detaillierteren Verbindungsausgleich je nach Workload-Typ. In diesem Fall können Sie mehrere Clients so konfigurieren, dass sie Verbindungen zu verschiedenen Aurora-Replicas in einem DB-Cluster herstellen, um die Lese-Workloads zu verteilen. Ein Beispiel für die Verwendung von Instance-Endpunkten zur Verbesserung der Verbindungsgeschwindigkeit nach einem Failover für Aurora PostgreSQL finden Sie unter Schnelles Failover mit Amazon Aurora PostgreSQL. Ein Beispiel für die Verwendung von Instance-Endpunkten zur Verbesserung der Verbindungsgeschwindigkeit nach einem Failover für Aurora MySQL finden Sie unter MariaDB Connector/J failover support - case Amazon Aurora.

Das folgende Beispiel zeigt einen Instance-Endpunkt für eine DB-Instance in einem Aurora MySQL-DB-Cluster.

```
mydbinstance.c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Jede DB-Instance in einem Aurora-Cluster verfügt über ihren eigenen integrierten Instance-Endpunkt, dessen Name und andere Attribute von Aurora verwaltet werden. Sie können einen solchen Endpunkt nicht erstellen, löschen oder ändern. Möglicherweise sind Sie mit Instance-Endpunkten vertraut, wenn Sie Amazon RDS verwenden. Bei Aurora verwenden Sie die Schreiber- und Leser-Endpunkte jedoch häufiger als die Instance-Endpunkte.

Bei täglichen Aurora-Operationen werden Instance-Endpunkte hauptsächlich dazu verwendet, um Kapazitäts- oder Leistungsprobleme zu untersuchen, die eine spezifische Instance in einem Aurora-Cluster betreffen. Während eine Verbindung zu einer spezifischen Instance besteht, können Sie unter anderem deren Statusvariablen oder Metriken untersuchen. Hierdurch ist es möglich, Unterschiede zwischen den Aktivitäten verschiedener Cluster-Instances zu ermitteln.

In fortgeschrittenen Anwendungsfällen konfigurieren Sie einige DB-Instances möglicherweise anders als andere DB-Instances. Verwenden Sie in diesem Fall den Instance-Endpunkt, um eine direkte Verbindung mit einer Instance herzustellen, die kleiner oder größer ist oder sich auf eine andere Art von den anderen Instances unterscheidet. Legen Sie auch eine Failover-Priorität fest, damit diese spezielle DB-Instance bei einer Hochstufung zur primären Instance als letztes berücksichtigt wird. In solchen Fällen empfehlen wir, dass Sie nicht den Instance-Endpunkt, sondern benutzerdefinierte Endpunkte verwenden. Diese Vorgehensweise vereinfacht die Verbindungsverwaltung und trägt zur Gewährleistung von hoher Verfügbarkeit bei, wenn Sie Ihrem Cluster weitere DB-Instances hinzufügen.

Benutzerdefinierte Endpunkte für Amazon Aurora

Ein benutzerdefinierter Endpunkt für einen Aurora-Cluster ist ein von Ihnen gewählter Satz an DB-Instances. Wenn Sie eine Verbindung mit dem Endpunkt herstellen, führt Aurora einen Verbindungsausgleich durch und wählt eine Instance aus der Gruppe aus, um die Verbindung zu ermöglichen. Sie bestimmen, auf welche Instances sich dieser Endpunkt bezieht, und Sie entscheiden auch, welchen Zweck der Endpunkt erfüllen soll.

Ein Aurora-DB-Cluster verfügt über keine benutzerdefinierten Endpunkte. Ein benutzerdefinierter Endpunkt muss daher von Ihnen erstellt werden. Sie können für jeden bereitgestellten Aurora-Cluster oder Cluster von Aurora Serverless v2 bis zu fünf benutzerdefinierte Endpunkte erstellen. Für Aurora Serverless v1-Cluster können keine benutzerdefinierten Endpunkte verwendet werden.

Der benutzerdefinierte Endpunkt stellt Datenbankverbindungen mit Lastausgleich bereit. Diese basieren auf anderen Kriterien als die schreibgeschützte oder die Lese-Schreib-Funktion der DB-Instances. So können Sie z. B. einen benutzerdefinierten Endpunkt definieren, um eine Verbindung mit Instances herzustellen, die eine bestimmte AWS-Instance-Klasse oder eine bestimmte DB-Parametergruppe verwenden. Sie können dann bestimmte Benutzergruppen über diesen benutzerdefinierten Endpunkt informieren. Beispielsweise können Sie internen Benutzern Instances mit geringer Kapazität für die Berichterstellung oder Ad-hoc-Abfragen (einmalig) und dem Produktionsdatenverkehr Instances mit hohen Kapazitäten zuweisen.

Da die Verbindung mit jeder mit dem benutzerdefinierten Endpunkt verknüpften DB-Instance erfolgen kann, sollten Sie sicherstellen, dass alle DB-Instances innerhalb dieser Gruppe bestimmte gemeinsame Eigenschaften aufweisen. Hierdurch wird sichergestellt, dass Leistung, Speicherkapazität usw. für alle Verbindungen mit diesem Endpunkt konsistent verfügbar sind.

Diese Funktion richtet sich an fortgeschrittene Benutzer mit besonderen Arten von Workloads, bei denen es nicht sinnvoll ist, wenn alle Aurora-Replicas im Cluster identisch sind. Mit benutzerdefinierten Endpunkten können Sie die Kapazität der DB-Instance vorhersagen, die für jede Verbindung verwendet wird. Bei der Verwendung von benutzerdefinierten Endpunkten wird der Reader-Endpunkt für diesen Cluster üblicherweise nicht verwendet.

Das folgende Beispiel zeigt einen benutzerdefinierten Endpunkt für eine DB-Instance in einem Aurora MySQL-DB-Cluster.

```
myendpoint.cluster-custom-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Benutzerdefinierte Endpunkte werden verwendet, um die Verbindungsverwaltung zu vereinfachen, wenn Ihr Cluster DB-Instances mit unterschiedlichen Kapazitäten und Konfigurationseinstellungen enthält.

Zuvor hätten Sie möglicherweise den CNAMES-Mechanismus verwendet, um DNS (Domain Name Service)-Aliasse Ihrer eigenen Domäne einzurichten und ähnliche Ergebnisse zu erzielen. Durch benutzerdefinierte Endpunkte müssen CNAME-Datensätze bei zu- oder abnehmender Größe Ihres Clusters nicht aktualisiert werden. Mit benutzerdefinierten Endpunkten können auch verschlüsselte Transport Layer Security (TLS)- und Secure Sockets Layer (SSL)-Verbindungen verwendet werden.

Anstatt für jeden bestimmten Zweck eine DB-Instance zu verwenden und eine Verbindung zu deren Instance-Endpunkt herzustellen, können Sie mehrere Gruppen spezifischer DB-Instances nutzen. In diesem Fall verfügt jede Gruppe über ihren eigenen benutzerdefinierten Endpunkt. Hierdurch kann Aurora einen Verbindungsausgleich unter allen Instances durchführen, die für Aufgaben wie Berichterstellung, die Verarbeitung der Produktionslast oder interne Abfragen bestimmt sind. Die benutzerdefinierten Endpunkte verteilen die Verbindungen passiv auf die Instances und verwenden DNS, um die IP-Adresse einer der Instances nach dem Zufallsprinzip zurückzugeben. Sollte eine DB-Instance innerhalb einer Gruppe nicht mehr verfügbar sein, leitet Aurora nachfolgende Verbindungen benutzerdefinierter Endpunkte an eine der anderen DB-Instances weiter, die mit demselben Endpunkt verknüpft sind.

Themen

- Überlegungen zu benutzerdefinierten Endpunkten in Amazon Aurora
- Erstellen eines benutzerdefinierten Endpunkts
- Anzeigen von benutzerdefinierten Endpunkten
- Bearbeiten eines benutzerdefinierten Endpunkts
- Löschen eines benutzerdefinierten Endpunkts
- AWS CLI-Beispiele zu benutzerdefinierten Endpunkten für Amazon Aurora

Überlegungen zu benutzerdefinierten Endpunkten in Amazon Aurora

Verwenden Sie die Informationen in den folgenden Abschnitten, um benutzerdefinierte Endpunkte zu verwalten, Eigenschaften dafür anzugeben und Mitgliedschaftsregeln zu verwenden.

Themen

- Verwalten von benutzerdefinierten Endpunkten

- Angeben der Eigenschaften für benutzerdefinierte Endpunkte
- Mitgliedschaftsregeln für benutzerdefinierte Endpunkte

Verwalten von benutzerdefinierten Endpunkten

Da neu erstellte Aurora-Cluster über keine benutzerdefinierten Endpunkte verfügen, müssen Sie diese Objekte selbst erstellen und verwalten. Verwenden Sie hierzu die AWS-Managementkonsole, die AWS CLI oder die Amazon-RDS-API.

> **Note**
>
> Sie müssen auch benutzerdefinierte Endpunkte für Aurora-Cluster erstellen und verwalten, die anhand von Snapshots wiederhergestellt wurden. Benutzerdefinierte Endpunkte sind nicht im Snapshot enthalten. Sie erstellen sie nach der Wiederherstellung erneut und wählen neue Endpunktnamen, wenn sich der wiederhergestellte Cluster in der gleichen Region wie der ursprüngliche Cluster befindet.

Navigieren Sie zur Detailseite für Ihren Aurora-Cluster und nutzen Sie die Steuerelemente im Abschnitt Benutzerdefinierte Endpunkte, um über die AWS-Managementkonsole mit benutzerdefinierten Endpunkten zu arbeiten.

Zum Arbeiten mit benutzerdefinierten Endpunkten in der AWS CLI können Sie die folgenden Operationen verwenden:

- create-db-cluster-endpoint
- describe-db-cluster-endpoints
- modify-db-cluster-endpoint
- delete-db-cluster-endpoint

Sie können die folgenden Funktionen verwenden, um über die Amazon-RDS-API mit benutzerdefinierten Endpunkten zu arbeiten:

- CreateDBClusterEndpoint
- DescribeDBClusterEndpoints
- ModifyDBClusterEndpoint
- DeleteDBClusterEndpoint

Angeben der Eigenschaften für benutzerdefinierte Endpunkte

Die maximale Länge des Namens für einen benutzerdefinierten Endpunkt beträgt 63 Zeichen. Der Name hat folgendes Format:

```
endpoint_name.cluster-custom-customer_DNS_identifier.AWS_Region.rds.amazonaws.com
```

Sie können den Namen eines benutzerdefinierten Endpunkts nicht für mehr als einen Cluster in derselben AWS-Region verwenden. Die benutzerdefinierte DNS-ID ist eine eindeutige Kennung, die mit Ihrem AWS-Konto in einer bestimmten AWS-Region verknüpft ist.

Jedem benutzerdefinierten Endpunkt ist ein Typ zugeordnet, der bestimmt, welche DB-Instances diesem Endpunkt zugeordnet werden können. Zurzeit kann der Typ entweder `READER` oder `ANY` sein. Für die benutzerdefinierten Endpunkttypen gelten die folgenden Überlegungen:

- Sie können den benutzerdefinierten Endpunkttyp nicht in der AWS-Managementkonsole auswählen. Alle von Ihnen mit der AWS-Managementkonsole erstellten benutzerdefinierten Endpunkte haben den Typ `ANY`.

  Den benutzerdefinierten Endpunkttyp können Sie mithilfe der AWS CLI oder der Amazon-RDS-API festlegen und ändern.
- Nur Reader-DB-Instances können Teil eines benutzerdefinierten `READER`-Endpunkts sein.
- Sowohl Reader- als auch Writer-DB-Instances können Teil eines benutzerdefinierten `ANY`-Endpunkts sein. Aurora leitet Verbindungen zu Cluster-Endpunkten mit Typ `ANY` mit gleicher Wahrscheinlichkeit zu jeder zugehörigen DB-Instance. Der Typ `ANY` gilt für Cluster mit allen Replikationstopologien.
- Wenn Sie versuchen, einen benutzerdefinierten Endpunkt mit einem Typ zu erstellen, der mit der Replikationskonfiguration eines Clusters nicht kompatibel ist, zeigt Aurora eine Fehlermeldung an.

Mitgliedschaftsregeln für benutzerdefinierte Endpunkte

Wenn Sie eine DB-Instance einem benutzerdefinierten Endpunkt hinzufügen oder aus einem benutzerdefinierten Endpunkt entfernen, bleiben alle vorhandenen Verbindungen zu dieser DB-Instance bestehen.

Sie können eine Liste mit DB-Instances definieren, die im benutzerdefinierten Endpunkt enthalten oder aus diesem ausgeschlossen sein sollen. Wir bezeichnen diese Listen als statische bzw. als Ausschlusslisten. Sie können den Aufnahme- bzw. Ausschlussmechanismus verwenden, um die

DB-Instances-Gruppen weiter aufzuteilen und sicherzustellen, dass der Satz an benutzerdefinierten Endpunkten alle DB-Instances im Cluster abdeckt. Jeder benutzerdefinierte Endpunkt kann nur einen dieser Listentypen enthalten.

In AWS-Managementkonsole:

- Die Auswahl wird durch das Kontrollkästchen Attach future instances added to this cluster (Zukünftige Instances diesem Cluster anfügen) dargestellt. Wenn Sie das Kontrollkästchen deaktivieren, verwendet der benutzerdefinierte Endpunkt eine statische Liste, die nur die auf der Seite angegebenen DB-Instances enthält. Wenn Sie das Kontrollkästchen aktivieren, verwendet der benutzerdefinierte Endpunkt eine Ausschlussliste. In diesem Fall repräsentiert der benutzerdefinierte Endpunkt alle DB-Instances im Cluster (einschließlich zukünftig hinzugefügter DB-Instances) außer jenen, die auf der Seite nicht ausgewählt wurden.
- In der Konsole können Sie den Endpunkttyp nicht angeben. Jeder benutzerdefinierte Endpunkt, der mit der Konsole erstellt wurde, weist den Typ `ANY` auf.

  Daher ändert Aurora die Mitgliedschaft des benutzerdefinierten Endpunkts nicht, wenn DB-Instances aufgrund eines Failovers oder einer Hochstufung die Rollen zwischen Writer und Reader ändern.

In der AWS CLI und Amazon-RDS-API:

- Sie können den Endpunkttyp angeben. Wenn der Endpunkttyp auf `READER` gesetzt ist, wird die Endpunkt-Mitgliedschaft daher bei Failovers und Hochstufungen automatisch angepasst.

  Beispielsweise enthält ein benutzerdefinierter Endpunkt des Typs `READER` ein Aurora-Replikat, das anschließend zu einer Writer-DB-Instance hochgestuft wird. Die neue Writer-Instance ist nicht mehr Teil des benutzerdefinierten Endpunkts.
- Sie können einzelne Mitglieder den Listen hinzufügen und sie aus den Listen entfernen, nachdem sie ihre Rollen geändert haben. Verwenden Sie den AWS CLI-Befehl modify-db-cluster-endpoint oder die API-Operation ModifyDBClusterEndpoint.

Sie können eine DB-Instance mit mehreren benutzerdefinierten Endpunkten verknüpfen. Angenommen, Sie fügen einem Cluster eine neue DB-Instance hinzu, oder eine DB-Instance wird von Aurora automatisch durch den Mechanismus der automatischen Skalierung hinzugefügt. In diesen Fällen wird die DB-Instance allen infrage kommenden benutzerdefinierten Endpunkten hinzugefügt. Welchen Endpunkten die DB-Instance hinzugefügt wird, hängt vom benutzerdefinierten

Endpunkt des Typs `READER` oder `ANY` sowie von statischen Listen oder Ausschlusslisten ab, die für jeden Endpunkt definiert wurden. Wenn der Endpunkt beispielsweise eine statische Liste von DB-Instances enthält, werden kürzlich hinzugefügte Aurora-Replicas nicht diesem Endpunkt hinzugefügt. Wenn der Endpunkt über eine Ausschlussliste verfügt, werden neu hinzugefügte Aurora-Replicas dem Endpunkt hinzugefügt, wenn sie nicht in der Ausschlussliste aufgeführt sind und ihre Rollen dem Typ des benutzerdefinierten Endpunkts entsprechen.

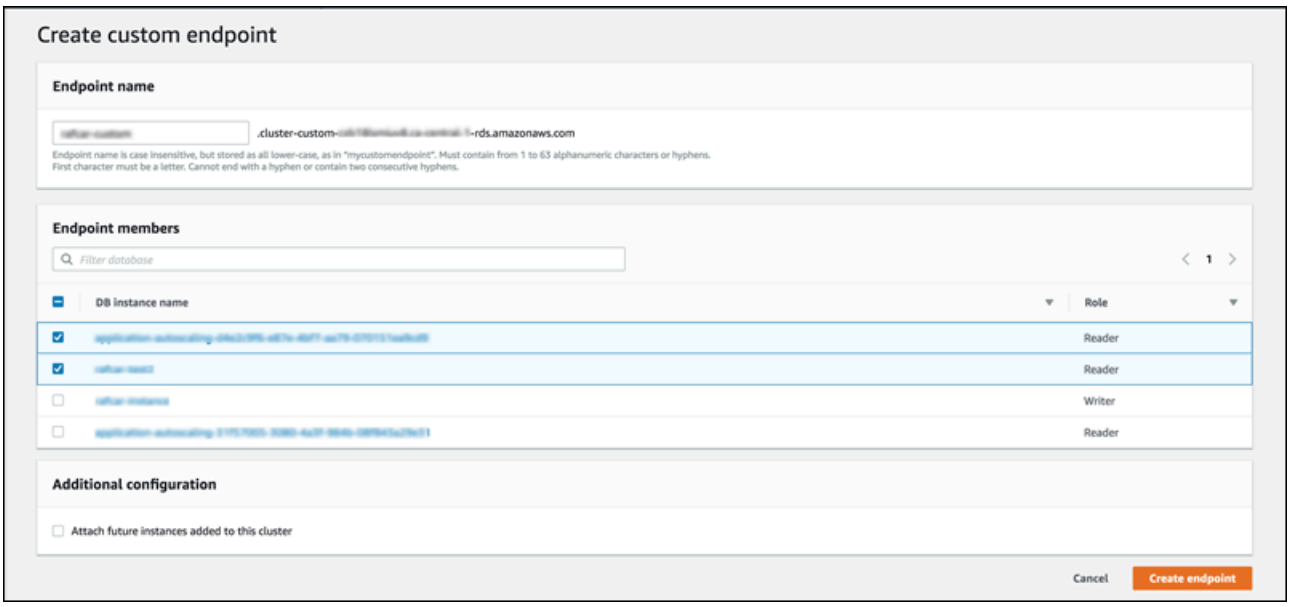
Wenn ein Aurora-Replica nicht mehr verfügbar ist, bleibt es weiterhin mit benutzerdefinierten Endpunkten verknüpft. Das Replica bleibt Teil des benutzerdefinierten Endpunkts, wenn es z. B. nicht voll funktionsfähig ist, angehalten wurde oder neu startet. Sie können jedoch erst dann eine Verbindung über diese Endpunkte zum Replica herstellen, wenn es wieder verfügbar ist.

Erstellen eines benutzerdefinierten Endpunkts

Erstellen Sie einen benutzerdefinierten Endpunkt mit der AWS-Managementkonsole, der AWS CLI oder der Amazon-RDS-API.

Konsole

Zum Erstellen eines benutzerdefinierten Endpunkts mit der AWS-Managementkonsole besuchen Sie die Cluster-Detailseite und wählen Sie im Abschnitt Endpoints (Endpunkte) die Aktion `Create custom endpoint` aus. Wählen Sie einen Namen für den benutzerdefinierten Endpunkt aus. Dieser muss für Ihre Benutzer-ID und Region eindeutig sein. Um eine Liste von DB-Instances auszuwählen, die auch bei einer Erweiterung des Clusters gleich bleibt, lassen Sie das Kontrollkästchen Attach future instances added to this cluster (Zukünftige zu diesem Cluster hinzugefügte Instances anhängen) deaktiviert. Wenn Sie dieses Kontrollkästchen aktivieren, fügt der benutzerdefinierte Endpunkt dynamisch alle neuen Instances hinzu, wenn Sie sie zum Cluster hinzufügen.



Sie können den benutzerdefinierten Endpunkttyp nicht in der AWS-Managementkonsole auswählen. Alle von Ihnen mit der AWS-Managementkonsole erstellten benutzerdefinierten Endpunkte haben den Typ ANY.

AWS CLI

Zum Erstellen eines benutzerdefinierten Endpunkts mit der AWS CLI führen Sie den Befehl create-db-cluster-endpoint aus.

Mit folgendem Befehl wird ein benutzerdefinierter Endpunkt erstellt, der an einen bestimmten Cluster angefügt ist. Zu Beginn wird der Endpunkt mit allen Aurora-Replica-Instances des Clusters verknüpft. Über einen nachfolgenden Befehl wird er mit einem spezifischen Satz an DB-Instances im Cluster verknüpft.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --endpoint-type reader \
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --static-members instance_name_1 instance_name_2
```

Für Windows:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --endpoint-type reader ^
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --static-members instance_name_1 instance_name_2
```

RDS-API

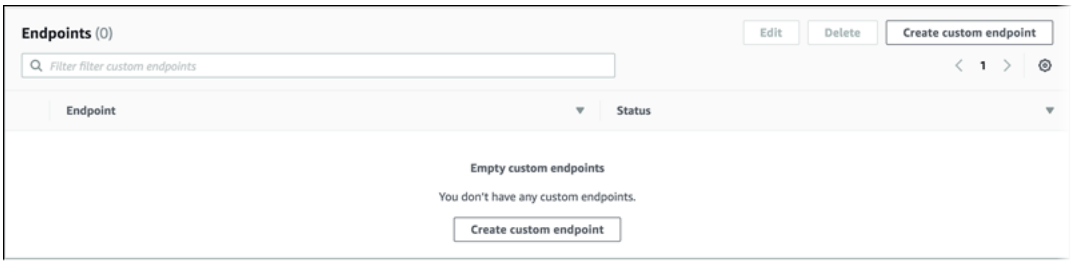
Zum Erstellen eines benutzerdefinierten Endpunkts mit der RDS-API führen Sie die Aktion CreateDBClusterEndpoint aus.

Anzeigen von benutzerdefinierten Endpunkten

Konsole

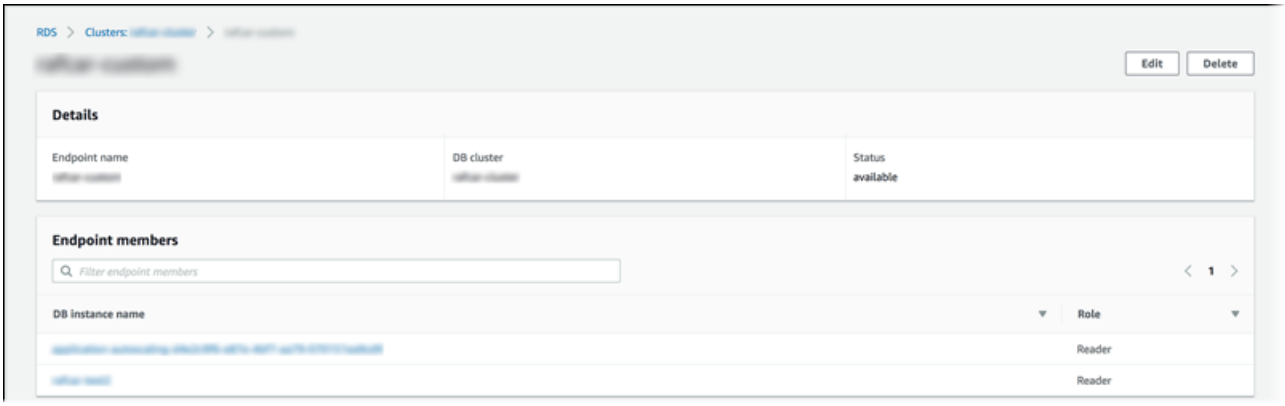
Zum Anzeigen von benutzerdefinierten Endpunkten mit der AWS-Managementkonsole gehen Sie zur Cluster-Detailseite des entsprechenden Clusters und rufen den Abschnitt Endpoints (Endpunkte) auf. Dieser Abschnitt enthält nur Informationen zu benutzerdefinierten Endpunkten. Die Details der integrierten Endpunkte sind im Abschnitt zu den wichtigsten Details aufgeführt. Um die Details für einen spezifischen benutzerdefinierten Endpunkt anzuzeigen, wählen Sie dessen Namen aus. Daraufhin wird die Detailseite für diesen Endpunkt geöffnet.

Der folgende Screenshot zeigt, dass die Liste der benutzerdefinierten Endpunkte für einen Aurora-Cluster zu Beginn leer ist.



Nachdem Sie einige benutzerdefinierte Endpunkte für diesen Cluster erstellt haben, werden diese im Abschnitt Endpoints (Endpunkte) angezeigt.

Auf der Detailseite sehen Sie, mit welchen DB-Instances der Endpunkt derzeit verknüpft ist.



Um darüber hinaus anzuzeigen, ob dem Cluster neu hinzugefügte DB-Instances automatisch auch in den Endpunkt integriert werden, rufen Sie die Seite Bearbeiten für den Endpunkt auf.

AWS CLI

Zum Anzeigen benutzerdefinierter Endpunkte mit der AWS CLI führen Sie den Befehl describe-db-cluster-endpoints aus.

Mit folgendem Befehl werden die benutzerdefinierten Endpunkte angezeigt, die mit einem bestimmten Cluster in einer bestimmten Region verknüpft sind. Die Ausgabe beinhaltet sowohl die integrierten Endpunkte als auch jeden benutzerdefinierten Endpunkt.

Für Linux, macOS oder Unix:

```
aws rds describe-db-cluster-endpoints --region region_name \
  --db-cluster-identifier cluster_id
```

Für Windows:

```
aws rds describe-db-cluster-endpoints --region region_name ^
  --db-cluster-identifier cluster_id
```

Im folgenden Beispiel wird der Befehl `describe-db-cluster-endpoints` mit einer Beispielausgabe gezeigt. Der `EndpointType` von `WRITER` oder `READER` bezeichnet die integrierten Schreib- und Lese- sowie schreibgeschützten Endpunkte des Clusters. Der `EndpointType` `CUSTOM` bezeichnet Endpunkte, die von Ihnen erstellt und deren verknüpfte DB-Instances von Ihnen ausgewählt werden. Einer der Endpunkte verfügt über ein Feld `StaticMembers`, das angibt, dass der Endpunkt mit einem genau bestimmten Satz an DB-Instances verknüpft ist. Dieses Feld kann

nicht leer sein. Der andere Endpunkt verfügt über ein Feld `ExcludedMembers`, das angibt, dass der Endpunkt mit anderen als den unter `ExcludedMembers` aufgeführten DB-Instances verknüpft ist. Auch dieses Feld kann nicht leer sein. Die zweite Art benutzerdefinierter Endpunkte wird größer, um neue Instances zu berücksichtigen, die dem Cluster hinzufügt werden.


```
{
 "DBClusterEndpoints": [
  {
   "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "WRITER"
  },
  {
   "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "READER"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "powers-of-2",
   "ExcludedMembers": [],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "powers-of-2.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-08",
     "custom-endpoint-demo-01",
     "custom-endpoint-demo-02"
   ],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:powers-of-2"
  },
  {
   "CustomEndpointType": "ANY",
```

```
   "DBClusterEndpointIdentifier": "eight-and-higher",
   "ExcludedMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-02",
     "custom-endpoint-demo-07",
     "custom-endpoint-demo-05",
     "custom-endpoint-demo-03",
     "custom-endpoint-demo-06",
     "custom-endpoint-demo-01"
   ],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHYQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:eight-and-higher"
  }
 ]
}
```

RDS-API

Zum Anzeigen benutzerdefinierter Endpunkte mit der RDS-API führen Sie die Aktion DescribeDBClusterEndpoints.html aus.

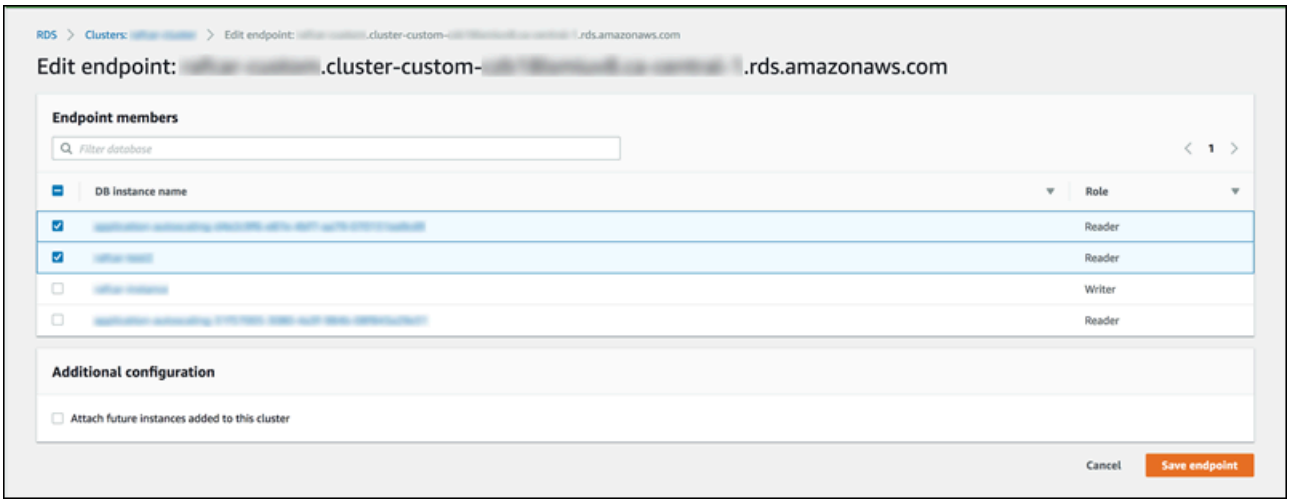
Bearbeiten eines benutzerdefinierten Endpunkts

Sie können die Eigenschaften eines benutzerdefinierten Endpunkts bearbeiten, um die mit dem Endpunkt verknüpften DB-Instances zu ändern. Sie können bei einem Endpunkt auch zwischen einer statischen und einer Ausschlussliste wechseln. Weitere Informationen zu diesen Endpunkteigenschaften finden Sie unter Mitgliedschaftsregeln für benutzerdefinierte Endpunkte.

Sie können einen benutzerdefinierten Endpunkt weiterhin verwenden oder eine Verbindung zu diesem herstellen, wenn die Änderungen einer Bearbeitungsaktion noch verarbeitet werden.

Konsole

Zum Erstellen eines benutzerdefinierten Endpunkts mit der AWS-Managementkonsole wählen Sie den Endpunkt auf der Cluster-Detailseite aus oder rufen die Detailseite des Endpunkts auf und wählen dort die Aktion Edit (Bearbeiten) aus.



AWS CLI

Zum Bearbeiten eines benutzerdefinierten Endpunkts mit der AWS CLI führen Sie den Befehl modify-db-cluster-endpoint aus.

Mit den folgenden Befehlen wird der Satz an DB-Instances geändert, die für einen benutzerdefinierten Endpunkt gelten, und optional zwischen dem Verhalten einer statischen Liste oder einer Ausschlussliste gewechselt. Die Parameter `--static-members` und `--excluded-members` weisen eine durch Leerzeichen getrennte Liste von DB-Instance-Kennungen auf.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 \
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --excluded-members db-instance-id-4 db-instance-id-5 \
  --region region_name
```

Für Windows:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 ^
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --excluded-members db-instance-id-4 db-instance-id-5 ^
  --region region_name
```

RDS-API

Zum Bearbeiten eines benutzerdefinierten Endpunkts mit der RDS-API führen Sie die Aktion ModifyDBClusterEndpoint.html aus.

Löschen eines benutzerdefinierten Endpunkts

Löschen Sie einen benutzerdefinierten Endpunkt mit der AWS-Managementkonsole, der AWS CLI oder der Amazon-RDS-API.

Konsole

Zum Löschen eines benutzerdefinierten Endpunkts mit der AWS-Managementkonsole wählen Sie auf der Cluster-Detailseite den entsprechenden benutzerdefinierten Endpunkt und dann die Aktion Delete (Löschen) aus.

AWS CLI

Zum Löschen eines benutzerdefinierten Endpunkts mit der AWS CLI führen Sie den Befehl delete-db-cluster-endpoint aus.

Mit folgendem Befehl wird ein benutzerdefinierter Endpunkt gelöscht: Sie müssen nicht den verknüpften Cluster, aber die Region angeben.

Für Linux, macOS oder Unix:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 \
  --region region_name
```

Für Windows:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 ^
  --region region_name
```

RDS-API

Zum Löschen eines benutzerdefinierten Endpunkts mit der RDS-API führen Sie die Aktion DeleteDBClusterEndpoint aus.

AWS CLI-Beispiele zu benutzerdefinierten Endpunkten für Amazon Aurora

Das folgende Tutorial verwendet AWS CLI-Beispiele mit der Unix-Shell-Syntax, um zu verdeutlichen, dass Sie einen Cluster mit mehreren kleinen DB-Instances und einigen großen DB-Instances definieren können. Außerdem wird gezeigt, wie Sie benutzerdefinierte Endpunkte erstellen können, um eine Verbindung mit jedem Satz von DB-Instances herzustellen. Zum Ausführen ähnlicher Befehle auf Ihrem System sollten Sie mit den Grundlagen der Arbeit mit Aurora-Clustern und dem Umgang mit AWS CLI vertraut sein, damit Sie Ihre eigenen Werte für Parameter wie Region, Subnetzgruppe und VPC-Sicherheitsgruppe verwenden können.

Dieses Beispiel verdeutlicht die ersten Einrichtungsschritte: das Erstellen eines Aurora-Clusters und das Hinzufügen von DB-Instances. Hierbei handelt es sich um einen heterogenen Cluster, weshalb nicht alle DB-Instances über die gleiche Kapazität verfügen. Die meisten Instances verwenden die AWS-Instance-Klasse `db.r4.4xlarge`. Die letzten beiden DB-Instances nutzen jedoch `db.r4.16xlarge`. Bei jedem dieser `create-db-instance`-Beispielbefehle wird die Ausgabe auf dem Bildschirm angezeigt und eine Kopie der JSON-Datei zur späteren Prüfung gespeichert.

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora-
mysql \
     --engine-version 8.0.mysql_aurora.3.04.0 --master-username $MASTER_USER --manage-
master-user-password \
     --db-subnet-group-name $SUBNET_GROUP  --vpc-security-group-ids $VPC_SECURITY_GROUP
 \
     --region $REGION
```

```
for i in 01 02 03 04 05 06 07 08
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.4xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done

for i in 09 10
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.16xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done
```

Die größeren Instances sind für spezialisierte Arten von Berichtsabfragen vorgesehen. Damit diese nicht auf die primäre Instance hochgestuft werden, wird im folgenden Beispiel gezeigt, wie der Hochstufungsebene für diese Instances die niedrigste Priorität zugewiesen wird. In diesem Beispiel wird die Option `--manage-master-user-password` zum Generieren des Hauptbenutzerpassworts und zum Verwalten dieses Passworts in Secrets Manager angegeben. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager. Alternativ können Sie die Option `--master-password` verwenden, um das Passwort selbst festzulegen und zu verwalten.

```
for i in 09 10
do
  aws rds modify-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
    --region $REGION --promotion-tier 15
done
```

Angenommen, Sie möchten die zwei „größeren“ Instances nur für die ressourcenintensivsten Abfragen verwenden. Dazu können Sie zunächst einen benutzerdefinierten schreibgeschützten Endpunkt erstellen. Anschließend können Sie eine statische Liste von Elementen hinzufügen, sodass der Endpunkt nur mit diesen DB-Instances verbunden ist. Da sich diese Instances bereits in der niedrigsten Hochstufungsebene befinden, ist es unwahrscheinlich, dass eine von ihnen auf die primäre Instance hochgestuft wird. Sollte eine der Instances auf die primäre Instance hochgestuft

worden sein, wäre sie über diesen Endpunkt nicht erreichbar, da der READER-Typ anstatt des ANY-Typs angegeben wurde.

Das folgende Beispiel verdeutlicht das Erstellen und Ändern von Endpunktbefehlen sowie eine JSON-Beispielausgabe mit dem ursprünglichen und geänderten Status des benutzerdefinierten Endpunkts.

```
$ aws rds create-db-cluster-endpoint --region $REGION \
    --db-cluster-identifier custom-endpoint-demo \
    --db-cluster-endpoint-identifier big-instances --endpoint-type reader
{
    "EndpointType": "CUSTOM",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterIdentifier": "custom-endpoint-demo",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "ExcludedMembers": [],
    "CustomEndpointType": "READER",
    "Status": "creating",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
```

```
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

Der standardmäßige READER-Endpunkt des Clusters kann eine Verbindung entweder zur kleinen oder zur großen DB-Instance herstellen. Hierdurch wird das Vorhersagen der Abfrageleistung und der Skalierbarkeit bei einer hohen Auslastung des Clusters erschwert. Um den Workload zwischen den DB-Instance-Sätzen klar aufzuteilen, können Sie den standardmäßigen READER-Endpunkt ignorieren und einen zweiten benutzerdefinierten Endpunkt erstellen, der eine Verbindung mit den anderen DB-Instances herstellt. Im folgenden Beispiel wird dies erreicht, indem zunächst ein benutzerdefinierter Endpunkt erstellt und dann eine Ausschlussliste hinzugefügt wird. Alle anderen DB-Instances, die Sie dem Cluster später hinzufügen, werden automatisch in diesen Endpunkt integriert. Der Typ ANY signalisiert, dass dieser Endpunkt mit insgesamt acht Instances verknüpft ist: mit der primären Instance und sieben anderen Aurora-Replicas. Wenn im Beispiel der Typ READER verwendet würde, wäre der benutzerdefinierte Endpunkt nur mit den sieben Aurora-Replicas verknüpft.

```
$ aws rds create-db-cluster-endpoint --region $REGION --db-cluster-identifier custom-
endpoint-demo \
    --db-cluster-endpoint-identifier small-instances --endpoint-type any
{
    "Status": "creating",
    "DBClusterEndpointIdentifier": "small-instances",
    "CustomEndpointType": "ANY",
    "EndpointType": "CUSTOM",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "ExcludedMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier small-instances \
    --excluded-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "DBClusterEndpointIdentifier": "small-instances",
```

```
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:c7tj4example:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [
        "custom-endpoint-demo-09",
        "custom-endpoint-demo-10"
    ],
    "StaticMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

Im folgenden Beispiel wird der Status der Endpunkte dieses Clusters geprüft. Mit dem `EndPointType` `WRITER` verfügt der Cluster immer noch über seinen ursprünglichen Cluster-Endpunkt, den Sie weiterhin für die Administration, für Extraktions-, Transformations- und Ladevorgänge (ETL) und andere Schreibvorgänge nutzen würden. Er verfügt immer noch über seinen ursprünglichen `READER`-Endpunkt, den Sie nicht verwenden würden, da jede zu ihm hergestellte Verbindung möglicherweise zu einer kleinen oder großen DB-Instance weitergeleitet wird. Die benutzerdefinierten Endpunkte machen dieses Verhalten vorhersehbar, da Verbindungen verwendet werden, die, je nach angegebenem Endpunkt, garantiert eine der kleinen oder großen DB-Instances verwenden.

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
```

```
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "ANY",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
            "ExcludedMembers": [
                "custom-endpoint-demo-09",
                "custom-endpoint-demo-10"
            ],
            "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "small-instances",
            "Status": "modifying"
        },
        {
            "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "READER",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
            "ExcludedMembers": [],
            "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [
                "custom-endpoint-demo-10",
                "custom-endpoint-demo-09"
            ],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "big-instances",
            "Status": "available"
        }
    ]
}
```

In den abschließenden Beispielen wird gezeigt, wie nachfolgende Datenbankverbindungen zu den benutzerdefinierten Endpunkten eine Verbindung mit den zahlreichen DB-Instances im

Aurora-Cluster herstellen. Der `small-instances`-Endpunkt stellt immer eine Verbindung zu den `db.r4.4xlarge`-DB-Instances her. Diese stellen die Hosts mit den niedrigeren Nummern in diesem Cluster dar.

```
$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-02 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-07 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-01 |
+--------------------------+
```

Der `big-instances`-Endpunkt stellt immer eine Verbindung zu den `db.r4.16xlarge`-DB-Instances her, die die beiden höchst nummerierten Hosts in diesem Cluster sind.

```
$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-10 |
+--------------------------+
```

```
$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-09 |
+--------------------------+
```

DB-Instance-Klassen von Amazon Aurora

Die DB-Instance-Klasse bestimmt die Berechnungs- und Speicherkapazität einer Amazon-Aurora-DB-Instance. Die benötigte DB-Instance-Klasse richtet sich nach Ihren Rechen- und Speicheranforderungen.

Eine DB-Instance-Klasse besteht sowohl aus dem DB-Instance-Klassentyp als auch aus der -größe. Beispielsweise ist db.r6g ein speicheroptimierter DB-Instance-Klassentyp, der von AWS-Graviton2-Prozessoren betrieben wird. Innerhalb des db.r6g-Instance-Klassentyps ist db.r6g.2xlarge eine DB-Instance-Klasse. Die Größe dieser Klasse ist 2xlarge.

Weitere Informationen zu Preisen der Instance-Klassen erhalten Sie unter Amazon-RDS-Preise.

Weitere Informationen zu DB-Instance-Klassentypen, unterstützten DB-Engines, unterstützten AWS-Regionen oder zu Hardwarespezifikationen für DB-Instance-Klassen finden Sie in den folgenden Abschnitten.

Themen

- DB-Instance-Klassenarten
- Unterstützte DB-Engines für DB-Instance-Klassen
- Bestimmung des Support von DB-Instance-Klassen in AWS-Regionen
- Hardware-Spezifikationen für DB-Instance-Klassen für Aurora

DB-Instance-Klassenarten

Amazon Aurora unterstützt DB-Instance-Klassen für die folgenden Anwendungsfälle:

- Aurora Serverless v2
- Speicheroptimiert
- Spitzenlastleistung
- Optimierte Lesevorgänge

Weitere Informationen zu EC2 Amazon-Instance-Typen finden Sie unter Instance-Typen in der EC2 Amazon-Dokumentation.

Typ der Instance-Klasse von Aurora Serverless v2

Der folgende Typ von Aurora Serverless v2 ist verfügbar:

- db.serverless – Ein spezieller DB-Instance-Klassentyp, der von Aurora Serverless v2 verwendet wird. Aurora passt Rechen-, Speicher- und Netzwerkressourcen dynamisch an, wenn sich die Workload ändert. Details zur Verwendung finden Sie unter Verwenden von Aurora Serverless v2.

Typen arbeitsspeicheroptimierter Instance-Klassen

Die speicheroptimierte X-Familie unterstützt die folgenden Instance-Klassen:

- db.x2g — Instance-Klassen, die für speicherintensive Anwendungen optimiert sind und auf Graviton2-Prozessoren basieren. AWS Diese Instance-Klassen bieten niedrige Kosten pro GiB Speicher.

  Sie können eine DB-Instance so ändern, dass sie eine der DB-Instance-Klassen verwendet, die von Graviton2-Prozessoren unterstützt werden. AWS Führen Sie dazu die gleichen Schritte wie bei jeder anderen Änderung der DB-Instance aus.

Die speicheroptimierte R-Familie unterstützt die folgenden Typen von Instance-Klassen:

- db.r8g – Instance-Klassen mit AWS -Graviton4-Prozessoren. Diese Instance-Klassen eignen sich ideal für die Ausführung speicherintensiver Workloads. Diese Instances bieten größere Instance-Größen mit bis zu dreimal mehr V und Arbeitsspeicher als die Graviton3-basierten db.r7g-Instances der siebten Generation. CPUs AWS Sie werden vom AWS Nitro System betrieben, einer Kombination aus dedizierter Hardware und leichtem Hypervisor.
- Sie können eine DB-Instance so modifizieren, dass sie eine der DB-Instance-Klassen verwendet, die von AWS Graviton4-Prozessoren unterstützt werden. Führen Sie dazu die gleichen Schritte wie bei jeder anderen Änderung der DB-Instance aus.
- db.r7g — Instance-Klassen, die auf Graviton3-Prozessoren basieren. AWS Diese Instance-Klassen eignen sich ideal für die Ausführung speicherintensiver Workloads.

  Sie können eine DB-Instance so ändern, dass sie eine der DB-Instance-Klassen verwendet, die von Graviton3-Prozessoren unterstützt werden. AWS Führen Sie dazu die gleichen Schritte wie bei jeder anderen Änderung der DB-Instance aus. Sie werden vom AWS Nitro System betrieben, einer Kombination aus dedizierter Hardware und leichtem Hypervisor.

- db.r7i – Instance-Klassen mit skalierbaren Intel-Xeon-Prozessoren der 4. Generation
  Diese Instance-Klassen sind SAP-zertifiziert und eignen sich optimal für die Ausführung speicherintensiver Workloads. Sie können eine DB-Instance so ändern, dass sie eine der DB-Instance-Klassen mit skalierbaren Intel-Xeon-Prozessoren der 4. Generation verwendet Führen Sie dazu die gleichen Schritte wie bei jeder anderen Änderung der DB-Instance aus. Sie werden vom AWS Nitro System betrieben, einer Kombination aus dedizierter Hardware und leichtem Hypervisor.
- db.r6g – Instance-Klassen mit AWS -Graviton2-Prozessoren. Diese Instance-Klassen eignen sich ideal für die Ausführung speicherintensiver Workloads mit hoher Geschwindigkeit und geringer Latenz benötigen. Sie werden vom AWS Nitro System betrieben, einer Kombination aus dedizierter Hardware und leichtem Hypervisor.
- Sie können eine DB-Instance so modifizieren, dass sie eine der DB-Instance-Klassen verwendet, die von AWS Graviton2-Prozessoren unterstützt werden. Führen Sie dazu die gleichen Schritte wie bei jeder anderen Änderung der DB-Instance aus.
- db.r6i – Instance-Klassen mit Unterstützung der skalierbaren Intel-Xeon-Prozessoren der 3. Generation. Diese Instance-Klassen sind SAP-zertifiziert und eignen sich ideal für die Ausführung speicherintensiver Workloads.
- db.r5 – Optimierte Instance-Klassen für speicherintensive Anwendungen. Diese Instance-Klassen bieten eine verbesserte Netzwerk- und Amazon Elastic Block Store (Amazon EBS)-Leistung.
  Sie werden vom AWS Nitro System betrieben, einer Kombination aus dedizierter Hardware und leichtem Hypervisor.
- db.r4 – Diese Instance-Klassen werden nur für Aurora MySQL 2.x und die Aurora-PostgreSQL-Versionen 11 und 12 unterstützt. Für alle Aurora-DB-Cluster, die DB-Instance-Klassen vom Typ „db.r4" verwenden, wird empfohlen, schnellstmöglich auf eine Instance-Klasse einer höheren Generation zu aktualisieren.

  Die db.r4-Instance-Klassen sind für die Cluster-Speicherkonfiguration Aurora I/O-Optimized nicht verfügbar.

Instance-Klassen mit Spitzenlastleistung

Die folgenden DB-Instance-Klassentypen mit Spitzenlastleistung sind verfügbar:

- db.t4g — Allzweck-Instanzklassen, die auf ARM-basierten Graviton2-Prozessoren basieren. AWS Diese Instance-Klassen bieten ein besseres Preis-Leistungs-Verhältnis als die DB-Instance-Klassen mit Spitzenlastleistung der vorherigen Generation für eine breite Palette von Allzweck-

Workloads mit Spitzenleistung. Amazon RDS db.t4g-Instances sind für den unbegrenzten Modus konfiguriert. Das bedeutet, dass sie gegen eine zusätzliche Gebühr während eines 24-Stunden-Zeitfensters Burst-Leistung über die Baseline hinaus bieten können.

Sie können eine DB-Instance so ändern, dass sie eine der DB-Instance-Klassen verwendet, die von Graviton2-Prozessoren unterstützt werden. AWS Führen Sie dazu die gleichen Schritte wie bei jeder anderen Änderung der DB-Instance aus.

- db.t3: Instance-Klassen, die ein Basisleistungsniveau bieten, mit der Möglichkeit, die volle CPU-Auslastung zu erreichen. Die db.t3-T3-Instances sind für den unbegrenzten Modus konfiguriert. Diese Instance-Klassen bieten mehr Rechenkapazität als die vorherigen db.t2-Instance-Klassen. Sie werden vom AWS -Nitro System angetrieben, einer Kombination aus dedizierter Hardware und leichtem Hypervisor. Wir empfehlen, diese Instance-Klassen ausschließlich für das Entwickeln und Testen von Servern oder Nicht-Produktionsservern zu verwenden.
- db.t2: Instance-Klassen, die ein Basisleistungsniveau bieten, mit der Möglichkeit, die volle CPU-Auslastung zu erreichen. Die db.t2-Instances sind für den unbegrenzten Modus konfiguriert. Wir empfehlen, diese Instance-Klassen ausschließlich für das Entwickeln und Testen von Servern oder Nicht-Produktionsservern zu verwenden.

  Die db.t2-Instance-Klassen sind für die Cluster-Speicherkonfiguration Aurora I/O-Optimized nicht verfügbar.

> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Empfehlungen für die T-Instance-Typen finden Sie unter Verwendung von T-Instance-Klassen für Entwicklung und Tests.

Informationen zu den Hardware-Spezifikationen für DB-Instance-Klassen finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

Instance-Klassentypen für optimierte Lesevorgänge

Folgende Instance-Klassentypen für optimierte Lesevorgänge sind verfügbar:

- db.r8gd — Instanzklassen, die von Graviton4-Prozessoren unterstützt werden. Diese Instance-Klassen eignen sich ideal für die Ausführung speicherintensiver Workloads und bieten lokalen SSD-Speicher auf Blockebene für Anwendungen, die lokalen NVMe Speicher mit hoher

Geschwindigkeit und niedriger Latenz benötigen. Sie bieten einen maximalen Speicher von 1,5 TiB und bis zu 11,4 TB direkt angeschlossenen SSD-Speicher NVMe.

- db.r6gd — Instanzklassen, die von Graviton2-Prozessoren unterstützt werden. AWS Diese Instance-Klassen eignen sich ideal für die Ausführung speicherintensiver Workloads und bieten lokalen SSD-Speicher auf Blockebene für Anwendungen, die lokalen NVMe Speicher mit hoher Geschwindigkeit und niedriger Latenz benötigen.
- db.r6id – Instance-Klassen mit skalierbaren Intel-Xeon-Prozessoren der 3. Generation. Diese Instance-Klassen sind SAP-zertifiziert und eignen sich ideal für speicherintensive Workloads. Sie bieten einen maximalen Speicher von 1 TiB und bis zu 7,6 TB direkt angeschlossenen SSD-Speicher NVMe.

Unterstützte DB-Engines für DB-Instance-Klassen

Die folgenden Tabellen zeigen die unterstützten DB-Instance-Klassen für DB-Engines von Amazon Aurora.

db.serverless – Aurora Serverless v2-Instance-Klasse mit automatischer Kapazitätsskalierung

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.serverless	Siehe Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2.	Siehe Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2.

db.x2g — speicheroptimierte Instanzklassen, die auf Graviton2-Prozessoren basieren AWS

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.x2g.16xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.x2g.12xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher,

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
		13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.x2g.8xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.x2g.4xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.x2g.2xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.x2g.xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.x2g.large	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher

db.r6gd — Optimierte Lese-Instanzklassen, die auf Graviton2-Prozessoren basieren AWS

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r6gd.16xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r6gd.12xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher
db.r6gd.8xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher
db.r6gd.4xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher
db.r6gd.2xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher
db.r6gd.xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher

db.r6id – Instance-Klassen für optimierte Lesevorgänge

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r6id.32xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher
db.r6id.24xlarge	Nein	17.4 und höher, 16.1 und höher, 15.4 und höher, 14.9 und höher

db.r8g — speicheroptimierte Instanzklassen, die auf Graviton4-Prozessoren basieren AWS

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r8g.48xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r8g.24xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r8g.16xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r8g.12xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r8g.8xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r8g.4xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r8g.2xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r8g.xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r8g.large	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher

db.r8gd — optimierte Lese-Instanzklassen, die auf Graviton4-Prozessoren basieren AWS

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r8gd.48xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher
db.r8gd.24xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher
db.r8gd.16xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher
db.r8gd.12xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher
db.r8gd.8xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher
db.r8gd.4xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher
db.r8gd.2xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher
db.r8g.xlarge	Nein	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher

db.r7g — speicheroptimierte Instanzklassen, die auf Graviton3-Prozessoren basieren AWS

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r7g.16xlarge	2.12.0 und höher, 3.03.1 und höher	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher, 13.10 und höher

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r7g.12xlarge	2.12.0 und höher, 3.03.1 und höher	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher, 13.10 und höher
db.r7g.8xlarge	2.12.0 und höher, 3.03.1 und höher	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher, 13.10 und höher
db.r7g.4xlarge	2.12.0 und höher, 3.03.1 und höher	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher, 13.10 und höher
db.r7g.2xlarge	2.12.0 und höher, 3.03.1 und höher	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher, 13.10 und höher
db.r7g.xlarge	2.12.0 und höher, 3.03.1 und höher	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher, 13.10 und höher
db.r7g.large	2.12.0 und höher, 3.03.1 und höher	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.7 und höher, 13.10 und höher

db.r7i – arbeitsspeicheroptimierte Instance-Klassen

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r7i.48xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r7i.24xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r7i.16xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r7i.12xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r7i.8xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r7i.4xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r7i.2xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r7i.xlarge	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher
db.r7i.large	3.08.0 und höher	17.4 und höher, 16.3 und höher, 15.7 und höher, 14.12 und höher, 13.15 und höher

db.r6g — speicheroptimierte Instanzklassen, die auf Graviton2-Prozessoren basieren AWS

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r6g.16xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher,

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
		13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.r6g.12xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.r6g.8xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.r6g.4xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.r6g.2xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.r6g.xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher
db.r6g.large	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.8 und höher, 11.9 und 11.12 und höher

db.r6i – arbeitsspeicheroptimierte Instance-Klassen

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r6i.32xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.24xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.16xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.12xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.8xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.4xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.2xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.xlarge	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher
db.r6i.large	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.5 und höher und 12.9 und höher

db.r5 – arbeitsspeicheroptimierte Instance-Klassen

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r5.24xlarge	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen
db.r5.16xlarge	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen
db.r5.12xlarge	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen
db.r5.8xlarge	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen
db.r5.4xlarge	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen
db.r5.2xlarge	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen
db.r5.xlarge	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen
db.r5.large	Alle aktuell verfügbaren Versionen	Alle aktuell verfügbaren Versionen

db.r4 – arbeitsspeicheroptimierte Instance-Klassen

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r4.16xlarge	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Alle unterstützten 11- und 12-Versionen
db.r4.8xlarge	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Alle unterstützten 11- und 12-Versionen
db.r4.4xlarge	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Alle unterstützten 11- und 12-Versionen
db.r4.2xlarge	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Alle unterstützten 11- und 12-Versionen
db.r4.xlarge	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Alle unterstützten 11- und 12-Versionen

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.r4.large	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Alle unterstützten 11- und 12-Versionen

db.t4g — Instance-Klassen mit hervorragender Leistung, die auf Graviton2-Prozessoren basieren AWS

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.t4g.2xlarge	Nein	Nein
db.t4g.xlarge	Nein	Nein
db.t4g.large	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.7 und höher und 11.12 und höher
db.t4g.medium	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.7 und höher und 11.12 und höher
db.t4g.klein	Nein	Nein

db.t3 – Instance-Klassen mit Spitzenlastleistung

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.t3.2xlarge	Nein	Nein
db.t3.xlarge	Nein	Nein
db.t3.large	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher,

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
		13.3 und höher, 12.7 und höher und 11.12 und höher
db.t3.medium	Alle aktuell verfügbaren Versionen	17.4 und höher, 16.1 und höher, 15.2 und höher, 14.3 und höher, 13.3 und höher, 12.7 und höher und 11.12 und höher
db.t3.small	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Nein
db.t3.micro	Nein	Nein

db.t2 – Instance-Klassen mit Spitzenlastleistung

Instance-Klasse	Aurora MySQL	Aurora PostgreSQL
db.t2.medium	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Nein
db.t2.small	Alle 2.x-Versionen; nicht unterstützt in 3.x-Versionen	Nein

Bestimmung des Support von DB-Instance-Klassen in AWS-Regionen

Zur Bestimmung der DB-Instance-Klassen, die von jeder DB-Engine in einer bestimmten AWS-Region unterstützt werden, stehen mehrere Ansätze zur Verfügung. Sie können die AWS-Managementkonsole, die Seite Amazon RDS Pricing (Amazon-RDS-Preise) oder den AWS CLI-Befehl describe-orderable-db-instance-options verwenden.

> **Note**
>
> Wenn Sie Vorgänge mit AWS-Managementkonsole ausführen, werden automatisch die unterstützten DB-Instance-Klassen für eine bestimmte DB-Engine, DB-Engine-Version und

AWS-Region angezeigt. Beispiele für Vorgänge, die Sie ausführen können, sind das Erstellen und Ändern einer DB-Instance.

Inhalt

- Verwenden der Amazon-RDS-Preisseite zur Angabe des Supports von DB-Instance-Klassen in AWS-Regionen
- Verwenden von AWS CLI zur Bestimmung des Supports der DB-Instance-Klassen in AWS-Regionen
  - Auflistung der DB-Instance-Klassen, die von einer bestimmten DB-Engine-Version in einer AWS-Region unterstützt werden
  - Auflisten der DB-Engine-Versionen, die eine bestimmte DB-Instance-Klasse in einer AWS-Region unterstützen

Verwenden der Amazon-RDS-Preisseite zur Angabe des Supports von DB-Instance-Klassen in AWS-Regionen

Sie können die Seite Amazon Aurora Pricing verwenden, um die DB-Instance-Klassen zu bestimmen, die von jeder DB-Engine in einer bestimmten AWS-Region unterstützt werden.

So verwenden Sie die Preisseite, um die DB-Instance-Klassen zu bestimmen, die von jeder Engine in einer Region unterstützt werden

1. Gehen Sie zu Amazon Aurora Pricing.
2. Wählen Sie im Bereich AWS Pricing Calculator eine Amazon Aurora-Engine aus.
3. Wählen Sie unter Region auswählen eine AWS-Region aus.
4. Wählen Sie unter Cluster-Konfigurationsoption eine Konfigurationsoption aus.
5. Verwenden Sie den Abschnitt für kompatible Instances, um sich die unterstützten DB-Instance-Klassen anzusehen.
6. (Optional) Wählen Sie andere Optionen im Rechner und dann Zusammenfassung speichern und anzeigen oder Service speichern und hinzufügen aus.

Verwenden von AWS CLI zur Bestimmung des Supports der DB-Instance-Klassen in AWS-Regionen

Sie können AWS CLI verwenden, um zu bestimmen, welche DB-Instance-Klassen für bestimmte DB-Engines und DB-Engine-Versionen in einer AWS-Region unterstützt werden.

Wenn Sie die AWS CLI-Beispiele in diesem Abschnitt verwenden möchten, geben Sie gültige Werte für die DB-Engine, die DB-Engine-Version, die DB-Instance-Klasse und die AWS-Region ein. Die folgende Tabelle zeigt die gültigen DB-Engine-Werte.

Engine-Name	Engine-Wert in CLI-Befehlen	Weitere Informationen zu den Versionen
MySQL 5.7-kompatibles Aurora und 8.0-compatible Aurora	`aurora-mysql`	Aktualisierungen der Datenbank-Engine für Amazon Aurora MySQL Version 2 und Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 3 in den Versionshinweisen für Aurora MySQL
Aurora PostgreSQL	`aurora-postgresql`	Versionshinweise für Aurora PostgreSQL

Weitere Informationen zu AWS-Region-Namen finden Sie unter AWS Regionen.

Die folgenden Beispiele zeigen, wie Sie den Support von DB-Instance-Klassen in einer AWS-Region mithilfe des AWS CLI-Befehls describe-orderable-db-instance-options ermitteln.

Themen

- Auflistung der DB-Instance-Klassen, die von einer bestimmten DB-Engine-Version in einer AWS-Region unterstützt werden
- Auflisten der DB-Engine-Versionen, die eine bestimmte DB-Instance-Klasse in einer AWS-Region unterstützen

Auflistung der DB-Instance-Klassen, die von einer bestimmten DB-Engine-Version in einer AWS-Region unterstützt werden

Um die DB-Instance-Klassen aufzulisten, die von einer bestimmten DB-Engine-Version in einer AWS-Region unterstützt werden, führen Sie den folgenden Befehl aus.

Für Linux, macOS oder Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

Für Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

Die Ausgabe zeigt auch die Engine-Modi, die für jede DB-Instance-Klasse unterstützt werden.

Der folgende Befehl listet beispielsweise die unterstützten DB-Instance-Klassen für Version 13.6 der Aurora-PostgreSQL-DB-Engine in USA Ost (Nord-Virginia) auf.

Für Linux, macOS oder Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

Für Windows:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 ^
```

```
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}"  ^
    --output table ^
    --region us-east-1
```

Auflisten der DB-Engine-Versionen, die eine bestimmte DB-Instance-Klasse in einer AWS-Region unterstützen

Um die DB-Engine-Versionen aufzulisten, die eine bestimmte DB-Instance-Klasse in einer AWS-Region unterstützen, führen Sie den folgenden Befehl aus.

Für Linux, macOS oder Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

Für Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

Die Ausgabe zeigt auch die Engine-Modi, die für jede DB-Engine-Version unterstützt werden.

Der folgende Befehl listet beispielsweise die DB-Engine-Versionen der RDS für Aurora PostgreSQL-DB-Engine auf, welche die db.r5.large DB-Instance-Klasse in US East (N. Virginia) unterstützen.

Für Linux, macOS oder Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

Für Windows:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region us-east-1
```

Hardware-Spezifikationen für DB-Instance-Klassen für Aurora

In der folgenden Tabelle finden Sie Hardware-Details zu den DB-Instance-Klassen von Amazon RDS für Aurora.

Informationen zur Aurora-DB-Engine-Unterstützung für die einzelnen DB-Instance-Klassen finden Sie unter Unterstützte DB-Engines für DB-Instance-Klassen.

Themen

- Hardware-Terminologie für DB-Instance-Klassen für Aurora
- Hardwarespezifikationen für arbeitsspeicheroptimierte Instance-Klassen
- Hardwarespezifikationen für Instance-Klassen mit Spitzenlastleistung

Hardware-Terminologie für DB-Instance-Klassen für Aurora

Die folgende Terminologie wird zum Beschreiben der Hardwarespezifikationen für DB-Instance-Klassen verwendet:

vCPU

Die Anzahl der virtuellen Zentraleinheiten (CPUs). Eine virtuelle CPU ist eine Kapazitätseinheit, mit der Sie DB-Instance-Klassen vergleichen können. Anstatt einen bestimmten Prozessor für mehrere Monate oder Jahre zu erwerben oder zu leasen, wird jetzt Kapazität stundenweise gemietet. Unser Ziel ist es, eine konsistente und spezifische Menge an CPU-Kapazität innerhalb der Grenzen der zugrunde liegenden Hardware zur Verfügung zu stellen.

EC2-Recheneinheiten

Das relative Maß für die ganzzahlige Verarbeitungsleistung einer EC2 Amazon-Instance. Um Entwicklern den Vergleich der CPU-Kapazität zwischen verschiedenen Instance-Klassen zu

erleichtern, haben wir eine Amazon EC2 Compute Unit definiert. Die Menge an CPU, die einer bestimmten Instance zugewiesen ist, wird in diesen EC2 Recheneinheiten ausgedrückt. Ein Steuergerät bietet derzeit eine CPU-Kapazität, die einem Opteron- oder GHz 2007 Xeon-Prozessor von 1,0 bis 1,2 entspricht.

Arbeitsspeicher (GiB)

Der Arbeitsspeicher (RAM) in Gibibytes, der der DB-Instance zugeteilt ist. Häufig ist das Verhältnis zwischen Arbeitsspeicher- und vCPU konsistent. Beispielsweise hat die Instance-Klasse db.r4 das gleiche Verhältnis von Speicher zu vCPU wie die Instance-Klasse db.r5. Für die meisten Anwendungsfälle bietet die Instance-Klasse db.r5 jedoch eine bessere und konsistentere Performance als die Instance-Klasse db.r4.

Max. EBS-Bandbreite (Mbit/s)

Die maximale EBS-Bandbreite in Megabit pro Sekunde. Dividieren Sie durch 8, um den erwarteten Durchsatz in Megabyte pro Sekunde zu erhalten.

> **Note**
>
> Diese Zahl bezieht sich auf die I/O Bandbreite für den lokalen Speicher innerhalb der DB-Instance. Sie gilt nicht für die Kommunikation mit dem Aurora-Clustervolume.

Netzwerkbandbreite

Die Netzwerkgeschwindigkeit relativ zu anderen DB-Instance-Klassen.

Informationen zur Verwendung von CloudWatch Amazon-Metriken zur Überwachung Ihres Aurora-DB-Instance-Durchsatzes finden Sie unter Bewerten der DB-Instance-Nutzung für Aurora MySQL mit Amazon-CloudWatch-Metriken und Bewertung der DB-Instance-Nutzung für Aurora PostgreSQL anhand von Metriken CloudWatch.

Hardwarespezifikationen für arbeitsspeicheroptimierte Instance-Klassen

Die folgenden Tabellen zeigen die Rechen-, Arbeitsspeicher-, Speicher- und Bandbreitenspezifikationen für arbeitsspeicheroptimierte Instance-Klassen.

db.x2g — speicheroptimierte Instance-Klassen mit Graviton2-Prozessoren AWS

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.x2g.16xlarge	64	—	1024	Nur EBS-optimiert	19.000	25
db.x2g.12xlarge	48	—	768	Nur EBS-optimiert	14.250	20
db.x2g.8xlarge	32	—	512	Nur EBS-optimiert	9.500	12
db.x2g.4xlarge	16	—	256	Nur EBS-optimiert	4.750	Bis zu 10
db.x2g.2xlarge	8	—	128	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10
db.x2g.xlarge	4	—	64	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10
db.x2g.large	2	—	32	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10

db.r8gd — speicheroptimierte Instanzklassen, die auf Graviton4-Prozessoren und SSD-Speicher basieren AWS

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r8gd.48xlarge	192	—	1536	6 x 1900 SSD NVMe	40 000	50

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r8gd.24x groß	96	—	768	3 x 1900 SSD NVMe	30 000	40
db.r8gd.16x groß	64	—	512	2 x 1900 SSD NVMe	20 000	30
db.r8gd.12x groß	48	—	384	3 x 950 SSD NVMe	15 000	22,5
db.r8gd.8xgroß	32	—	256	1 x 1900 SSD NVMe	10.000	15
db.r8gd.4 x groß	16	—	128	1 x 950 SSD NVMe	Bis zu 10 000*	Bis zu 15
db.r8gd.2 x groß	8	—	64	1 x 474 SSD NVMe	Bis zu 10 000*	Bis zu 15
db.r8gd.xlarge	4	—	32	1 x 237 SSD NVMe	Bis zu 10 000*	Bis zu 12,5
db.r8gd.groß	2	—	16	1 x 118 SSD NVMe	Bis zu 10 000*	Bis zu 12,5

db.r8g — speicheroptimierte Instanzklassen, die auf Graviton4-Prozessoren basieren AWS

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r8g.48xlarge	192	—	1536	Nur EBS-optimiert	40 000	50

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r8g.24xlarge	96	—	768	Nur EBS-optimiert	30 000	40
db.r8g.16xlarge	64	—	512	Nur EBS-optimiert	20 000	30
db.r8g.12xlarge	48	—	384	Nur EBS-optimiert	15 000	22.5
db.r8g.8xlarge	32	—	256	Nur EBS-optimiert	10.000	15
db.r8g.4xlarge	16	—	128	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 15
db.r8g.2xlarge	8	—	64	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 15
db.r8g.xlarge	4	—	32	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r8g.large	2	—	16	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5

db.r7i – arbeitsspeicheroptimierte Instance-Klassen, die mit skalierbaren Intel-Xeon-Prozessoren der 4. Generation betrieben werden

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r7i.48xlarge	192	—	1536	Nur EBS-optimiert	40 000	50
db.r7i.24xlarge	96	—	768	Nur EBS-optimiert	30 000	37,5
db.r7i.16xlarge	64	—	512	Nur EBS-optimiert	20 000	25
db.r7i.12xlarge	48	—	384	Nur EBS-optimiert	15 000	18,75
db.r7i.8xlarge	32	—	256	Nur EBS-optimiert	10.000	12,5
db.r7i.4xlarge	16	—	128	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r7i.2xlarge	8	—	64	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r7i.xlarge	4	—	32	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r7i.large	2	—	16	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5

db.r7g — speicheroptimierte Instanzklassen mit Graviton3-Prozessoren AWS

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r7g.16xlarge	64	—	512	Nur EBS-optimiert	20 000	30
db.r7g.12xlarge	48	—	384	Nur EBS-optimiert	15 000	22.5
db.r7g.8xlarge	32	—	256	Nur EBS-optimiert	10.000	15
db.r7g.4xlarge	16	—	128	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 15
db.r7g.2xlarge	8	—	64	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 15
db.r7g.xlarge	4	—	32	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r7g.large	2	—	16	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5

db.r6id – arbeitsspeicheroptimierte Instance-Klassen mit skalierbaren Intel-Xeon-Prozessoren der 3. Generation und SSD-Speicher

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r6id.32xlarge	128	—	1,024	4 x 1900 SSD NVMe	40 000	50

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r6id.24xlarge	96	—	768	4 x 1425 SSD NVMe	30 000	37,5

db.r6gd — speicheroptimierte Instanzklassen mit Graviton2-Prozessoren und SSD-Speicher AWS

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r6gd.16xlarge	64	—	512	2 NVMe x 1900 SSD	19.000	25
db.r6gd.12xlarge	48	—	384	2 x 1425 SSD NVMe	13.500	20
db.r6gd.8xlarge	32	—	256	1 x 1900 SSD NVMe	9 000	12
db.r6gd.4xlarge	16	—	128	1 x 950 NVMe SSD	4.750	Bis zu 10
db.r6gd.2xlarge	8	—	64	1 x 474 NVMe SSD	Bis zu 4750.	Bis zu 10
db.r6gd.xlarge	4	—	32	1 x 237 NVMe SSD	Bis zu 4750.	Bis zu 10

db.r6g — speicheroptimierte Instanzklassen mit Graviton2-Prozessoren AWS

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r6g.16xlarge	64	—	512	Nur EBS-optimiert	19.000	25
db.r6g.12xlarge	48	—	384	Nur EBS-optimiert	13.500	20
db.r6g.8xlarge	32	—	256	Nur EBS-optimiert	9 000	12
db.r6g.4xlarge	16	—	128	Nur EBS-optimiert	4.750	Bis zu 10
db.r6g.2xlarge	8	—	64	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10
db.r6g.xlarge	4	—	32	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10
db.r6g.large	2	—	16	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10

db.r6i – arbeitsspeicheroptimierte Instance-Klassen mit skalierbaren Intel-Xeon-Prozessoren der 3. Generation

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r6i.32xlarge	128	—	1,024	Nur EBS-optimiert	40 000	50

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r6i.24xlarge	96	—	768	Nur EBS-optimiert	30 000	37,5
db.r6i.16xlarge	64	—	512	Nur EBS-optimiert	20 000	25
db.r6i.12xlarge	48	—	384	Nur EBS-optimiert	15 000	18.75
db.r6i.8xlarge	32	—	256	Nur EBS-optimiert	10.000	12,5
db.r6i.4xlarge	16	—	128	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r6i.2xlarge	8	—	64	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r6i.xlarge	4	—	32	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5
db.r6i.large	2	—	16	Nur EBS-optimiert	Bis zu 10 000*	Bis zu 12,5

db.r5 – arbeitsspeicheroptimierte Instance-Klassen

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r5.24xlarge	96	347	768	Nur EBS-optimiert	19.000	25

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r5.16xlarge	64	264	512	Nur EBS-optimiert	13.600	20
db.r5.12xlarge	48	173	384	Nur EBS-optimiert	9.500	12
db.r5.8xlarge	32	132	256	Nur EBS-optimiert	6.800	10
db.r5.4xlarge	16	71	128	Nur EBS-optimiert	4.750	Bis zu 10
db.r5.2xlarge	8	38	64	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10
db.r5.xlarge	4	19	32	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10
db.r5.large	2	10	16	Nur EBS-optimiert	Bis zu 4750.	Bis zu 10

db.r4 – arbeitsspeicheroptimierte Instance-Klassen mit skalierbaren Intel-Xeon-Prozessoren

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r4.16xlarge	64	195	488	Nur EBS-optimiert	14.000	25
db.r4.8xlarge	32	99	244	Nur EBS-optimiert	7.000	10

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.r4.4xlarge	16	53	122	Nur EBS-optimiert	3.500	Bis zu 10
db.r4.2xlarge	8	27	61	Nur EBS-optimiert	1.700	Bis zu 10
db.r4.xlarge	4	13.5	30.5	Nur EBS-optimiert	850	Bis zu 10
db.r4.large	2	7	15.25	Nur EBS-optimiert	425	Bis zu 10

Hardwarespezifikationen für Instance-Klassen mit Spitzenlastleistung

Die folgenden Tabellen zeigen die Rechen-, Arbeitsspeicher-, Speicher- und Bandbreitenspezifikationen für Instance-Klassen mit Spitzenlastleistung.

db.t4g — Instance-Klassen mit hervorragender Leistung, die auf Graviton2-Prozessoren basieren AWS

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.t4g.large	2	—	8	Nur EBS-optimiert	Bis zu 2.780	Bis zu 5
db.t4g.medium	2	—	4	Nur EBS-optimiert	Bis zu 2 085	Bis zu 5

db.t3 – Instance-Klassen mit Spitzenlastleistung

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.t3.large	2	Variab	8	Nur EBS-optimiert	Bis zu 2.048	Bis zu 5
db.t3.medium	2	Variab	4	Nur EBS-optimiert	Bis zu 1.536	Bis zu 5
db.t3.small	2	Variab	2	Nur EBS-optimiert	Bis zu 1.536	Bis zu 5

db.t2 – Instance-Klassen mit Spitzenlastleistung

Instance-Klasse	vCPU	EC2-Recheneinheiten	Arbeitsspeicher (GiB)	Instance-Speicher (GB)	Max. EBS-Bandbreite (Mbit/s)	Netzwerkbandbreite (Gbit/s)
db.t2.medium	2	Variab	4	Nur EBS	—	Mittel
db.t2.small	1	Variab	2	Nur EBS	—	Niedrig

Amazon-Aurora-Speicher

Im Folgenden können Sie mehr über das Aurora-Speichersubsystem erfahren. Aurora verwendet eine verteilte und gemeinsam genutzte Speicherarchitektur, die ein wichtiger Faktor für Leistung, Skalierbarkeit und Zuverlässigkeit für Aurora-Cluster ist.

Themen

- Übersicht über Amazon-Aurora-Speicher
- Inhalt des Cluster-Volumes
- Speicherkonfigurationen für DB-Cluster von Amazon Aurora

- Wie sich die Größe des Aurora-Speichers automatisch ändert
- Informationen zur Abrechnung des Aurora-Datenspeichers

Übersicht über Amazon-Aurora-Speicher

Aurora-Daten werden im Cluster-Volume gespeichert. Hierbei handelt es sich um ein einzelnes, virtuelles Volume, das SSD-Laufwerke verwendet. Ein Cluster-Volume besteht aus Datenkopien, die sich zwischen drei Availability Zones in einer einzelnen AWS-Region befinden. Da Daten automatisch zwischen Availability Zones repliziert werden, bleiben Ihre Daten sehr lange beständig, mit einer geringeren Wahrscheinlichkeit für Datenverlust. Durch diese Replikation wird eine höhere Verfügbarkeit Ihrer Datenbank bei einem Failover sichergestellt. Dies ist möglich, weil die Datenkopien bereits in den anderen Availability Zones vorhanden sind und Datenanfragen für die DB-Instances in Ihrem DB-Cluster weiterhin bearbeitet werden. Die Menge der replizierten Daten ist unabhängig von der Anzahl der DB-Instances in Ihrem Cluster.

Aurora verwendet separaten lokalen Speicher für nicht persistente, temporäre Dateien. Dazu gehören Dateien, die für Zwecke wie das Sortieren großer Datensätze während der Abfrageverarbeitung oder für die Indexerstellung verwendet werden. Weitere Informationen erhalten Sie unter Temporäre Speicherlimits für Aurora MySQL und Temporäre Speicherlimits für Aurora PostgreSQL.

Inhalt des Cluster-Volumes

Das Aurora-Cluster-Volume enthält all Ihre Benutzerdaten, Schema-Objekte sowie interne Metadaten wie die Systemtabellen und das Binärprotokoll. Beispielsweise speichert Aurora alle Tabellen, Indizes, Binary Large Objects (BLOBs) oder gespeicherte Prozeduren eines Aurora-Clusters im Cluster-Volume.

Die verschiedene Speicherquellen nutzende Architektur von Aurora sorgt dafür, dass Ihre Daten von den DB-Instances im Cluster unabhängig sind. So können Sie eine DB-Instance schnell hinzufügen, weil Aurora keine neue Kopie der Tabellendaten erstellt. Stattdessen stellt die DB-Instance eine Verbindung zum freigegebenen Volume her, das bereits alle Daten von Ihnen enthält. Sie können eine DB-Instance aus einem Cluster entfernen, ohne die zugrunde liegenden Daten aus dem Cluster entfernen zu müssen. Aurora entfernt die Daten nur, wenn Sie den gesamten Cluster löschen.

Speicherkonfigurationen für DB-Cluster von Amazon Aurora

Amazon Aurora bietet zwei Speicherkonfigurationen für DB-Cluster:

- Aurora I/O-Optimized – Verbessertes Preis-Leistungs-Verhältnis und bessere Vorhersagbarkeit für I/O-intensive Anwendungen. Sie zahlen nur für die Nutzung und Speicherung Ihrer DB-Cluster, ohne zusätzliche Gebühren für Lese- und Schreib-I/O-Operationen.

  Aurora I/O-Optimized ist am besten geeignet, wenn Ihre I/O-Ausgaben 25 % oder mehr Ihrer gesamten Aurora-Datenbankausgaben ausmachen.

  Sie können Aurora I/O-Optimized auswählen, wenn Sie einen DB-Cluster mit einer DB-Engine-Version erstellen oder ändern, die die Cluster-Konfiguration von Aurora I/O-Optimized unterstützt. Sie können jederzeit von Aurora I/O-Optimized zu Aurora Standard wechseln.
- Aurora Standard – Kostengünstige Preise für viele Anwendungen mit moderater I/O-Auslastung. Neben der Auslastung und Speicherung Ihrer DB-Cluster zahlen Sie auch einen Standardsatz pro 1 Million Anfragen für I/O-Operationen.

  Aurora Standard ist am besten geeignet, wenn Ihre I/O-Ausgaben 25 % oder weniger Ihrer gesamten Aurora-Datenbankausgaben ausmachen.

  Sie können alle 30 Tage von Aurora Standard auf Aurora I/O-Optimized wechseln. Wenn Sie bei nicht NVMe-basierten DB-Instances zwischen Aurora Standard- und Aurora I/O-Optimized-Speicheroptionen wechseln, entsteht keine Ausfallzeit. Bei NVMe-basierten DB-Instances erfordert der Wechsel zwischen Aurora I/O-Optimized- und Aurora Standard-Speicheroptionen jedoch einen Neustart der Datenbank-Engine, was zu einer kurzen Ausfallzeit führen kann.

Weitere Informationen zu AWS-Region und zur Unterstützung der Versionen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Cluster-Speicherkonfigurationen.

Weitere Informationen zu den Preisen für Speicherkonfigurationen von Amazon Aurora finden Sie unter Amazon Aurora: Preise.

Informationen zur Auswahl der Speicherkonfiguration beim Erstellen eines DB-Clusters finden Sie unter Erstellen eines DB-Clusters. Informationen zum Ändern der Speicherkonfiguration für einen DB-Cluster finden Sie unter Einstellungen für Amazon Aurora.

Wie sich die Größe des Aurora-Speichers automatisch ändert

Aurora-Cluster-Volumes nehmen automatisch an Größe zu, wenn die Datenmenge in Ihrer Datenbank zunimmt. Weitere Informationen zu den maximalen Volume-Größen von Aurora-Clustern für jede Engine-Version finden Sie unter Amazon Aurora-Größenbeschränkungen. Diese automatische Speicherskalierung wird mit einem leistungsstarken und hochverteilten Speicher-

Subsystem kombiniert. Dadurch ist Aurora eine gute Wahl für Ihre wichtigen Unternehmensdaten, wenn Ihre Hauptziele Zuverlässigkeit und Hochverfügbarkeit sind.

Informationen zum Anzeigen des Volume-Status finden Sie unter Anzeigen des Volume-Status für einen Aurora MySQL-DB-Cluster oder Anzeigen des Volume-Status für einen Aurora PostgreSQL-DB-Cluster. Für Möglichkeiten, die Lagerkosten mit anderen Prioritäten auszugleichen, beschreibt Speicherskalierung, wie die Amazon Aurora-Metriken `AuroraVolumeBytesLeftTotal` und `VolumeBytesUsed` in CloudWatch zu überwachen sind.

Wenn Aurora-Daten entfernt werden, wird der für diese Daten zugewiesene Speicherplatz freigegeben. Beispiele für das Entfernen von Daten sind das Löschen oder Abschneiden einer Tabelle. Diese automatische Reduzierung der Speichernutzung hilft Ihnen, Speichergebühren zu minimieren.

> **Note**
>
> Die hier beschriebenen Speicherlimits und das dynamische Größenanpassungsverhalten gelten für persistente Tabellen und andere Daten, die auf dem Cluster-Volume gespeichert sind.
> Für Aurora PostgreSQL werden temporäre Tabellendaten in der lokalen DB-Instance gespeichert.
> Bei Aurora MySQL Version 2 werden temporäre Tabellendaten standardmäßig auf dem Cluster-Volume für Writer-Instances und im lokalen Speicher für Reader-Instances gespeichert. Weitere Informationen finden Sie unter Speicher-Engine für temporäre Tabellen auf der Festplatte.
> Bei Aurora MySQL Version 3 werden temporäre Tabellendaten in der lokalen DB-Instance oder auf dem Cluster-Volume gespeichert. Weitere Informationen finden Sie unter Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3.
> Die maximale Größe temporärer Tabellen im lokalen Speicher ist durch die maximale lokale Speichergröße der DB-Instance begrenzt. Die lokale Speichergröße hängt von der Instance-Klasse ab, die Sie verwenden. Weitere Informationen erhalten Sie unter Temporäre Speicherlimits für Aurora MySQL und Temporäre Speicherlimits für Aurora PostgreSQL.

Einige Speicherfunktionen, wie die maximale Größe eines Cluster-Volumes und die automatische Größenanpassung beim Entfernen von Daten, hängen von der Aurora-Version des Clusters ab. Weitere Informationen finden Sie unter Speicherskalierung. Außerdem erfahren Sie, wie Sie

Speicherprobleme vermeiden und den zugewiesenen Speicher und freien Speicherplatz in Ihrem Cluster überwachen können.

Informationen zur Abrechnung des Aurora-Datenspeichers

Auch wenn ein Aurora-Cluster-Volume für bestimmte Engine-Versionen auf bis zu 256 Tebibyte (TiB) anwachsen kann, werden Ihnen nur die Kosten für den Speicherplatz berechnet, den Sie in einem Aurora-Cluster-Volume verwenden. In früheren Aurora-Versionen konnte das Cluster-Volume Speicherplatz wiederverwenden, der beim Entfernen von Daten freigegeben wurde, der zugewiesene Speicherplatz nahm jedoch nie ab. Wenn Aurora-Daten jetzt entfernt werden, z. B. durch Löschen einer Tabelle oder Datenbank, verringert sich der insgesamt zugewiesene Speicherplatz um einen vergleichbaren Betrag. So können Sie Speicherkosten senken, indem Sie Tabellen, Indizes, Datenbanken usw. löschen, die Sie nicht mehr benötigen.

> **Tip**
>
> Bei früheren Versionen ohne die Funktion für die dynamische Größenanpassung muss die Speicherverwendung für einen Cluster zurückgesetzt werden, um einen logischen Dump auszuführen und auf einem neuen Cluster wiederherzustellen. Diese Operation kann bei großen Datenmengen sehr lange dauern. Wenn eine solche Situation auftritt, sollten Sie das Upgrade Ihres Clusters auf eine Version durchführen, die die dynamische Größenanpassung des Volumes unterstützt.

Informationen dazu, welche Aurora-Versionen die dynamische Größenanpassung unterstützen und wie Sie Speicherkosten durch Überwachen der Speichernutzung für Ihren Cluster minimieren können, finden Sie unter Speicherskalierung. Informationen zur Speicherabrechnung von Aurora-Backups finden Sie unter Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora. Weitere Informationen über die Preise für Aurora-Datenspeicher finden Sie unter Amazon RDS für Aurora: Preise.

Amazon Aurora-Zuverlässigkeit

Aurora ist darauf ausgelegt, zuverlässig, beständig und fehlertolerant zu sein. Sie können Ihren Aurora-DB-Cluster so gestalten, dass eine bessere Verfügbarkeit erzielt wird, indem Sie beispielsweise Aurora-Replicas hinzufügen und in verschiedenen Availability Zones platzieren. Zudem beinhaltet Aurora mehrere automatische Funktionen, die es zu einer verlässlichen Datenbanklösung machen.

Themen

- Automatische Speicherplatzreparatur
- Überlebensfähiger Seiten-Cache
- Wiederherstellung nach ungeplanten Neustarts

Automatische Speicherplatzreparatur

Da Aurora mehrere Kopien Ihrer Daten in drei Availability Zones aufbewahrt, sind die Risiken für einen laufwerksbedingten Verlust von Daten stark reduziert. Aurora erkennt automatisch Fehler in den Laufwerk-Volumes, aus denen das Cluster-Volume besteht. Wenn ein Segment eines Laufwerk-Volumes ausfällt, repariert Aurora das Segment umgehend. Wenn Aurora das Laufwerk-Segment repariert, verwendet es die Daten in den anderen Volumes des Cluster-Volumes, um sicherzustellen, dass die Daten im reparierten Segment auf dem aktuellen Stand sind. Dadurch vermeidet Aurora Datenverlust und reduziert die Notwendigkeit, eine zeitpunktbezogene Wiederherstellung durchzuführen, um sich von einem Laufwerksausfall zu erholen.

Überlebensfähiger Seiten-Cache

Bei Aurora wird der Seiten-Cache jeder DB-Instance in einem von der Datenbank separaten Prozess verwaltetet, was dem Seiten-Cache ermöglicht, unabhängig von der Datenbank zu überleben. (Der Seiten-Cache wird auch als InnoDB-Puffer-Pool in Aurora MySQL und Puffer-Cache in Aurora PostgreSQL bezeichnet.)

Im unwahrscheinlichen Fall eines Datenbankausfalls bleibt der Seiten-Cache im Arbeitsspeicher, was aktuelle Datenseiten im Seiten-Cache „warm“ hält, wenn die Datenbank neu gestartet wird. Dies bringt einen Leistungsgewinn, da umgangen werden kann, dass die ersten Abfragen Lese-I/O-Operationen zum „Aufwärmen“ des Seiten-Caches ausführen müssen.

Bei Aurora MySQL ist das Verhalten des Seiten-Caches beim Neustart und Failover wie folgt:

- Sie können die Schreiber-Instance ohne die Leser-Instances neu starten.
  - Wenn die Reader-Instances beim Neustart der Writer-Instance nicht neu gestartet werden, verlieren sie ihre Seiten-Caches nicht.
  - Wenn die Reader-Instances beim Neustart der Writer-Instance neu gestartet werden, verlieren sie ihre Seiten-Caches.
- Wenn eine Reader-Instance neu gestartet wird, überleben die Seiten-Caches auf den Writer- und Reader-Instances.

- Wenn der DB-Cluster ein Failover ausführt, wirkt sich dies ähnlich aus wie ein Neustart einer Writer-Instance. Auf der neuen Writer-Instance (zuvor die Reader-Instance) bleibt der Seiten-Cache erhalten, aber auf der Reader-Instance (zuvor der Writer-Instance) überlebt der Seiten-Cache nicht.

Bei Aurora PostgreSQL können Sie die Cluster-Cache-Verwaltung verwenden, um den Seiten-Cache einer bestimmten Reader-Instance beizubehalten, die nach dem Failover zur Writer-Instance wird. Weitere Informationen finden Sie unter Schnelle Wiederherstellung nach Failover mit der Cluster-Cacheverwaltung für Aurora PostgreSQL.

Wiederherstellung nach ungeplanten Neustarts

Aurora ist für die nahezu umgehende Wiederherstellung nach einem ungeplanten Neustart und die weitere Bereitstellung Ihrer Anwendungsdaten ohne das Binärprotokoll ausgelegt. Aurora führt die Wiederherstellung asynchron auf parallelen Threads durch, sodass Ihre Datenbank nach einem ungeplanten Neustart sofort geöffnet und verfügbar ist.

Weitere Informationen erhalten Sie unter Fehlertoleranz für einen Aurora-DB-Cluster und Optimierungen reduzieren die Neustartzeit der Datenbank.

Die folgenden Betrachtungen gelten für eine Binärprotokollierung und die Wiederherstellung nach einem ungeplanten Neustart auf Aurora MySQL:

- Die Aktivierung der Binärprotokollierung auf Aurora wirkt sich direkt auf die Wiederherstellungsdauer nach einem ungeplanten Neustart aus, da die DB-Instance gezwungen wird, das Binärprotokoll wiederherzustellen.
- Der Typ der Binärprotokollierung wirkt sich auf die Größe und die Effizienz der Protokollierung aus. Einige Formate protokollieren dieselbe Menge an Datenbankaktivitäten mehr Informationen in den Binärprotokollen auf als andere. Die folgenden Einstellungen für den Parameter `binlog_format` führen zu unterschiedlichen Mengen an Protokolldaten:
  - `ROW` – die meisten Protokolldaten
  - `STATEMENT` – die wenigsten Protokolldaten
  - `MIXED` – Eine mittlere Menge an Protokolldaten, was normalerweise die beste Kombination von Datenintegrität und Leistung darstellt

  Die Menge der Binärprotokolldaten wirkt sich auf die Wiederherstellungszeit aus. Wenn in den Binärprotokollen mehr Daten protokolliert sind, muss die DB-Instance bei der Wiederherstellung mehr Daten verarbeiten, was wiederum die Wiederherstellungsdauer erhöht.

- Zur Reduzierung des Rechenaufwands und zur Verkürzung der Wiederherstellungszeiten durch binäre Protokollierung können Sie das erweiterte Binärprotokoll verwenden. Das erweiterte Binärprotokoll verbessert die Wiederherstellungszeit der Datenbank um bis zu 99 %. Weitere Informationen finden Sie unter Einrichten eines erweiterten Binärprotokolls für Aurora MySQL.
- Aurora benötigt die Binärprotokolle nicht, um Daten in einem DB-Cluster zu replizieren oder eine zeitpunktbezogene Wiederherstellung (PITR) durchzuführen.
- Wenn Sie das Binärprotokoll nicht für die externe Replikation (oder einen externen Binärprotokoll-Stream) benötigen, empfehlen wir Ihnen, den Parameter `binlog_format` auf `OFF` zu setzen, um die Binärprotokollierung zu deaktivieren. Dies reduziert die Wiederherstellungsdauer.

Weitere Informationen über die Binärprotokollierung und Replikation in Aurora finden Sie unter Replikation mit Amazon Aurora. Weitere Informationen über die Auswirkungen der Verwendung unterschiedlicher MySQL-Replikationstypen finden Sie unter Vorteile und Nachteile einer auf Anweisungen und einer auf Zeilen basierenden Replikation in der MySQL-Dokumentation.

Amazon Aurora-Sicherheit

Die Sicherheit für Amazon Aurora wird auf drei Ebenen verwaltet:

- Mit AWS Identity and Access Management (IAM) können Sie steuern, wer Verwaltungsaktionen in Amazon RDS für Aurora-DB-Cluster und DB-Instances ausführen darf. Wenn Sie eine Verbindung zu AWS mithilfe von IAM-Anmeldeinformationen herstellen, muss Ihr AWS-Konto über IAM-Richtlinien verfügen, die die erforderlichen Berechtigungen zum Ausführen von Amazon-RDS-Verwaltungsvorgängen gewähren. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora.

  Wenn Sie IAM verwenden, um auf die Amazon-RDS-Konsole zuzugreifen, müssen Sie sich zuerst mit Ihren Anmeldeinformationen bei der AWS-Managementkonsole anmelden und anschließend die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds öffnen.
- Aurora-DB-Cluster müssen in einer Virtual Private Cloud (VPC) basierend auf dem Amazon VPC-Service erstellt werden. Mithilfe einer VPC-Sicherheitsgruppe können Sie steuern, welche Geräte und Amazon EC2-Instances Verbindungen zum Endpunkt und Port der DB-Instance für Aurora-DB-Cluster in einer VPC herstellen können. Diese Endpunkt- und Portverbindungen können mithilfe von Transport Layer Security (TLS)/Secure Sockets Layer (SSL) erstellt werden. Zusätzlich können Firewall-Regeln in Ihrem Unternehmen steuern, ob in Ihrem Unternehmen verwendete Geräte

Verbindungen mit einer DB-Instance herstellen dürfen. Weitere Informationen zu VPCs finden Sie unter Amazon VPC und Amazon Aurora.

- Um Anmeldungen und Berechtigungen für einen Amazon Aurora-DB-Cluster zu authentifizieren, können Sie einen der folgenden Ansätze oder eine Kombination aus diesen wählen.
  - Sie können denselben Ansatz wie bei einer eigenständigen DB-Instance in MySQL oder PostgreSQL wählen.

    Techniken zur Authentifizierung von Anmeldungen und Berechtigungen für eigenständige MySQL- oder PostgreSQL-DB-Instances, wie z. B. die Verwendung von SQL-Befehlen oder die Änderung von Datenbankschematabellen, funktionieren auch mit Aurora. Weitere Informationen finden Sie unter Sicherheit in Amazon Aurora MySQL oder Sicherheit in Amazon Aurora PostgreSQL.
  - Sie können die IAM-Datenbank-Authentifizierung verwenden.

    Mit der IAM-Datenbank-Authentifizierung authentifizieren Sie sich bei Ihrem Aurora-DB-Cluster, indem Sie einen Benutzer oder eine IAM-Rolle und ein Authentifizierungstoken verwenden. Ein Authentifizierungstoken ist ein eindeutiger Wert, der mithilfe des Signatur-Version 4-Signiervorgangs erstellt wird. Durch das Verwenden der IAM-Datenbank-Authentifizierung können Sie dieselben Anmeldeinformationen verwenden, um den Zugang zu Ihren AWS-Ressourcen und Ihrer Datenbank zu steuern. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung.
  - Sie können die Kerberos-Authentifizierung für Aurora PostgreSQL und Aurora MySQL verwenden.

    Sie können Kerberos verwenden, um Benutzer zu authentifizieren, wenn diese sich mit Ihrem DB-Cluster von Aurora PostgreSQL und Aurora MySQL verbinden. In diesem Fall arbeitet der DB-Cluster mit AWS Directory Service for Microsoft Active Directory, um die Kerberos-Authentifizierung zu aktivieren. AWS Directory Service for Microsoft Active Directory wird auch AWS Managed Microsoft AD genannt. Wenn Sie alle Ihre Anmeldeinformationen im selben Verzeichnis aufbewahren, können Sie Zeit und Mühe sparen. Sie haben einen zentralen Ort für die Speicherung und Verwaltung von Anmeldeinformationen für mehrere DB-Cluster. Die Verwendung eines Verzeichnisses kann auch Ihr allgemeines Sicherheitsprofil verbessern. Weitere Informationen erhalten Sie unter Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL und Verwenden der Kerberos-Authentifizierung für Aurora MySQL.

Weitere Informationen zur Sicherheitskonfiguration finden Sie unter Sicherheit in Amazon Aurora.

Verwenden von SSL mit Aurora-DB-Clustern

Amazon Aurora-DB-Cluster unterstützten Secure Sockets Layer (SSL)-Verbindungen von Anwendungen mithilfe desselben Prozesses und desselben öffentlichen Schlüssels wie Amazon-RDS-DB-Instances. Weitere Informationen finden Sie unter Sicherheit in Amazon Aurora MySQL, Sicherheit in Amazon Aurora PostgreSQL oder Verwenden von TLS/SSL mit Aurora Serverless v1.

Hohe Verfügbarkeit für Amazon Aurora

Die Amazon-Aurora-Architektur beinhaltet eine Trennung von Speicher und Rechenleistung. Aurora enthält einige Hochverfügbarkeitsfunktionen, die sich auf die Daten in Ihrem DB-Cluster beziehen. Die Daten bleiben auch dann sicher, wenn einige oder alle DB-Instances im Cluster nicht mehr verfügbar sind. Weitere Hochverfügbarkeitsfunktionen gelten für die DB-Instances. Diese Funktionen stellen sicher, dass eine oder mehrere DB-Instances bereit sind, Datenbankanforderungen Ihrer Anwendung zu verarbeiten.

Themen

- Hohe Verfügbarkeit für Aurora-Daten
- Hohe Verfügbarkeit für Aurora-DB-Instances
- Hohe Verfügbarkeit über AWS-Regionen hinweg mit globalen Aurora-Datenbanken
- Fehlertoleranz für einen Aurora-DB-Cluster
- Hochverfügbarkeit mit Amazon-RDS-Proxy

Hohe Verfügbarkeit für Aurora-Daten

Aurora speichert Kopien der Daten in einem DB-Cluster über mehrere Availability Zones in einer einzigen AWS-Region. Dieser Speichervorgang von Aurora erfolgt unabhängig davon, ob die DB-Instances im DB-Cluster mehrere Availability Zones umfassen. Weitere Informationen zu finden Sie unter Auror Verwalten eines Amazon Aurora-DB-Clusters.

Wenn Daten in die primäre DB-Instance geschrieben werden, repliziert Aurora die Daten synchron über die Availability Zones auf sechs Speicherknoten, die Ihrem Cluster-Volume zugeordnet sind. Auf diese Weise werden eine Datenredundanz erreicht, das Blockieren von I/Os verhindert und Latenzspitzen bei System-Backups minimiert. Wenn Sie eine DB-Instance mit hoher Verfügbarkeit ausführen, kann dies die Verfügbarkeit bei geplanten Systemwartungen verbessern. Außerdem kann

dies Ihre Datenbanken bei Ausfällen und bei Nichtverfügbarkeit von Availability Zones schützen. Weitere Informationen über Availability Zones finden Sie unter Regionen und Availability Zones.

Hohe Verfügbarkeit für Aurora-DB-Instances

Nachdem Sie die primäre (Writer-)Instance erstellt haben, können Sie bis zu 15 schreibgeschützte Aurora-Replikate erstellen. Die Aurora-Replikate werden auch als Reader-Instances bezeichnet. Aurora-Replikate verwenden asynchrone Replikation, um Hochverfügbarkeit zu unterstützen, ohne die Leistung der primären Instance zu beeinträchtigen.

Während des täglichen Betriebs können Sie einen Teil der Arbeit für leseintensive Anwendungen auslagern, indem Sie die Reader-Instances zum Verarbeiten von `SELECT`-Abfragen verwenden. Wenn sich ein Problem auf die primäre Instance auswirkt, übernimmt eine dieser Reader-Instances als primäre Instance. Dieser Mechanismus wird als Failover bezeichnet. Viele Aurora-Funktionen gelten für den Failover-Mechanismus. Beispielsweise erkennt Aurora Datenbankprobleme und aktiviert den Failover-Mechanismus bei Bedarf automatisch. Aurora verfügt jedoch über Funktionen, die die Zeit für den Abschluss des Failovers verkürzen. Dadurch wird die Zeit minimiert, in der die Datenbank während eines Failovers nicht zum Schreiben verfügbar ist.

Aurora wurde für eine schnellstmögliche Wiederherstellung konzipiert. Der schnellste Weg zur Wiederherstellung ist oft ein Neustart oder ein Failover auf dieselbe DB-Instance. Ein Neustart ist schneller und mit weniger Aufwand verbunden als ein Failover.

Um eine Verbindungszeichenfolge zu verwenden, die auch dann unverändert bleibt, wenn ein Failover eine neue primäre Instance heraufstuft, stellen Sie eine Verbindung mit dem Cluster-Endpunkt her. Der Cluster-Endpunkt stellt stets die aktuelle primäre Instance im Cluster dar. Weitere Informationen zum Cluster-Endpunkt finden Sie unter Amazon-Aurora-Endpunktverbindungen.

> **Tip**
>
> Innerhalb jeder AWS-Region stellen Availability Zones Standorte dar, die sich voneinander unterscheiden, um im Falle von Ausfällen eine Isolierung zu ermöglichen. Wir empfehlen Ihnen, die primäre Instance und die Reader-Replikate in Ihrem DB-Cluster über mehrere Availability Zones zu verteilen, um die Verfügbarkeit Ihres DB-Clusters zu verbessern. Auf diese Weise verursacht ein Problem, das eine gesamte Availability Zone betrifft, keinen Ausfall für Ihren Cluster.
> Sie können einen Multi-AZ-DB-Cluster einrichten, indem Sie beim Erstellen des Clusters eine einfache Auswahl treffen. Sie können die AWS-Managementkonsole, AWS CLI oder

die Amazon-RDS-API benutzen. Sie können auch einen vorhandenen Aurora-DB-Cluster in einen Multi-AZ-DB-Cluster konvertieren, indem Sie eine neue Reader-DB-Instance hinzufügen und eine andere Availability Zone angeben.

Hohe Verfügbarkeit über AWS-Regionen hinweg mit globalen Aurora-Datenbanken

Für hohe Verfügbarkeit in mehreren AWS-Regionen können Sie globale Aurora-Datenbanken einrichten. Jede globale Aurora-Datenbank erstreckt sich über mehrere AWS-Regionen und ermöglicht globales Lesen mit geringer Latenz sowie eine Notfallwiederherstellung nach Ausfällen in einer AWS-Region. Aurora repliziert auf asynchrone Weise alle Daten und Aktualisierungen aus der primären AWS-Region für jede der sekundären Regionen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.

Fehlertoleranz für einen Aurora-DB-Cluster

Ein Aurora-DB-Cluster ist darauf ausgelegt, fehlertolerant zu sein. Das Cluster-Volumen erstreckt sich über mehrere Availability Zones (AZs) in einer einzelnen AWS-Region und jede Availability Zone beinhaltet eine Kopie der Daten im Cluster-Volumen. Diese Funktionalität bedeutet, dass Ihr DB-Cluster einen Fehler in einer Availability Zone ohne Datenverlust und mit einer nur sehr kurzen Unterbrechung des Services toleriert.

Wenn die primäre Instance in einem DB-Cluster ausfällt, führt ein automatisches Failover auf eine von zwei Arten zu einer neuen primären Instance:

- Durch das Hochstufen einer bestehenden Aurora-Replica zu einer neuen primären Instance
- Durch das Erstellen einer neuen primären Instance

Wenn ein DB-Cluster über ein oder mehrere Aurora verfügt, wird eine Aurora-Replica während eines Ausfallereignisses zur primären Instance hochgestuft. Ein Fehlerereignis hat eine kurze Unterbrechung zufolge, während die Lese- und Schreibvorgänge mit einer Ausnahme fehlschlagen. Jedoch wird der Service im Normalfall in weniger als 60 Sekunden und oft sogar schon nach 30 Sekunden wiederhergestellt. Wir empfehlen Ihnen, mindestens ein oder mehrere Aurora Replicas in zwei oder mehreren verschiedenen Availability Zones zu erstellen, um die Verfügbarkeit Ihres DB-Clusters zu erhöhen.

> **Tip**
>
> In Aurora MySQL können Sie die Verfügbarkeit während eines Failovers verbessern, indem Sie mehr als eine Reader-DB-Instance in einem Cluster haben. In Aurora MySQL startet Aurora nur die Writer-DB-Instance und die Reader-Instance neu, auf die das Failover erfolgt ist. Andere Reader-Instances im Cluster bleiben während eines Failovers verfügbar, um Abfragen über Verbindungen zum Reader-Endpunkt fortzusetzen.
> Sie können die Verfügbarkeit auch während eines Failovers verbessern, indem Sie RDS-Proxy mit Ihrem Aurora-DB-Cluster verwenden. Weitere Informationen finden Sie unter Hochverfügbarkeit mit Amazon-RDS-Proxy.

Sie können die Reihenfolge, in der Ihre Aurora Replicas auf die primäre Instance nach einem Ausfall hochgestuft werden, anpassen, indem Sie jedem Replica eine Priorität zuweisen. Prioritäten liegen im Bereich zwischen 0 als höchste Priorität und 15 als niedrigste Priorität. Wenn die primäre Instance ausfällt, stuft Amazon RDS die Aurora-Replica mit der höchsten Priorität zur neuen primären Instance hoch. Sie können die Priorität eines Aurora Replica jederzeit anpassen. Das Ändern der Priorität löst kein Failover aus.

Mehrere Aurora Replicas können dieselbe Priorität haben, was zu Hochstufungsebenen führt. Wenn zwei oder mehrere Aurora-Replicas dieselbe Priorität haben, stuft Amazon RDS die größte Replica hoch. Wenn zwei oder mehrere Aurora-Replicas dieselbe Priorität und Größe haben, stuft Amazon RDS eine beliebige Replica auf derselben Hochstufungsebene hoch.

> **Note**
>
> Bei der Identifizierung eines Failover-Ziels spielen mehrere Faktoren eine Rolle. Nach fünf erfolglosen Failover-Versuchen werden keine Hochstufungsstufen mehr berücksichtigt.

Wenn der DB-Cluster keine Aurora-Replikate hat, wird die primäre Instance bei einem Fehlerereignis in derselben AZ neu erstellt. Ein Fehlerereignis hat eine Unterbrechung zufolge, während die Lese- und Schreibvorgänge mit einer Ausnahme fehlschlagen. Der Service wird wiederhergestellt, wenn die primäre Instance erstellt wird. Dies dauert im Normalfall weniger als 10 Minuten. Das Hochstufen eines Aurora Replicas auf eine primäre Instance funktioniert viel schneller als das Erstellen einer neuen primären Instance.

Angenommen, die primäre Instance in Ihrem Cluster ist aufgrund eines Ausfalls, der sich auf eine gesamte AZ auswirkt, nicht verfügbar. In diesem Fall hängt die Möglichkeit, eine neue primäre Instance online zu stellen, davon ab, ob Ihr Cluster eine Multi-AZ-Konfiguration verwendet.

- Wenn Ihr bereitgestellter oder Aurora Serverless v2-Cluster Reader-Instances in anderen AZs enthält, verwendet Aurora den Failover-Mechanismus, um eine dieser Reader-Instances als neue primäre Instance hochzustufen.
- Wenn Ihr bereitgestellter oder Aurora Serverless v2-Cluster nur eine einzelne DB-Instance enthält oder wenn sich die primäre Instance und alle Reader-Instances in derselben AZ befinden, müssen Sie manuell eine oder mehrere neue DB-Instances in einer anderen AZ erstellen.
- Wenn Ihr Cluster Aurora Serverless v1 verwendet, erstellt Aurora automatisch eine neue DB-Instance in einer anderen AZ. Dieser Prozess erfordert jedoch eine Host-Ersetzung und dauert daher länger als ein Failover.

> **Note**
>
> Amazon Aurora unterstützt auch die Replikation mit einer externen MySQL-Datenbank oder einer RDS-MySQL-DB-Instance. Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Hochverfügbarkeit mit Amazon-RDS-Proxy

Mit RDS-Proxy können Sie Anwendungen erstellen, die Datenbankfehler transparent tolerieren können, ohne komplexen Fehlerbehandlungscode schreiben zu müssen. Der Proxy leitet den Datenverkehr automatisch an eine neue Datenbank-Instance weiter, wobei Anwendungsverbindungen erhalten bleiben. Außerdem umgeht er Domain Name System (DNS)-Caches, um die Failover-Zeiten für Aurora-Multi-AZ-Datenbanken um bis zu 66 % zu reduzieren. Weitere Informationen finden Sie unter Amazon-RDS-Proxy für Aurora.

Replikation mit Amazon Aurora

Es gibt mehrere Replikations-Optionen für Aurora. Jeder Aurora-DB-Cluster verfügt über eine integrierte Replikation zwischen mehreren DB-Instances im selben Cluster. Sie können auch die Replikation mit Ihrem Aurora-Cluster als Quelle oder Ziel einrichten. Wenn Sie Daten in oder aus

einem Aurora-Cluster replizieren, können Sie zwischen integrierten Funktionen wie Aurora globalen Datenbanken oder den herkömmlichen Replikationsmechanismen für die MySQL- oder PostgreSQL-DB-Engines wählen. Sie können die geeigneten Optionen auswählen, je nachdem, welche die richtige Kombination aus Hochverfügbarkeit, Komfort und Leistung für Ihre Anforderungen bietet. In den folgenden Abschnitten wird erläutert, wie und wann Sie die jeweilige Technik wählen.

Themen

- Aurora-Replikate
- Replikation mit Aurora MySQL
- Replikation mit Aurora PostgreSQL

Aurora-Replikate

Wenn Sie eine zweite, dritte und so weiter DB-Instance in einem bereitgestellten Aurora-DB-Cluster erstellen, wird Aurora automatisch die Replikation von der Writer-DB-Instance zu allen anderen DB-Instances einrichten. Diese anderen DB-Instances sind schreibgeschützt und werden als Aurora-Replicas bezeichnet. Wir bezeichnen sie auch als Reader-Instances, wenn wir besprechen, wie Sie Writer- und Reader-DB-Instances innerhalb eines Clusters kombinieren können.

Aurora-Replicas haben zwei Hauptzwecke. Sie können über sie Abfragen stellen, um die Lesevorgänge für Ihre Anwendung zu skalieren. In der Regel tun Sie dies, indem Sie eine Verbindung zum Reader-Endpunkt des Clusters herstellen. Auf diese Weise kann Aurora die Last für schreibgeschützte Verbindungen auf so viele Aurora-Replicas verteilen, wie Sie im Cluster haben. Aurora-Replicas tragen auch dazu bei, die Verfügbarkeit zu erhöhen. Wenn die Writer-Instance in einem Cluster nicht verfügbar ist, stuft Aurora automatisch eine der Reader-Instances hoch, um ihren Platz als neuer Autor einzunehmen.

Ein Aurora-DB-Cluster kann bis zu 15 Aurora-Replicas enthalten. Die Aurora-Replikate können über die Availability Zones verteilt werden, über die sich ein DB-Cluster innerhalb einer AWS-Region erstreckt.

Die Daten in Ihrem DB-Cluster haben ihre eigenen Hochverfügbarkeits- und Zuverlässigkeitsfunktionen, unabhängig von den DB-Instances im Cluster. Wenn Sie mit den Aurora-Speicherfunktionen nicht vertraut sind, lesen Sie Übersicht über Amazon-Aurora-Speicher. Das DB-Cluster-Volume besteht physisch aus mehreren Kopien der Daten für den DB-Cluster. Die primäre Instance und die Aurora-Replicas im DB-Cluster sehen die Daten im Cluster-Volume als ein einziges logisches Volume.

Daher geben alle Aurora-Replikate die gleichen selben Daten als Abfrageergebnisse bei minimaler Replikatsverzögerung zurück. Diese Verzögerung beträgt in der Regel sehr viel weniger als 100 Millisekunden nach dem Schreiben einer Aktualisierung durch die primäre Instance. Die Replica-Verzögerung variiert in Abhängigkeit vom Veränderungsgrad in der Datenbank. In den Zeitabschnitten, in denen sehr viele Schreibvorgänge in der Datenbank durchgeführt werden, könnte es zu einer erhöhten Replica-Verzögerung kommen.

> **Note**
>
> Ein Aurora-Replikat wird in den folgenden Aurora-PostgreSQL-Versionen automatisch neu gestartet, wenn die Kommunikation mit der Schreiber-DB-Instance für mehr als 60 Sekunden unterbrochen wird:
>
> - 14.6 und ältere Versionen
> - 13.9 und ältere Versionen
> - 12.13 und ältere Versionen
> - Alle Versionen von Aurora PostgreSQL 11
>
> Mit der Funktion zur Leseverfügbarkeit werden die Aurora-Replikate nicht automatisch neu gestartet. Weitere Informationen zur Leseverfügbarkeitsfunktion sowie zu den unterstützten Versionen finden Sie unter Verbesserung der Leseverfügbarkeit von Aurora Replicas.

Aurora Replicas funktionieren für das Skalieren von Lesevorgängen, da sie in Ihrem Cluster-Volume vollständig für Lesevorgänge bereit stehen. Schreibvorgänge werden von der primären Instance verwaltet. Da das Cluster-Volume zwischen allen Instances in Ihrem DB-Cluster geteilt wird, bedeutet es nur einen geringen Aufwand, um eine Kopie der Daten für die einzelnen Aurora-Replicas zu replizieren.

Sie können Aurora Replicas als Failover-Vorgaben verwenden, um die Verfügbarkeit zu erhöhen. Das heißt, wenn die primäre Instance ausfällt, wird ein Aurora-Replikat zur primären Instance heraufgestuft. Es gibt dann eine kurze Unterbrechung, während der Lese- und Schreibanfragen an die primäre Instance mit einer Ausnahme fehlschlagen.

Die Hochstufung einer Aurora-Replica per Failover ist eine viel schnellere Methode als die Neuerstellung der primären Instance. Wenn Ihr Aurora-DB-Cluster keine Aurora-Replikate enthält,

steht Ihr DB-Cluster während der Wiederherstellung Ihrer DB-Instance nach dem Ausfall nicht zur Verfügung.

Bei einem Failover werden möglicherweise einige der Aurora-Replikate je nach Version der DB-Engine neu gestartet. In Aurora MySQL startet Aurora nur die Schreiber-DB-Instance und das Failover-Ziel während eines Failovers neu. Informationen zum Neustartverhalten verschiedener Aurora-DB-Engine-Versionen finden Sie unter Neustart eines Amazon Aurora DB-Clusters oder einer Amazon Aurora DB-Instance. Hinweise dazu, was mit Seitencaches beim Neustart oder Failover geschieht, finden Sie unterÜberlebensfähiger Seiten-Cache.

Für Szenarien mit hoher Verfügbarkeit empfehlen wir Ihnen, mindestens ein Aurora-Replikat zu erstellen. Diese sollten dieselbe DB-Instance-Class haben wie die primäre Instance und in verschiedenen Availability Zones für Ihren Aurora-DB-Cluster liegen. Weitere Informationen über Aurora-Replikate als Failover-Ziele finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.

Sie können keine verschlüsselte Aurora-Replica für einen unverschlüsselten Aurora-DB-Cluster erstellen. Sie können keine unverschlüsselte Aurora-Replica für einen verschlüsselten Aurora-DB-Cluster erstellen.

> **Tip**
>
> Sie können Aurora-Replicas innerhalb eines Aurora-Clusters als Ihre einzige Form der Replikation verwenden, um Ihre Daten hochverfügbar zu halten. Sie können die integrierte Aurora-Replikation auch mit den anderen Arten der Replikation kombinieren. Dies kann dazu beitragen, ein zusätzliches Maß an hoher Verfügbarkeit und geografischer Verteilung Ihrer Daten zu gewährleisten.

Details zum Erstellen von Aurora-Replicas finden Sie unter Hinzufügen von Aurora-Replicas zu einem DB-Cluster.

Replikation mit Aurora MySQL

Zusätzlich zu Aurora-Replicas haben Sie die folgenden Möglichkeiten für Replikationen mit Aurora MySQL:

- Aurora-MySQL-DB-Cluster in verschiedenen AWS-Regionen.

  - Sie können Daten über mehrere Regionen mithilfe einer Aurora globalen Datenbank replizieren. Details hierzu finden Sie unter Hohe Verfügbarkeit über AWS-Regionen hinweg mit globalen Aurora-Datenbanken.
  - Sie können ein Aurora-Lesereplikat eines Aurora-MySQL-DB-Clusters in einer anderen AWS-Region erstellen, indem Sie die MySQL-Binlog-Replikation (Binärprotokoll) verwenden. Jeder Cluster kann über bis zu fünf auf diese Weise erstellte Lesereplikate verfügen, die sich jeweils in einer anderen Region befinden.
- Zwei Aurora-MySQL-DB Cluster in der gleichen Region, indem Sie mit dem MySQL-Binärprotokoll (binlog) eine Replikation einrichten.
- Eine RDS-for-MySQL-DB-Instance als Datenquelle und ein Aurora-MySQL-DB-Cluster, indem Sie ein Aurora-Lesereplikat einer RDS-for-MySQL-DB-Instance erstellen. Üblicherweise verwenden Sie diese Methode eher bei Migrationsvorgängen nach Aurora MySQL als für die fortlaufende Replikation.

Weitere Informationen zur Replikation mit Aurora MySQL finden Sie unter Replikation mit Amazon Aurora My SQL.

Replikation mit Aurora PostgreSQL

Zusätzlich zu Aurora-Replicas haben Sie die folgenden Möglichkeiten für Replikationen mit Aurora PostgreSQL:

- Ein primärer Aurora-DB-Cluster in einer Region und bis zu 10 schreibgeschützte sekundäre DB-Cluster in verschiedenen Regionen unter Verwendung einer globalen Aurora-Datenbank. Aurora PostgreSQL unterstützt keine regionsübergreifenden Aurora-Replikate. Sie können jedoch die globale Aurora-Datenbank verwenden, um die Lesefunktionen Ihres Aurora-PostgreSQL-DB-Clusters auf mehr als eine AWS-Region zu skalieren und die Verfügbarkeitsziele zu erreichen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.
- Zwei Aurora-PostgreSQL-DB-Cluster in derselben Region, indem die logische Replikationsfunktion von PostgreSQL verwendet wird.
- Eine RDS-for-PostgreSQL-DB-Instance als Datenquelle und ein Aurora-PostgreSQL-DB-Cluster, indem ein Aurora-Lesereplikat einer RDS-for-PostgreSQL-DB-Instance erstellt wird. Üblicherweise verwenden Sie diese Methode eher bei Migrationsvorgängen nach Aurora PostgreSQL als für die fortlaufende Replikation.

Weitere Informationen zur Replikation mit Aurora PostgreSQL finden Sie unter Replikation mit Amazon Aurora PostgreSQL.

Abrechnung von DB-Instances für Aurora

Von Amazon RDS in einem Aurora-Cluster bereitgestellte Instances werden basierend auf den folgenden Komponenten abgerechnet:

- DB-Instance-Stunden (pro Stunde): Basierend auf der DB-Instance-Klasse der DB-Instance (z. B. db.t2.small oder db.m4.large). Die Preise werden auf Stundenbasis aufgeführt, aber Rechnungen werden jetzt auf die Sekunde genau kalkuliert und zeigen die Zeiten im Dezimalformat an. Die RDS-Nutzung wird pro Sekunde berechnet, wobei mindestens für 10 Minuten zu zahlen ist. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
- Speicher (pro GiB pro Monat: Die Speicherkapazität, die Sie für die DB-Instance bereitstellen lassen. Bei einer Skalierung Ihrer zur Verfügung gestellten Speicherkapazität im laufenden Monat werden die Gebühren entsprechend anteilig erfasst. Weitere Informationen finden Sie unter Amazon-Aurora-Speicher.
- Ein-/Ausgabe (I/O)-Anforderungen (pro 1 Mio. Anfragen) – Gesamtzahl der Speicher-I/O-Anforderungen, die Sie in einem Abrechnungszeitraum ausgeführt haben, nur für die DB-Clusterkonfiguration von Aurora Standard.

  Weitere Informationen zur I/O-Abrechnung von Amazon Aurora finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.
- Backup-Speicher (pro GiB pro Monat) – Ein Backup-Speicher ist Speicher, der automatisierten Datenbanksicherungen und allen aktiven Datenbank-Snapshots zugeordnet ist, die Sie erstellt haben. Wenn Sie die Aufbewahrungszeit Ihrer Backups erhöhen oder zusätzliche Datenbank-Snapshots erstellen, belegt Ihre Datenbank dementsprechend mehr Backup-Speicher. Die sekundengenaue Abrechnung gilt nicht für den Backup-Speicher (gemessen in GB/Monat).

  Weitere Informationen finden Sie unter Sichern und Wiederherstellen eines Amazon-Aurora-DB-Clusters.
- Datenübertragung (pro GB) Datenübertragung in und aus Ihrer DB-Instance vom oder zum Internet und von oder zu anderen AWS-Regionen. Hilfreiche Beispiele finden Sie im AWS-Blogbeitrag Exploring Data Transfer Costs for AWS Managed Databases.

Amazon RDS bietet die folgenden Kaufoptionen, um Ihnen die Möglichkeit zu bieten, Ihre Kosten basierend auf Ihren Anforderungen zu optimieren.

- On-Demand instances (On-Demand-Instances) – Sie bezahlen stundenweise für DB-Instance-Stunden, die Sie nutzen. Die Preise werden auf Stundenbasis aufgeführt, aber Rechnungen werden jetzt auf die Sekunde genau kalkuliert und zeigen die Zeiten im Dezimalformat an. Die RDS-Nutzung wird jetzt pro Sekunde berechnet, wobei mindestens für 10 Minuten zu zahlen ist.
- Reserved instances (Reserved Instances) – Reservieren Sie eine DB-Instance für einen Zeitraum von einem oder drei Jahren und erhalten Sie einen im Vergleich zu den Preisen für eine On-Demand-DB-Instance erheblichen Rabatt. Durch die Nutzung von Reserved Instances können Sie mehrere Instances innerhalb einer Stunde in Betrieb nehmen, starten, löschen oder beenden und die Reserved Instance-Rabatte für alle Instances erhalten.
- Aurora Serverless v2 – Aurora Serverless v2 bietet On-Demand-Kapazität, bei der die Abrechnungseinheit Aurora Capacity Unit (ACU)-Stunden anstelle von DB-Instance-Stunden ist. Die Aurora Serverless v2-Kapazität steigt und sinkt in einem von Ihnen angegebenen Bereich, abhängig von der Last Ihrer Datenbank. Sie können einen Cluster konfigurieren, dessen gesamte Kapazität Aurora Serverless v2 ist. Sie können auch ein Kombination aus Aurora Serverless v2 und On-Demand oder reservierten bereitgestellten Instances konfigurieren. Weitere Informationen zur Funktionsweise von Aurora Serverless v2-ACUs finden Sie unter Funktionsweise von Aurora Serverless v2.
- Aurora PostgreSQL Limitless Database — Aurora PostgreSQL Limitless Database ist eine automatisierte, horizontale Skalierungsfunktion, die über den Schreibdurchsatz und die Speichergrenzen einer einzelnen DB-Instance hinaus skaliert. Limitless Database verteilt den Workload auf mehrere Aurora-Writer-Instances und behält gleichzeitig den einfachen Betrieb als einzelne Datenbank bei. Limitless Database bietet On-Demand-Kapazität, wobei die Abrechnungseinheit Aurora-Kapazitätseinheitenstunden (ACU) in einer DB-Shard-Gruppe ist. Weitere Informationen zur Funktionsweise der ACUs von Limitless Database finden Sie unter Erstellen eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet.

Informationen zur Preisgestaltung von Aurora finden Sie in der Aurora-Preisliste.

Themen

- On-Demand-DB-Instances für Aurora
- Reserved-DB-Instances für Amazon Aurora

On-Demand-DB-Instances für Aurora

On-Demand-DB-Instances von Amazon RDS werden basierend auf der Klasse der DB-Instance abgerechnet (z. B. db.t3.small oder db.m5.large). Informationen zu Amazon-RDS-Preisen finden Sie auf der Amazon-RDS-Produktseite.

Die Abrechnung für eine DB-Instance beginnt, sobald die DB-Instance verfügbar ist. Die Preise werden auf Stundenbasis aufgeführt, aber Rechnungen werden jetzt auf die Sekunde genau kalkuliert und zeigen die Zeiten im Dezimalformat an. Die Amazon-RDS-Nutzung wird pro Sekunde berechnet, wobei mindestens für 10 Minuten zu zahlen ist. Im Falle einer abrechenbaren Konfigurationsänderung, wie Skalierung der Rechen- oder Speicherkapazität, wird Ihnen eine Mindestdauer von 10 Minuten in Rechnung gestellt. Die Abrechnung wird solange fortgesetzt, bis die DB-Instance beendet wird, was beim Löschen der DB-Instance oder beim Ausfall der DB-Instance passiert.

Wenn Sie nicht mehr mit Gebühren für Ihre DB-Instance belastet werden wollen, müssen Sie diese stoppen oder löschen, um zu vermeiden, dass zusätzliche Stunden der DB-Instance in Rechnung gestellt werden. Weitere Informationen über die Zustände der DB-Instance, die Ihnen in Rechnung gestellt werden, finden Sie unter Anzeigen von DB-Instance-Status in einem Aurora-Cluster.

Angehaltene DB-Instances

Während Ihre DB-Instance angehalten wird, werden nur Gebühren für bereitgestellten Speicher in Rechnung gestellt, einschließlich bereitgestellter IOPS. Es werden Ihnen auch Gebühren für den Backup-Speicher berechnet, einschließlich des Speichers für manuelle Snapshots und automatische Backups innerhalb des von Ihnen festgelegten Aufbewahrungsfensters. Für DB-Instance-Stunden werden Ihnen keine Gebühren in Rechnung gestellt.

Multi-AZ-DB-Instances

Eine Multi-AZ-Einrichtung verbessert die Datenbeständigkeit und -verfügbarkeit, indem automatisch ein synchrones Standby-Replikat in einer anderen Availability Zone bereitgestellt und verwaltet wird. Aufgrund der zusätzlichen Ressourcen und der erhöhten Verfügbarkeit sind Multi-AZ-Bereitstellungen teurer als Single-AZ-Bereitstellungen und können aufgrund der zusätzlichen Standby-Instance und der zugehörigen Ressourcen etwa doppelt so viel kosten.

Beachten Sie die folgenden wichtigen Details zu Multi-AZ-Preisen:

- Datenverarbeitungskosten: Werden pro DB-Instance-Stunde für Primär- und Standby-Instances abgerechnet.

- Speicherkosten: Werden pro GB-Monat für den Speicher berechnet, der für Primär- und Standby-Instances bereitgestellt wird.
- Datenübertragungskosten: Die Replikation zwischen der Primär- und der Standby-Instance ist in den Kosten enthalten, es können jedoch je nach Nutzung weitere Datenübertragungsgebühren anfallen.

Um Ihre monatlichen Kosten basierend auf Ihrem spezifischen Anwendungsfall und AWS-Region genau abzuschätzen, können Sie die AWS -Preisrechner verwenden. Dieses Tool ermöglicht Ihnen die Eingabe Ihrer Konfigurationsdetails und bietet eine umfassende Kostenaufschlüsselung.

> **Note**
>
> Die Preise können sich ändern. Die aktuellen Informationen finden Sie auf der Seite „Amazon RDS Pricing“.

Reserved-DB-Instances für Amazon Aurora

Mit Reserved DB-Instances können Sie eine DB-Instance für eine ein- oder dreijährige Laufzeit reservieren. Reservierte DB-Instances bieten Ihnen einen deutlichen Rabatt im Vergleich zu den bedarfsorientierten Preisen für DB-Instances. Bei reservierten DB-Instances handelt es sich nicht um physische Instances, sondern um einen Fakturierungsrabatt für die Nutzung gewisser On-Demand-Instances in Ihrem Konto. Rabatte für Reserved DB-Instances sind an den Instance-Typ und die AWS-Region gebunden.

Der allgemeine Prozess für das Arbeiten mit reservierten DB-Instances ist: Zuerst Informationen über verfügbare reservierte DB-Instance-Angebote erhalten, dann ein reserviertes DB-Instance-Angebot kaufen und schließlich Informationen über Ihre vorhandenen reservierten DB-Instances erhalten.

Informationen zum Erwerben von Reserved-DB-Instances und Anzeigen der Abrechnung für Reserved DB-Instances finden Sie in den folgenden Abschnitten.

- Erwerben von Reserved-DB-Instances für Amazon Aurora
- Anzeigen der Abrechnung für reservierte DB-Instances für Amazon Aurora

Übersicht über Reservierte DB-Instances

Wenn Sie eine Reserved DB-Instance in Amazon RDS kaufen, erwerben Sie eine Verpflichtung, eine reduzierte Rate für einen bestimmten DB-Instance-Typ für die Dauer der Reserved DB-Instance zu erhalten. Um eine Amazon RDS Reserved DB-Instance zu verwenden, erstellen Sie eine neue DB-Instance, genau wie bei einer On-Demand-Instance.

Die neu erstellte DB-Instance muss mit den Spezifikationen der Reserved DB-Instance in folgenden Punkten übereinstimmen:

- AWS-Region
- DB-Engine (Die Versionsnummer der DB-Engine muss nicht übereinstimmen.)
- DB-Instance-Typ

Wenn die Spezifikationen der neuen DB-Instance mit einer vorhandenen Reserved DB-Instance für Ihr Konto übereinstimmen, wird Ihnen der angebotene diskontierte Preis für die Reserved DB-Instance in Rechnung gestellt. Andernfalls wird der DB-Instance eine On-Demand-Rate berechnet.

Sie können eine DB-Instance, die Sie als Reserved-DB-Instance verwenden, ändern. Wenn die Änderung innerhalb der Spezifikationen der reservierten DB-Instance liegt, gilt der diskontierte Preis teilweise oder vollständig für die geänderte DB-Instance. Wenn die Änderung außerhalb der Spezifikationen liegt, z. B. die Änderung der Instance-Klasse, gilt der diskontierte Preis nicht mehr. Weitere Informationen finden Sie unter Größenflexible Reservierte DB-Instances.

Themen

- Angebotstypen
- Flexibilität bezüglich der Konfiguration des Aurora-DB-Clusters
- Größenflexible Reservierte DB-Instances
- Abrechnungsbeispiele für Reserved DB-Instances von Aurora
- Löschen einer Reserved DB-Instance

Weitere Informationen zu reservierten DB-Instances samt Preisen finden Sie unter Amazon RDS Reserved Instances.

Angebotstypen

Reservierte DB-Instances sind in drei Varianten verfügbar – Keine Vorauszahlung, Teilweise Vorauszahlung und Vollständige Vorauszahlung. Auf diese Weise können Sie die Amazon-RDS-Kosten auf der Basis der erwarteten Nutzung optimieren.

> **Note**
>
> Nicht alle RDS-Instance-Klassen unterstützen alle Reserved-Instance-Angebotstypen. Beispielsweise bieten einige Instance-Klassen möglicherweise nicht die Option ohne Vorabzahlung an. Um die Verfügbarkeit zu überprüfen, überprüfen Sie die Reserved Instance-Angebote im AWS-Managementkonsole oder verwenden Sie den `describe-reserved-db-instances-offerings` AWS CLI Befehl.

Keine Vorabzahlung

Diese Option ermöglicht den Zugriff auf eine reservierte DB-Instance, ohne dass eine Vorauszahlung erforderlich ist. Ihre reservierte DB-Instance ohne Vorauszahlung rechnet für jede Stunde innerhalb der Laufzeit einen ermäßigten Stundensatz ab, unabhängig von der Nutzung,

und es ist keine Vorauszahlung erforderlich. Diese Option ist lediglich als Reservierung für die Dauer eines Jahres verfügbar.

Teilweise Vorauszahlung

Diese Option beruht darauf, dass ein Teil der reservierten DB-Instance im Voraus bezahlt wird. Die innerhalb der Laufzeit verbleibenden Stunden werden unabhängig von der Nutzung zu einem vergünstigten Stundensatz berechnet. Diese Option ersetzt die vorherige Heavy Utilization-Option.

Komplette Vorauszahlung

Die vollständige Zahlung erfolgt zu Beginn der Laufzeit, unabhängig von der Nutzungsdauer und ohne weitere Kosten innerhalb der Restlaufzeit.

Wenn Sie die konsolidierte Fakturierung nutzen, werden alle Konten in der Organisation wie ein einziges Konto behandelt. Das bedeutet, dass alle Konten in der Organisation den stündlichen Kostenvorteil für Reserved DB-Instances erhalten können, die durch ein anderes Konto erworben wurden. Weitere Informationen zur konsolidierten Fakturierung finden Sie unter Reservierte Amazon-RDS-DB-Instances im Benutzerhandbuch AWS -Fakturierungs- und Kostenverwaltung.

Flexibilität bezüglich der Konfiguration des Aurora-DB-Clusters

Sie können Reserved DB-Instances von Aurora mit beiden DB-Cluster-Konfigurationen verwenden:

- Aurora I/O-Optimized— Sie zahlen nur für die Nutzung und Speicherung Ihrer DB-Cluster, ohne zusätzliche Gebühren für Lese- und I/O Schreibvorgänge.
- Aurora Standard— Neben der Nutzung und Speicherung Ihrer DB-Cluster zahlen Sie auch einen Standardtarif pro 1 Million Anfragen für I/O Operationen.

Aurora berücksichtigt automatisch den Preisunterschied zwischen diesen Konfigurationen. Aurora I/O-Optimizedverbraucht 30 % mehr normalisierte Einheiten pro Stunde als Aurora Standard.

Weitere Informationen zu den Speicherkonfigurationen von Aurora-Clustern finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora. Weitere Informationen zu den Preisen für Speicherkonfigurationen von Aurora-Clustern finden Sie unter Amazon Aurora: Preise.

Größenflexible Reservierte DB-Instances

Beim Kauf einer Reserved DB-Instance wird unter anderem die Instance-Klasse angegeben, z. B. db.r5.large. Weitere Informationen zu DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Wenn Sie eine DB-Instance haben und diese auf größere Kapazität skalieren müssen, wird Ihre Reserved DB-Instance automatisch auf Ihre skalierte DB-Instance angewendet. Das heißt, Ihre Reserved DB-Instances werden automatisch auf alle DB-Instance-Klassengrößen angewendet. Größenflexible reservierte DB-Instances sind für DB-Instances mit derselben AWS-Region DB-Engine verfügbar. Größenflexible Reserved DB-Instances können nur innerhalb ihres Instance-Klassentyps skalieren. Eine reservierte DB-Instance für eine db.r6i.large kann beispielsweise für eine db.r6i.xlarge gelten, aber nicht für eine db.r6id.large oder db.r7g.large, da db.r6id.large und db.r7g.large unterschiedliche Instance-Klassentypen sind.

Größenflexible reservierte DB-Instances sind für die folgenden Aurora-Datenbank-Engines verfügbar:

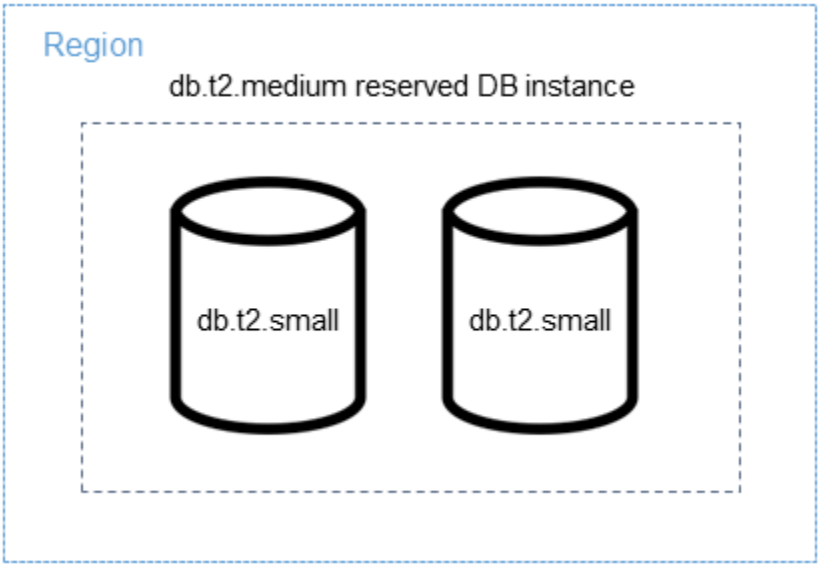
- Aurora MySQL
- Aurora PostgreSQL

Sie können die Nutzung für verschiedene Reserved DB-Instance-Größen vergleichen, indem Sie normalisierte Einheiten pro Stunde verwenden. Beispielsweise entspricht eine Nutzungseinheit auf zwei db.r3.large DB-Instances acht normalisierten Nutzungseinheiten pro Stunde auf einem db.r3.small. Die folgende Tabelle zeigt die Anzahl von normalisierten Einheiten pro Stunde für jede DB-Instance-Größe.

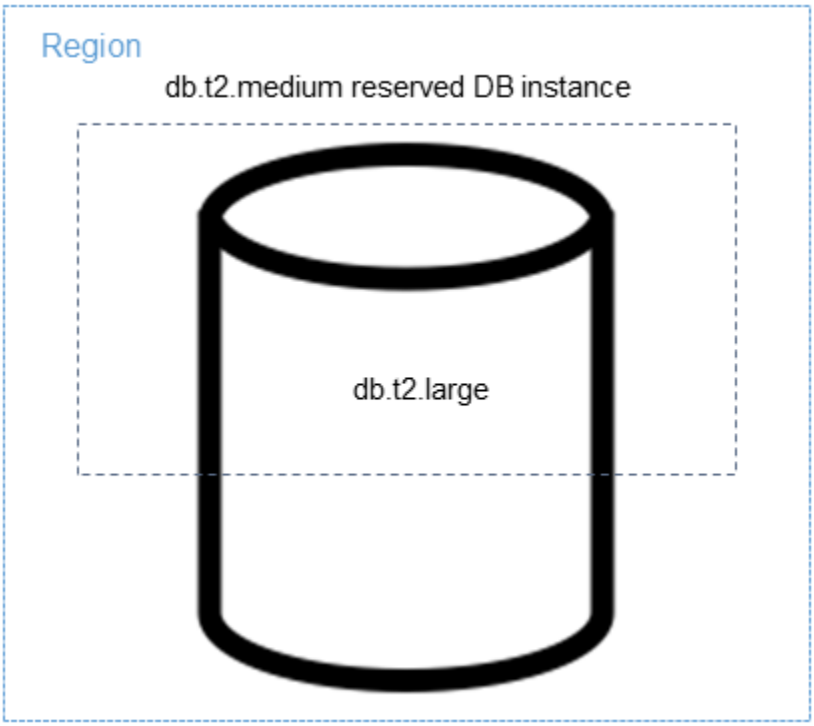
Instance-Größe	Normalisierte Einheiten pro Stunde für eine DB-Instance, Aurora Standard	Normalisierte Einheiten pro Stunde für eine DB-Instance, Aurora I/O-Optimized	Normalisierte Einheiten pro Stunde für drei DB-Instances (Writer und zwei Reader), Aurora Standard	Normalisierte Einheiten pro Stunde für drei DB-Instances (Writer und zwei Reader), Aurora I/O-Optimized
small	1	1.3	3	3.9
Medium	2	2.6	6	7.8
large	4	5.2	12	15,6
xlarge	8	10.4	24	31,2
2xlarge	16	20,8	48	62,4

Instance-Größe	Normalisierte Einheiten pro Stunde für eine DB-Instance, Aurora Standard	Normalisierte Einheiten pro Stunde für eine DB-Instance, Aurora I/O-Optimized	Normalisierte Einheiten pro Stunde für drei DB-Instances (Writer und zwei Reader), Aurora Standard	Normalisierte Einheiten pro Stunde für drei DB-Instances (Writer und zwei Reader), Aurora I/O-Optimized
4xlarge	32	41,6	96	124,8
8xlarge	64	83,2	192	249,6
12xlarge	96	124,8	288	374,4
16xlarge	128	166,4	384	499,2
24xlarge	192	249,6	576	748,8
32xlarge	256	332,8	768	998,4

Angenommen, Sie kaufen eine Reserved `db.t2.medium`-DB-Instance und haben zwei `db.t2.small`-DB-Instances in Ihrem Konto, die in der gleichen AWS-Region ausgeführt werden. In diesem Fall wird der Rabatt in vollem Umfang auf beide Instances angewendet.



Wenn in Ihrem Konto eine `db.t2.large` Instance ausgeführt wird AWS-Region, wird der Abrechnungsvorteil alternativ auf 50 Prozent der Nutzung der DB-Instance angerechnet.



> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-Instance-Klassenarten.

Abrechnungsbeispiele für Reserved DB-Instances von Aurora

Die folgenden Beispiele veranschaulichen die Preise für Reserved DB-Instances für Aurora-DB-Cluster, die sowohl die DB-Cluster-Konfiguration Aurora Standard als auch die Konfiguration Aurora I/O-Optimized verwenden.

Beispiel mit Aurora Standard

Der Preis für eine Reserved DB-Instance bietet keinen Rabatt für die Kosten in Verbindung mit Speicher, Backups und I/O. Das folgende Beispiel veranschaulicht die Gesamtkosten pro Monat für eine Reserved DB-Instance:

- Eine Reserved Single-AZ-DB-Instance von Aurora MySQL der Klasse db.r5.large in der Region USA Ost (Nord-Virginia) kostet 0,19 USD pro Stunde oder 138,70 USD pro Monat.
- Aurora-Speicher zu einem Preis von 0,10 USD pro GiB pro Monat (ungefähr 45,60 USD pro Monat in diesem Beispiel).
- Aurora I/O zu einem Preis von 0,20 USD pro 1 Million Anfragen (für dieses Beispiel gehen Sie von 20 USD pro Monat aus)
- Aurora-Sicherungsspeicher zu einem Preis von 0,021 USD pro GiB pro Monat (ungefähr 30 USD pro Monat in diesem Beispiel).

Wenn Sie die Kosten all dieser Optionen (138,70 USD + 45,60 USD + 20 USD + 30 USD) mit der Reserved DB-Instance addieren, liegen die monatlichen Gesamtkosten bei 234,30 USD.

Wenn Sie sich statt für eine Reserved DB-Instance für eine On-Demand-DB-Instance entscheiden, kostet eine Single-AZ-Instance der Klasse db.r5.large von Aurora MySQL in der Region USA Ost (Nord-Virginia) 0,29 USD pro Stunde oder 217,50 USD pro Monat. Wenn Sie die Kosten all dieser Optionen (217,50 USD + 45,60 USD + 20 USD + 30 USD) mit der On-Demand-DB-Instance addieren, liegen die monatlichen Gesamtkosten bei 313,10 USD. Sie sparen fast 79 USD pro Monat, indem Sie die reservierte DB-Instance verwenden.

Beispiel für die Verwendung eines Aurora Standard DB-Clusters mit zwei Reader-Instances

Um Reserved Instances für Aurora-DB-Cluster zu verwenden, kaufen Sie einfach eine Reserved Instance für jede DB-Instance im Cluster.

Gehen wir noch einmal auf das erste Beispiel ein und nehmen an, Sie verfügen über einen DB-Cluster von Aurora MySQL mit einer Writer-DB-Instance und zwei Aurora Replicas, also insgesamt drei DB-Instances im Cluster. Für die beiden Aurora Replicas fallen keine zusätzlichen Speicher- oder Backup-Kosten an. Wenn Sie drei reservierte db.r5.large Aurora MySQL-DB-Instances erwerben, belaufen sich Ihre Kosten auf 234,30 USD (für die Writer-DB-Instance) + 2 * (138,70 USD + 20 USD I/O pro Aurora Replica), also insgesamt 551,70 USD pro Monat.

Die entsprechenden On-Demand-Kosten für einen Aurora MySQL-DB-Cluster mit einer Writer-DB-Instance und zwei Aurora Replicas betragen 313,10$ + 2 * (217,50$ + 20$ I/O pro Instance), was insgesamt 788,10$ pro Monat entspricht. Sie sparen fast 236,40 USD pro Monat, wenn Sie die Reserved DB-Instances verwenden.

Beispiel mit Aurora I/O-Optimized

Sie können Ihre vorhandenen Aurora Standard Reserved DB-Instances mit Aurora I/O-Optimized wiederverwenden. Um die Vorteile Ihrer Reserved Instance-Rabatte mit Aurora I/O-Optimized in vollem Umfang nutzen zu können, können Sie 30 % zusätzliche Reserved Instances kaufen, die Ihren aktuellen Reserved Instances ähneln.

Die folgende Tabelle zeigt Beispiele dafür, wie Sie die zusätzlichen Reserved Instances abschätzen können, wenn Sie Aurora I/O-Optimized verwenden. Wenn es sich bei den erforderlichen Reserved Instances um einen Bruchteil handelt, können Sie die Größenflexibilität nutzen, die Reserved Instances bieten, um auf eine ganze Zahl zu kommen. In diesen Beispielen bezieht sich „aktuell“ auf die Aurora Standard Reserved Instances, über die Sie jetzt verfügen. Zusätzliche Reserved Instances sind die Anzahl der Aurora Standard Reserved Instances, die Sie kaufen müssen, um Ihre aktuellen Reserved Instance-Rabatte bei Verwendung von Aurora I/O-Optimized beizubehalten.

DB-Instance-Klasse	Aktuelle Aurora Standard Reserved Instances	Reserved Instances, die für Aurora I/O-Optimized erforderlich sind	Zusätzlich erforderliche Reserved Instances	Zusätzlich erforderliche Reserved Instances bei Nutzung der Größenflexibilität
db.r6g.large	10	10 * 1,3 = 13	3 * db.r6g.large	3 * db.r6g.large
db.r6g.4xlarge	20	20 * 1,3 = 26	6 * db.r6g.4x large	6 * db.r6g.4x large
db.r6g.12xlarge	5	5 * 1,3 = 6,5	1,5 * db.r6g.12 xlarge	db.r6g.12xlarge, r6g.4xlarge und r6g.2xlarge jeweils einmal (0,5 * db.r6g.12 xlarge = 1 * db.r6g.4xlarge + 1 * db.r6g.2x large)

DB-Instance-Klasse	Aktuelle Aurora Standard Reserved Instances	Reserved Instances, die für Aurora I/O-Optimized erforderlich sind	Zusätzlich erforderliche Reserved Instances	Zusätzlich erforderliche Reserved Instances bei Nutzung der Größenflexibilität
db.r6i.24xlarge	15	15 * 1,3 = 19,5	4,5 * db.r6i.24xlarge	4 * db.r6i.24xlarge + 1 * db.r6i.12xlarge (0,5 * db.r6i.24xlarge = 1 * db.r6i.12xlarge)

Beispiel für die Verwendung eines Aurora I/O-Optimized DB-Clusters mit zwei Reader-Instances

Sie verfügen über einen DB-Cluster von Aurora MySQL mit einer Writer-DB-Instance und zwei Aurora Replicas, also insgesamt drei DB-Instances im Cluster. Diese verwenden die DB-Cluster-Konfiguration Aurora I/O-Optimized. Um Reserved DB-Instances für diesen Cluster zu verwenden, müssten Sie vier Reserved DB-Instances derselben DB-Instance-Klasse kaufen. Drei DB-Instances verbrauchen mit Aurora I/O-Optimized 3,9 normalisierte Einheiten pro Stunde, verglichen mit 3 normalisierten Einheiten pro Stunde für drei DB-Instances mit Aurora Standard. Sie sparen jedoch die monatlichen Kosten für jede DB-Instance. I/O

> **Note**
>
> Die Preise in diesem Beispiel sind Beispiele und entsprechen möglicherweise nicht den tatsächlichen Preisen. Informationen zur Preisgestaltung von Aurora finden Sie unter Aurora-Preise.

Löschen einer Reserved DB-Instance

In den Bedingungen für eine Reserved DB-Instance ist eine einjährige oder dreijährige Verpflichtung enthalten. Sie können eine Reserved DB-Instance nicht stornieren. Sie können jedoch eine DB-Instance löschen, die durch einen Rabatt für eine Reserved DB-Instance abgedeckt ist. Der Vorgang

zum Löschen einer DB-Instance, für die ein Rabatt für eine Reserved DB-Instance gilt, ist der gleiche wie für jede andere DB-Instance.

Die Vorabkosten werden Ihnen in Rechnung gestellt, unabhängig davon, ob Sie die Ressourcen nutzen.

Wenn Sie eine DB-Instance löschen, die durch einen Rabatt für eine Reserved DB-Instance gedeckt ist, können Sie eine andere DB-Instance mit kompatiblen Spezifikationen starten und den ermäßigten Preis während der Reservierungslaufzeit (ein Jahr oder drei Jahre) erhalten. In diesem Fall erhalten Sie den Rabatt während des Reservierungszeitraums (ein Jahr oder drei Jahre).

Erwerben von Reserved-DB-Instances für Amazon Aurora

Sie können die AWS-Managementkonsole, und die RDS-API verwenden AWS CLI, um mit reservierten DB-Instances zu arbeiten.

Konsole

Sie können die verwenden AWS-Managementkonsole , um mit reservierten DB-Instances zu arbeiten, wie in den folgenden Verfahren gezeigt.

Preise und Informationen zu verfügbaren Angeboten für reservierte DB-Instances erhalten

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Reserved instances (Reservierte Instances) aus.
3. Klicken Sie auf Reserved DB-Instance kaufen.
4. Um die Produktbeschreibung einzusehen, wählen Sie die DB-Engine und den Lizenztyp aus.
5. Wählen Sie für DB-Instance-Klasse die DB-Instance-Klasse aus.
6. Wählen Sie für Bereitstellungsoption aus, ob Sie eine Single-AZ- oder eine Multi-AZ-Bereitstellung der DB-Instance wünschen.

> **Note**
>
> Bei Reserved Instances von Amazon Aurora ist die Bereitstellungsoption grundsätzlich auf Single-AZ-DB-Instance festgelegt. Wenn Sie jedoch einen Aurora-DB-Cluster erstellen, lautet die Standardbereitstellungsoption Ein Aurora-Replikat oder einen Reader in einer anderen AZ erstellen (Multi-AZ).

Sie müssen eine Reserved DB-Instance für jede Instance erwerben, die Sie verwenden möchten, einschließlich Aurora Replicas. Daher müssen Sie für Multi-AZ-Bereitstellungen in Aurora zusätzliche reservierte DB-Instances erwerben.

7. Für Laufzeit wählen Sie die Zeitspanne aus, für welche die DB-Instance reserviert werden soll.
8. Wählen Sie für Angebotstyp den Angebotstyp aus.

   Nachdem Sie die Angebotsart ausgewählt haben, werden Ihnen die Preisinformationen angezeigt.

   Important

   Klicken Sie auf Abbrechen, um den Kaufvorgang für die Reserved DB-Instance abzubrechen und Kosten zu vermeiden.

Nachdem Sie Informationen über die verfügbaren reservierten DB-Instance-Angebote erhalten haben, können Sie diese Informationen verwenden, um ein Angebot zu erwerben, wie in der folgenden Vorgehensweise gezeigt.

Kauf einer reservierten DB-Instance

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Reserved instances (Reservierte Instances) aus.
3. **Important**

   Stellen Sie sicher, dass Sie sich in der korrekten AWS-Region befinden. Reserved-DB-Instances sind regionsspezifisch und können nicht zwischen Regionen übertragen werden. Überprüfen Sie anhand der Regionsauswahl oben rechts in der Konsole, ob Sie die Reserved Instance in der gewünschten Region erwerben.
4. Klicken Sie auf Reserved DB-Instance kaufen.
5. Um die Produktbeschreibung einzusehen, wählen Sie die DB-Engine und den Lizenztyp aus.
6. Wählen Sie für DB-Instance-Klasse die DB-Instance-Klasse aus.
7. Wählen Sie für Multi-AZ-Bereitstellung aus, ob Sie eine Single-AZ- oder eine Multi-AZ-Bereitstellung der DB-Instance wünschen.

> **Note**
>
> Bei Reserved Instances von Amazon Aurora ist die Bereitstellungsoption grundsätzlich auf Single-AZ-DB-Instance festgelegt. Wenn Sie einen DB-Cluster von Amazon Aurora aus Ihrer Reserved DB-Instance erstellen, ist der DB-Cluster automatisch als Multi-AZ eingerichtet. Sie müssen eine Reserved DB-Instance für jede DB-Instance erwerben, die Sie verwenden möchten, einschließlich Aurora Replicas.

8. Für die Auswahl des Laufzeit wählen Sie die Zeitspanne aus, für welche die DB-Instance reserviert werden soll.
9. Wählen Sie für Angebotstyp den Angebotstyp aus.

   Nachdem Sie die Angebotsart ausgewählt haben, werden Ihnen die Preisinformationen angezeigt.
   Konsole für den Kauf von Reserved DB-Instances
10. (Optional) Sie können den Reserved DB-Instances, die Sie kaufen, Ihre eigene Kennung zuweisen, um die Übersicht zu behalten. Geben Sie unter Reservierte ID eine Kennzeichnung für Ihre reservierte DB-Instance ein.
11. Wählen Sie Absenden aus.

    Ihre Reserved DB-Instance wird gekauft und dann in der Liste Reserved Instances angezeigt.

Nachdem Sie reservierte DB-Instances gekauft haben, erhalten Sie Informationen über Ihre reservierten DB-Instances, wie in der folgenden Vorgehensweise gezeigt.

Um Informationen über reservierte DB-Instances für Ihr AWS Konto zu erhalten

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Bereich Navigation die Option Reserved Instances (Reservierte Instances) aus.

   Es erscheinen die reservierten DB-Instances für Ihr Konto. Um detaillierte Informationen zu einer bestimmten Reserved DB-Instance anzuzeigen, wählen Sie diese in der Liste aus. Sie können dann detaillierte Informationen über diese Instance im Detailbereich am unteren Rand der Konsole anzeigen.

AWS CLI

Sie können die verwenden AWS CLI , um mit reservierten DB-Instances zu arbeiten, wie in den folgenden Beispielen gezeigt.

Example Erhalten von verfügbaren Reserved DB-Instance-Angeboten

Rufen Sie den AWS CLI Befehl auf, um Informationen über verfügbare Angebote für reservierte DB-Instances zu erhalten `describe-reserved-db-instances-offerings`.

```
aws rds describe-reserved-db-instances-offerings
```

Diese Aktion führt zu folgender oder einer ähnlichen Ausgabe:

```
OFFERING  OfferingId                            Class         Multi-AZ  Duration  Fixed
 Price  Usage Price  Description  Offering Type
OFFERING  438012d3-4052-4cc7-b2e3-8d3372e0e706  db.r3.large   y         1y
 1820.00 USD  0.368 USD    mysql        Partial  Upfront
OFFERING  649fd0c8-cf6d-47a0-bfa6-060f8e75e95f  db.r3.small   n         1y
 227.50 USD  0.046 USD    mysql        Partial  Upfront
OFFERING  123456cd-ab1c-47a0-bfa6-12345667232f  db.r3.small   n         1y
 162.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   0.123   USD       Hourly
OFFERING  123456cd-ab1c-37a0-bfa6-12345667232d  db.r3.large   y         1y
 700.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   1.25    USD       Hourly
OFFERING  123456cd-ab1c-17d0-bfa6-12345667234e  db.r3.xlarge  n         1y
 4242.00 USD   2.42 USD    mysql        No       Upfront
```

Nachdem Sie Informationen über die verfügbaren reservierten DB-Instance-Angebote erhalten haben, können Sie diese Informationen verwenden, um ein Angebot zu erwerben.

Um eine reservierte DB-Instance zu erwerben, verwenden Sie den AWS CLI Befehl `purchase-reserved-db-instances-offering`mit den folgenden Parametern:

- `--reserved-db-instances-offering-id` – Die ID des Angebots, das Sie erwerben möchten Siehe das vorhergehende Beispiel, um die Angebots-ID zu erhalten.
- `--reserved-db-instance-id` – Sie können den Reserved DB-Instances, die Sie kaufen, Ihre eigene Kennung zuweisen, um die Übersicht zu behalten.

Example Kauf einer Reserved DB-Instance

Im folgenden Beispiel wird das reservierte DB-Instance-Angebot mit der ID *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f* gekauft und der Identifier von *MyReservation* zugewiesen.

Für Linux, macOS oder Unix:

```
aws rds purchase-reserved-db-instances-offering \
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-db-instance-id MyReservation
```

Für Windows:

```
aws rds purchase-reserved-db-instances-offering ^
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-db-instance-id MyReservation
```

Daraufhin erhalten Sie ein Ergebnis, das dem hier dargestellten entspricht:

```
RESERVATION  ReservationId      Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State           Description  Offering Type
RESERVATION  MyReservation      db.r3.small  y         2011-12-19T00:30:23.247Z  1y
    455.00 USD   0.092 USD    1      payment-pending  mysql        Partial  Upfront
```

Nachdem Sie Reserved DB-Instances gekauft haben, erhalten Sie Informationen über Ihre Reserved DB-Instances.

Um Informationen über reservierte DB-Instances für Ihr AWS Konto zu erhalten, rufen Sie den AWS CLI Befehl auf `describe-reserved-db-instances`, wie im folgenden Beispiel gezeigt.

Example Erhalten von Reserved DB-Instances

```
aws rds describe-reserved-db-instances
```

Daraufhin erhalten Sie ein Ergebnis, das dem hier dargestellten entspricht:

```
RESERVATION  ReservationId     Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State    Description  Offering Type
RESERVATION  MyReservation     db.r3.small  y         2011-12-09T23:37:44.720Z  1y
   455.00 USD   0.092 USD    1      retired  mysql        Partial  Upfront
```

RDS-API

Sie können die RDS-API verwenden, um mit Reserved DB-Instances zu arbeiten:

- Um Informationen zu verfügbaren Reserved DB-Instance-Angeboten zu erhalten, rufen Sie die Amazon-RDS-API-Aktion auf `DescribeReservedDBInstancesOfferings`.
- Nachdem Sie Informationen über die verfügbaren reservierten DB-Instance-Angebote erhalten haben, können Sie diese Informationen verwenden, um ein Angebot zu erwerben. Rufen Sie dazu die RDS-API-Operation `PurchaseReservedDBInstancesOffering` mit den folgenden Parametern auf:
  - `--reserved-db-instances-offering-id` – Die ID des Angebots, das Sie erwerben möchten
  - `--reserved-db-instance-id` – Sie können den Reserved DB-Instances, die Sie kaufen, Ihre eigene Kennung zuweisen, um die Übersicht zu behalten.
- Nachdem Sie Reserved DB-Instances gekauft haben, erhalten Sie Informationen über Ihre Reserved DB-Instances. Rufen Sie die Aktion `DescribeReservedDBInstances`-RDS-API auf.

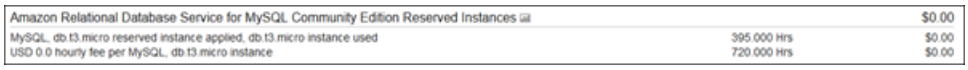
Anzeigen der Abrechnung für reservierte DB-Instances für Amazon Aurora

Sie können die Abrechnung für Ihre reservierten DB-Instances im Abrechnungs-Dashboard im AWS-Managementkonsole.

So zeigen Sie Reserved DB-Instance Abrechnung an

1. Melden Sie sich an der AWS-Managementkonsole an.
2. Wählen Sie im Kontomenü oben rechts Abrechnungs-Dashboard aus.
3. Wählen Sie oben rechts im Dashboard Rechnungsdetails aus.
4. Erweitern Sie unter AWS-Servicegebühren den Dienst für relationale Datenbanken.
5. Erweitern Sie die AWS-Region, in der sich Ihre Reserved DB-Instances befinden, beispielsweise USA West (Oregon).

   Ihre reservierten DB-Instances und ihre stündlichen Gebühren für den aktuellen Monat werden unter Amazon Relational Database Service for ***`Datenbank-Engine`*** Reserved Instances angezeigt.



Die reservierte DB-Instance in diesem Beispiel wurde „All Upfront“ gekauft, daher fallen keine Stundengebühren an.

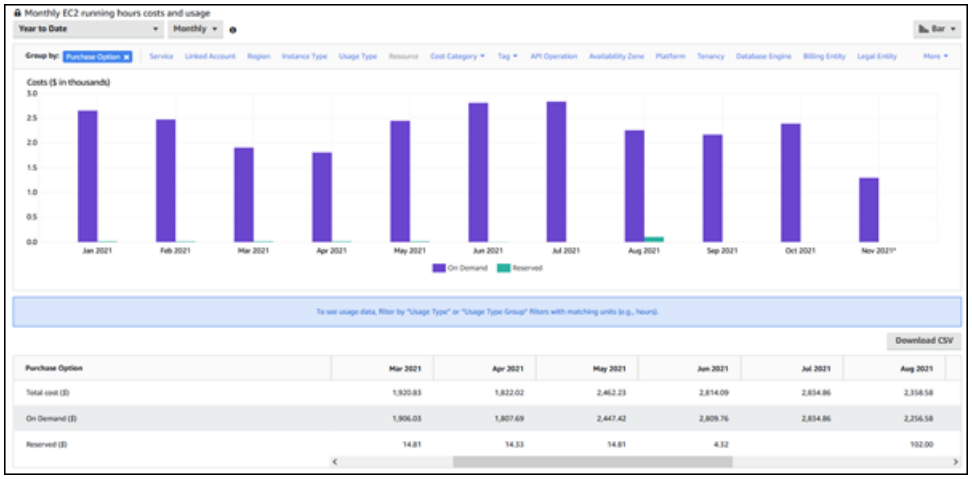
6. Wählen Sie das Cost Explorer-Symbol (Balkendiagramm) neben der Überschrift Reserved Instances.

   Der Cost Explorer zeigt die monatlichen EC2-Betriebsstundenkosten und das Nutzungsdiagramm an.

7. Deaktivieren Sie den Filter Verwendungstyp-Gruppe rechts neben dem Diagramm.

8. Wählen Sie den Zeitraum und die Zeiteinheit aus, für die Sie die Nutzungskosten untersuchen möchten.

   Das folgende Beispiel zeigt die Nutzungskosten für On-Demand- und reservierte DB-Instances für das bisherige Jahr bis nach Monaten an.



Die Kosten für reservierte DB-Instance von Januar bis Juni 2021 sind monatliche Gebühren für eine Partial-Upfront-Instance, während die Kosten im August 2021 eine einmalige Gebühr für eine All Upfront-Instance sind.

Der Rabatt der reservierten Instances für die Partial Upfront-Instance lief im Juni 2021 aus, die DB-Instance wurde jedoch nicht gelöscht. Nach dem Ablaufdatum wurde es einfach zum On-Demand-Tarif berechnet.

Einrichten Ihrer Umgebung für Amazon Aurora

Bevor Sie Amazon Aurora zum ersten Mal verwenden, führen Sie die folgenden Aufgaben aus.

Themen

- Melden Sie sich an für ein AWS-Konto
- Erstellen eines Benutzers mit Administratorzugriff
- Erteilen programmgesteuerten Zugriffs
- Ermitteln der Anforderungen
- Ermöglichen des Zugriffs auf den DB-Cluster in der VPC durch Erstellen einer Sicherheitsgruppe

Wenn Sie bereits über eine verfügen AWS-Konto, Ihre Aurora-Anforderungen kennen und lieber die Standardeinstellungen für IAM- und VPC-Sicherheitsgruppen verwenden möchten, fahren Sie mit fort. Erste Schritte mit Amazon Aurora

Melden Sie sich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Administratorbenutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Sie können Ihre aktuellen Kontoaktivitäten jederzeit einsehen und Ihr Konto verwalten, indem Sie zu https://aws.amazon.com/gehen und Mein Konto auswählen.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

   Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center Benutzerhandbuch.
2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center Benutzerhandbuch.

Erteilen programmgesteuerten Zugriffs

Benutzer benötigen programmatischen Zugriff, wenn sie mit AWS außerhalb des interagieren möchten. AWS-Managementkonsole Die Art und Weise, wie programmatischer Zugriff gewährt wird, hängt vom Benutzertyp ab, der zugreift. AWS

Um Benutzern programmgesteuerten Zugriff zu gewähren, wählen Sie eine der folgenden Optionen.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Empfohlen) Verwenden Sie Konsolenanmeldeinformationen als temporäre Anmeldeinformationen, um programmatische Anfragen an AWS CLI AWS SDKs, oder zu signieren. AWS APIs	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen zu den AWS CLI finden Sie unter Anmeldung für AWS lokale Entwicklung im AWS Command Line Interface Benutzerhandbuch.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
		• Weitere Informationen finden Sie unter Anmeldung für AWS lokale Entwicklung im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. AWS SDKs
Mitarbeiteridentität (Benutzer, die in IAM Identity Center verwaltet werden)	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an das AWS CLI AWS SDKs, oder AWS APIs zu signieren.	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen zu den AWS CLI finden Sie unter Konfiguration der AWS CLI zur Verwendung AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch. • Informationen zu AWS SDKs Tools und AWS APIs finden Sie unter IAM Identity Center-Authentifizierung im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.
IAM	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an das AWS CLI AWS SDKs, oder zu signieren. AWS APIs	Folgen Sie den Anweisungen unter Verwenden temporärer Anmeldeinformationen mit AWS Ressourcen im IAM-Benutzerhandbuch.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Nicht empfohlen) Verwenden Sie langfristige Anmeldeinformationen, um programmatische Anfragen an das AWS CLI AWS SDKs, oder zu signieren. AWS APIs	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen dazu AWS CLI finden Sie unter Authentifizierung mithilfe von IAM-Benutzeranmeldeinformationen im AWS Command Line Interface Benutzerhandbuch. • Informationen zu AWS SDKs und Tools finden Sie unter Authentifizieren mit langfristigen Anmeldeinformationen im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. • Weitere Informationen finden Sie unter Verwaltung von Zugriffsschlüsseln für IAM-Benutzer im IAM-Benutzerhandbuch. AWS APIs

Ermitteln der Anforderungen

Die Grundbausteine für Aurora sind DB-Cluster. Einem DB-Cluster können eine oder mehrere DB-Instances angehören. Ein DB-Cluster gibt eine Netzwerkadresse, den sogenannten Cluster-Endpunkt, an. Ihre Anwendungen verbinden sich immer dann mit dem vom DB-Cluster gezeigten Cluster-Endpunkt, wenn sie auf die Datenbanken zugreifen müssen, die in diesem DB-Cluster erstellt wurden. Die Informationen, die Sie beim Erstellen des DB-Clusters angeben, steuern Konfigurationselemente wie Arbeitsspeicher, Datenbank-Engine und -Version, Netzwerkkonfiguration, Sicherheit und Wartungszeiträume.

Bevor Sie einen DB-Cluster und eine Sicherheitsgruppe erstellen, müssen Sie Ihre DB-Cluster- und Netzwerkanforderungen kennen. Hier einige wichtige Dinge, die Sie berücksichtigen sollten:

- Ressourcenanforderungen – Welche Anforderungen haben Sie an den Arbeitsspeicher und den Prozessor für Ihre Anwendung oder Ihren Service? Sie verwenden diese Einstellungen, wenn Sie bestimmen, welche DB-Instance-Klasse Sie beim Erstellen Ihres DB-Clusters nutzen. Spezifikationen für alle verfügbaren DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
- VPC, Subnetz und Sicherheitsgruppe– Ihr DB-Cluster befindet sich in einer Virtual Private Cloud (VPC). Um eine Verbindung zu einem DB-Cluster herzustellen, müssen Regeln für Sicherheitsgruppen konfiguriert werden. Die folgende Liste beschreibt die Regeln für jede VPC-Option:
  - Standard-VPC — Wenn Ihr AWS Konto über eine Standard-VPC in der AWS Region verfügt, ist diese VPC für die Unterstützung von DB-Clustern konfiguriert. Wenn Sie beim Erstellen des DB-Clusters die Standard-VPC angeben:
    - Sie müssen eine VPC-Sicherheitsgruppe anlegen, die Verbindungen von der Anwendung oder dem Service zum Aurora-DB-Cluster autorisiert. Verwenden Sie die Option Sicherheitsgruppe auf der VPC-Konsole oder AWS CLI um VPC-Sicherheitsgruppen zu erstellen. Weitere Informationen finden Sie unter Schritt 3: Erstellen einer VPC-Sicherheitsgruppe.
    - Sie müssen die Standard-DB-Subnetzgruppe angeben. Wenn dies der erste DB-Cluster ist, den Sie in der AWS Region erstellt haben, erstellt Amazon RDS bei der Erstellung des DB-Clusters die Standard-DB-Subnetzgruppe.
  - Benutzerdefinierte VPC – wenn Sie beim Erstellen eines DB-Clusters eine benutzerdefinierte VPC angeben möchten:
    - Sie müssen eine VPC-Sicherheitsgruppe anlegen, die Verbindungen von der Anwendung oder dem Service zum Aurora-DB-Cluster autorisiert. Verwenden Sie die Option Sicherheitsgruppe auf der VPC-Konsole oder AWS CLI um VPC-Sicherheitsgruppen zu erstellen. Weitere Informationen finden Sie unter Schritt 3: Erstellen einer VPC-Sicherheitsgruppe.
    - Die VPC muss bestimmte Anforderungen erfüllen, um DB-Cluster bereitzustellen, z. B. das Vorhandensein von zwei Subnetzen in jeweils einer separaten Availability Zone. Weitere Informationen finden Sie unter Amazon VPC und Amazon Aurora.
    - Sie müssen eine DB-Subnetzgruppe angeben, die definiert, welche Subnetze in dieser VPC vom DB-Cluster genutzt werden können. Weitere Informationen erhalten Sie im Abschnitt "DB-Subnetzgruppen" unter Arbeiten mit einer DB-Cluster in einer VPC.

- Hohe Verfügbarkeit: Benötigen Sie Failover-Unterstützung? Bei Aurora erstellt eine Multi-AZ-Bereitstellung eine primäre Instance sowie Aurora-Replicas. Sie können die primäre Instance und Aurora-Replicas so konfigurieren, dass sie sich in verschiedenen Availability Zones befinden, um Failover-Support zu erhalten. Wir empfehlen Multi-AZ-Bereitstellungen, um die hohe Verfügbarkeit von Produktions-Workloads sicherzustellen. Zu Entwicklungs- und Testzwecken können Sie eine non-Multi-AZ Bereitstellung verwenden. Weitere Informationen finden Sie unter Hohe Verfügbarkeit für Amazon Aurora.
- IAM-Richtlinien: Verfügt Ihr AWS Konto über Richtlinien, die die für die Durchführung von Amazon RDS-Vorgängen erforderlichen Berechtigungen gewähren? Wenn Sie eine Verbindung AWS mit IAM-Anmeldeinformationen herstellen, muss Ihr IAM-Konto über IAM-Richtlinien verfügen, die die für die Durchführung von Amazon RDS-Vorgängen erforderlichen Berechtigungen gewähren. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora.
- Offene Ports: Auf welchem TCP/IP Port wird Ihre Datenbank lauschen? Die Firewalls einiger Unternehmen blockieren möglicherweise Verbindungen zum Standard-Port für Ihre Datenbank-Engine. Wenn die Firewall Ihres Unternehmens den Standardport blockiert, wählen Sie einen anderen Port für den neuen DB-Cluster aus. Nach dem Erstellen eines DB-Clusters, der einen angegebenen Port abfragt, können Sie diesen Port ändern, indem Sie den DB-Cluster modifizieren.
- AWS Region: In welcher AWS Region möchten Sie Ihre Datenbank haben? Indem sich die Datenbank nahe bei der Anwendung oder dem Webdienst befindet, könnten Netzwerklatenzen verringert werden. Weitere Informationen finden Sie unter Regionen und Availability Zones.

Sobald Ihnen die benötigten Informationen zur Erstellung der Sicherheitsgruppe und des DB-Clusters vorliegen, fahren Sie mit dem nächsten Schritt fort.

Ermöglichen des Zugriffs auf den DB-Cluster in der VPC durch Erstellen einer Sicherheitsgruppe

Ihr DB-Cluster wird in einer VPC erstellt. Sicherheitsgruppen bieten Zugriff auf den DB-Cluster in der VPC. Sie fungieren als Firewall für die zugeordneten DB-Cluster und steuern den ein- und ausgehenden Datenverkehr auf der Cluster-Ebene. DB-Cluster werden standardmäßig mit einer Firewall und einer Standard-Sicherheitsgruppe erstellt, die den Zugriff auf den DB-Cluster verhindert. Sie müssen daher einer Sicherheitsgruppe Regeln hinzufügen, die es Ihnen ermöglichen, eine Verbindung zu Ihrem DB-Cluster herzustellen. Verwenden Sie die Netzwerk- und Konfigurationsinformationen, die Sie im vorherigen Schritt festgelegt haben, um Regeln für den Zugriff auf Ihren DB-Cluster festzulegen.

Wenn beispielsweise eine Anwendung auf eine Datenbank in Ihrem DB-Cluster in einer VPC zugreifen soll, müssen Sie eine benutzerdefinierte TCP-Regel hinzufügen, die den Portbereich und IP-Adressen angibt, mit denen die Anwendung auf die Datenbank zugreift. Wenn Sie eine Anwendung auf einer EC2 Amazon-Instance haben, können Sie die VPC-Sicherheitsgruppe verwenden, die Sie für die EC2 Amazon-Instance eingerichtet haben.

Sie können die Konnektivität zwischen einer EC2 Amazon-Instance und einem DB-Cluster konfigurieren, wenn Sie den DB-Cluster erstellen. Weitere Informationen finden Sie unter Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren.

> **Tip**
>
> Sie können die Netzwerkkonnektivität zwischen einer EC2 Amazon-Instance und einem DB-Cluster automatisch einrichten, wenn Sie den DB-Cluster erstellen. Weitere Informationen finden Sie unter Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren.

Informationen dazu, wie Connect Ressourcen in Amazon Lightsail mit Ihren DB-Clustern verbinden, finden Sie unter Lightsail-Ressourcen mit VPC-Peering verbinden. AWS-Services

Weitere Informationen zum Erstellen einer VPC zur Verwendung mit Aurora finden Sie unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4). Informationen zu gängigen Szenarien für den Zugriff auf eine DB-Instance finden Sie unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC.

So erstellen Sie eine VPC-Sicherheitsgruppe

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon VPC-Konsole unter https://console.aws.amazon.com/vpc.

   > **Note**
   >
   > Stellen Sie sicher, dass Sie sich in der VPC-Konsole befinden, nicht in der RDS-Konsole.

2. Wählen Sie in der oberen rechten Ecke von die AWS Region aus AWS-Managementkonsole, in der Sie Ihre VPC-Sicherheitsgruppe und Ihren DB-Cluster erstellen möchten. Die Liste der Amazon-VPC-Ressourcen für diese AWS -Region sollte zeigen, dass Sie über mindestens eine VPC und mehrere Subnetze verfügen. Wenn Sie dies nicht tun, haben Sie in dieser AWS Region keine Standard-VPC.

3. Wählen Sie im Navigationsbereich Sicherheitsgruppen aus.
4. Wählen Sie Create security group (Sicherheitsgruppe erstellen) aus.

   Die Seite Sicherheitsgruppe erstellen wird angezeigt.
5. Geben Sie im Feld Grundlegende Details den Namen der Sicherheitsgruppe und die Beschreibung ein. Wählen Sie unter VPC die VPC aus, in der Sie Ihren DB-Cluster erstellen möchten.
6. Wählen Sie in Eingehende Regeln die Option Regel hinzufügen.

   a. Wählen Sie für Type (Typ) die Option Custom TCP (Benutzerdefiniertes TCP) aus.
   b. Geben Sie für Portbereich den Portwert ein, der für Ihren DB-Cluster verwendet werden soll.
   c. Wählen Sie für Source (Quelle) den Namen einer Sicherheitsgruppe oder geben Sie den IP-Adressbereich (CIDR-Wert) ein, von dem aus Sie auf den DB-Cluster zugreifen. Wenn Sie Meine IP auswählen, ermöglicht dies den Zugriff auf den DB-Cluster von der in Ihrem Browser erkannten IP-Adresse.
7. Wenn Sie weitere IP-Adressen oder andere Portbereiche hinzufügen müssen, wählen Sie Regel hinzufügen und geben Sie die Informationen für die Regel ein.
8. (Optional) Fügen Sie inRegeln für ausgehenden Datenverkehr Regeln für ausgehenden Datenverkehr hinzu. Standardmäßig ist der gesamte ausgehende Datenverkehr zugelassen.
9. Wählen Sie Sicherheitsgruppe erstellen aus.

Sie können die soeben erstellte VPC-Sicherheitsgruppe als Sicherheitsgruppe beim Anlegen Ihres DB-Clusters verwenden.

> **Note**
>
> Wenn Sie eine Standard-VPC verwenden, wird eine Standard-Subnetzgruppe für Sie angelegt, die alle Subnetze der VPC umfasst. Wenn Sie einen DB-Cluster erstellen, können Sie die Standard-VPC auswählen und Standard für die DB-Subnetzgruppe verwenden.

Nachdem Sie die Setup-Anforderungen erfüllt haben, können Sie einen DB-Cluster mit Ihren Anforderungen und Ihrer Sicherheitsgruppe erstellen, indem Sie den Anweisungen in Erstellen eines Amazon Aurora-DB Clusters. Informationen zu den ersten Schritten beim Erstellen eines DB-Clusters, der eine bestimmte DB-Engine verwendet, finden Sie unter Erste Schritte mit Amazon Aurora.

Erste Schritte mit Amazon Aurora

In diesem Abschnitt finden Sie heraus, wie Sie mit Amazon RDS einen Aurora-DB-Cluster erstellen und eine Verbindung mit diesem herstellen.

Bei den folgenden Prozeduren handelt es sich um Tutorials, die die Grundlagen der ersten Schritte mit Aurora demonstrieren. In den späteren Abschnitten werden erweiterte Aurora-Konzepte und -Verfahren vorgestellt, z. B. die verschiedenen Endpunkte und die Skalierung von Aurora-Clustern.

> **Important**
>
> Sie müssen die Aufgaben im Abschnitt Einrichten Ihrer Umgebung für Amazon Aurora abschließen, um einen DB-Cluster erstellen oder eine Verbindung mit einer DB-Instance herstellen zu können.

Themen

- Erstellen und Verbinden mit einem DB-Cluster von Aurora MySQL
- Erstellen eines DB-Clusters von Aurora PostgreSQL und Herstellen einer Verbindung
- Tutorial: Erstellen eines Webservers und einer eines Amazon Aurora-DB-Clusters

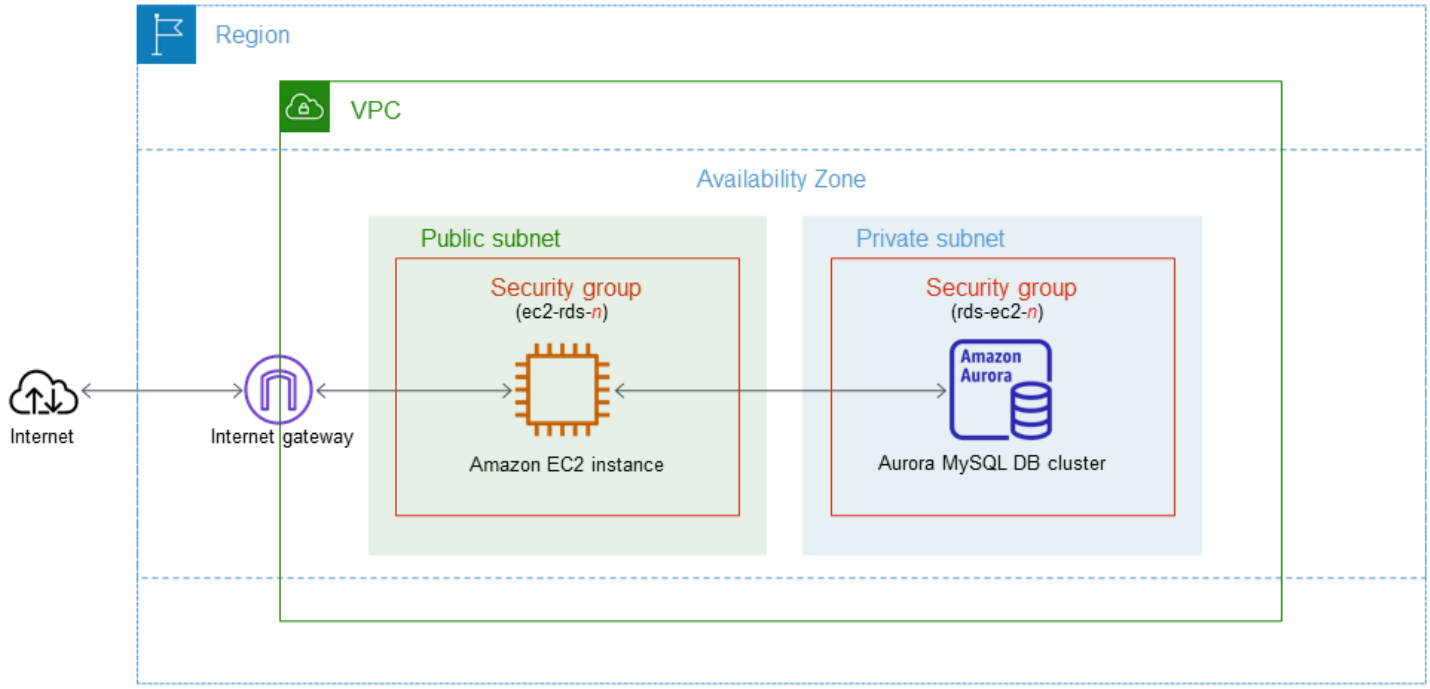
Erstellen und Verbinden mit einem DB-Cluster von Aurora MySQL

In diesem Tutorial werden eine EC2 Instance und ein Aurora MySQL-DB-Cluster erstellt. Das Tutorial zeigt Ihnen, wie Sie mit einem Standard-MySQL-Client von der EC2 Instance aus auf den DB-Cluster zugreifen. Als bewährte Methode erstellt dieses Tutorial einen privaten DB-Cluster in einer Virtual Private Cloud (VPC). In den meisten Fällen können andere Ressourcen in derselben VPC, z. B. EC2 Instances, auf den DB-Cluster zugreifen, aber Ressourcen außerhalb der VPC können nicht darauf zugreifen.

Nach Abschluss des Tutorials gibt es in jeder Availability Zone im VPC ein öffentliches und ein privates Subnetz. In einer Availability Zone befindet sich die EC2 Instance im öffentlichen Subnetz und die DB-Instance im privaten Subnetz.

> **Important**
>
> Für die Erstellung eines Kontos fallen keine Gebühren an AWS . Wenn Sie dieses Tutorial abschließen, können Ihnen jedoch Kosten für die AWS Ressourcen entstehen, die Sie verwenden. Sie können diese Ressourcen nach Abschluss des Tutorials löschen, wenn sie nicht mehr benötigt werden.

Das folgende Diagramm zeigt die Konfiguration nach Abschluss des Tutorials.



Dieses Tutorial ermöglicht es Ihnen, Ihre Ressourcen auf eine der folgenden Weisen zu erstellen:

1. Verwenden Sie das AWS-Managementkonsole - Schritt 1: Eine EC2 Instanz erstellen und Schritt 2:Erstellen eines DB-Clusters von Aurora MySQL
2. Verwenden Sie CloudFormation , um die Datenbankinstanz und die EC2 Instanz zu erstellen - (Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora MySQL-Cluster mit CloudFormation

Die erste Methode verwendet Einfache Erstellung, um einen privaten DB-Cluster von Aurora MySQL mit der AWS-Managementkonsole zu erstellen. Geben Sie hier nur den DB-Engine-Typ, die DB-Instance-Größe und die DB-Cluster-Kennung an. Einfache Erstellung verwendet für die anderen Konfigurationsoptionen die Standardeinstellung.

Wenn Sie stattdessen Standarderstellung verwenden, können Sie beim Erstellen eines DB-Clusters weitere Konfigurationsoptionen wählen. Zu diesen Optionen gehören Einstellungen für Verfügbarkeit, Sicherheit, Backups und Wartung. Wenn Sie einen öffentliche DB-Cluster erstellen möchten, müssen Sie Standarderstellung verwenden. Weitere Informationen finden Sie unter the section called “Erstellen eines DB-Clusters”.

Themen

- Voraussetzungen
- Schritt 1: Eine EC2 Instanz erstellen
- Schritt 2:Erstellen eines DB-Clusters von Aurora MySQL
- (Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora MySQL-Cluster mit CloudFormation
- Schritt 3: Herstellen einer Verbindung mit einem DB-Cluster von Aurora MySQL
- Schritt 4: Löschen Sie die EC2 Instance und den DB-Cluster
- (Optional) Löschen Sie die EC2 Instance und den DB-Cluster, die mit erstellt wurden CloudFormation
- (Optional) Verbinden Sie Ihren DB-Cluster mit einer Lambda-Funktion

Voraussetzungen

Bevor Sie die Schritte in diesem Abschnitt abschließen, stellen Sie sicher, dass Sie folgende Voraussetzungen erfüllen:

- Melden Sie sich an für ein AWS-Konto
- Erstellen eines Benutzers mit Administratorzugriff

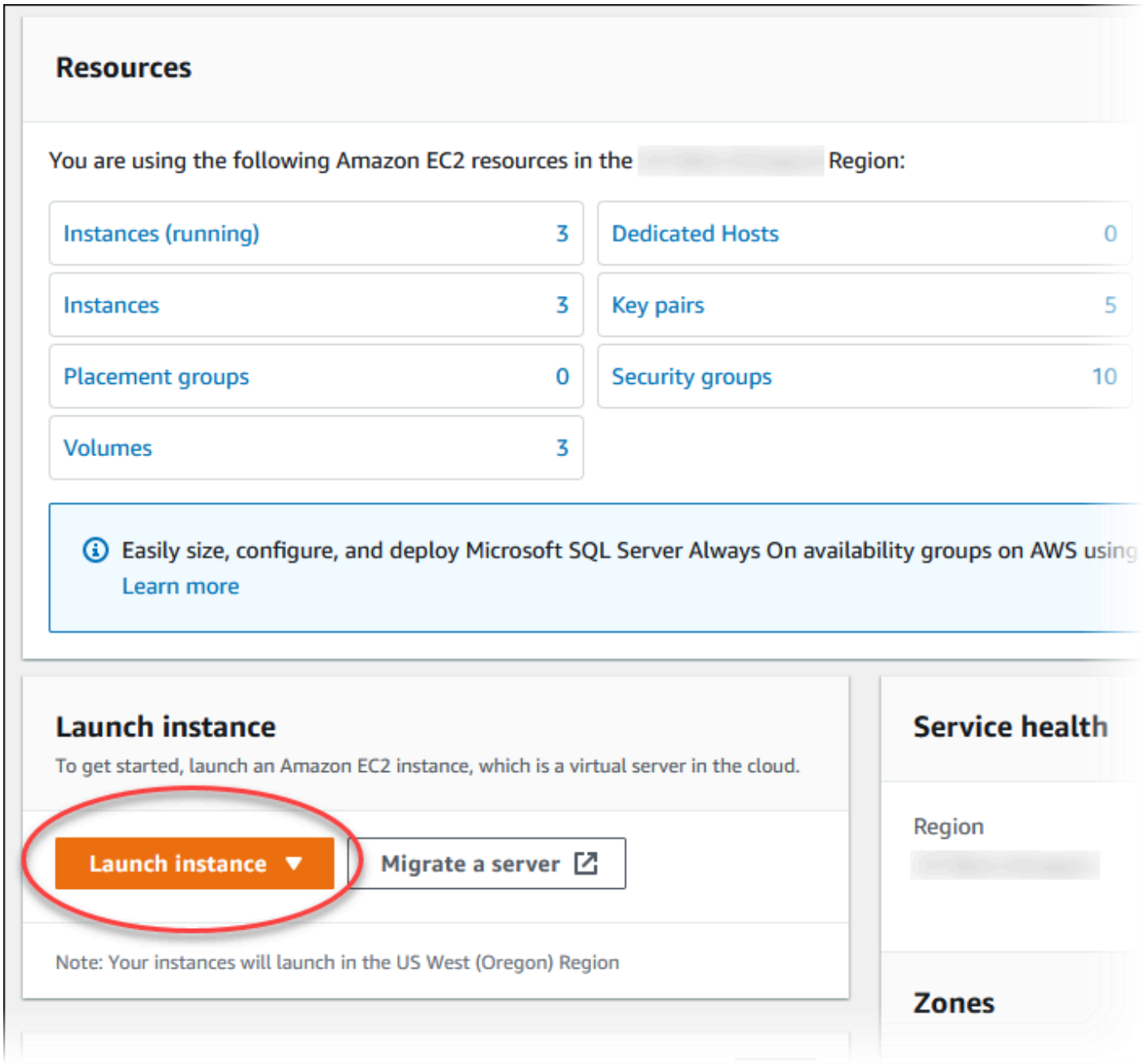
Schritt 1: Eine EC2 Instanz erstellen

Erstellen Sie eine EC2 Amazon-Instance, mit der Sie eine Verbindung zu Ihrer Datenbank herstellen.

Um eine EC2 Instanz zu erstellen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die EC2 Amazon-Konsole unter https://console.aws.amazon.com/ec2/.

2. Wählen Sie in der oberen rechten Ecke von die aus AWS-Managementkonsole, AWS-Region in der Sie die Instance erstellen möchten. EC2
3. Wählen Sie EC2 Dashboard und dann Launch instance aus, wie in der folgenden Abbildung gezeigt.

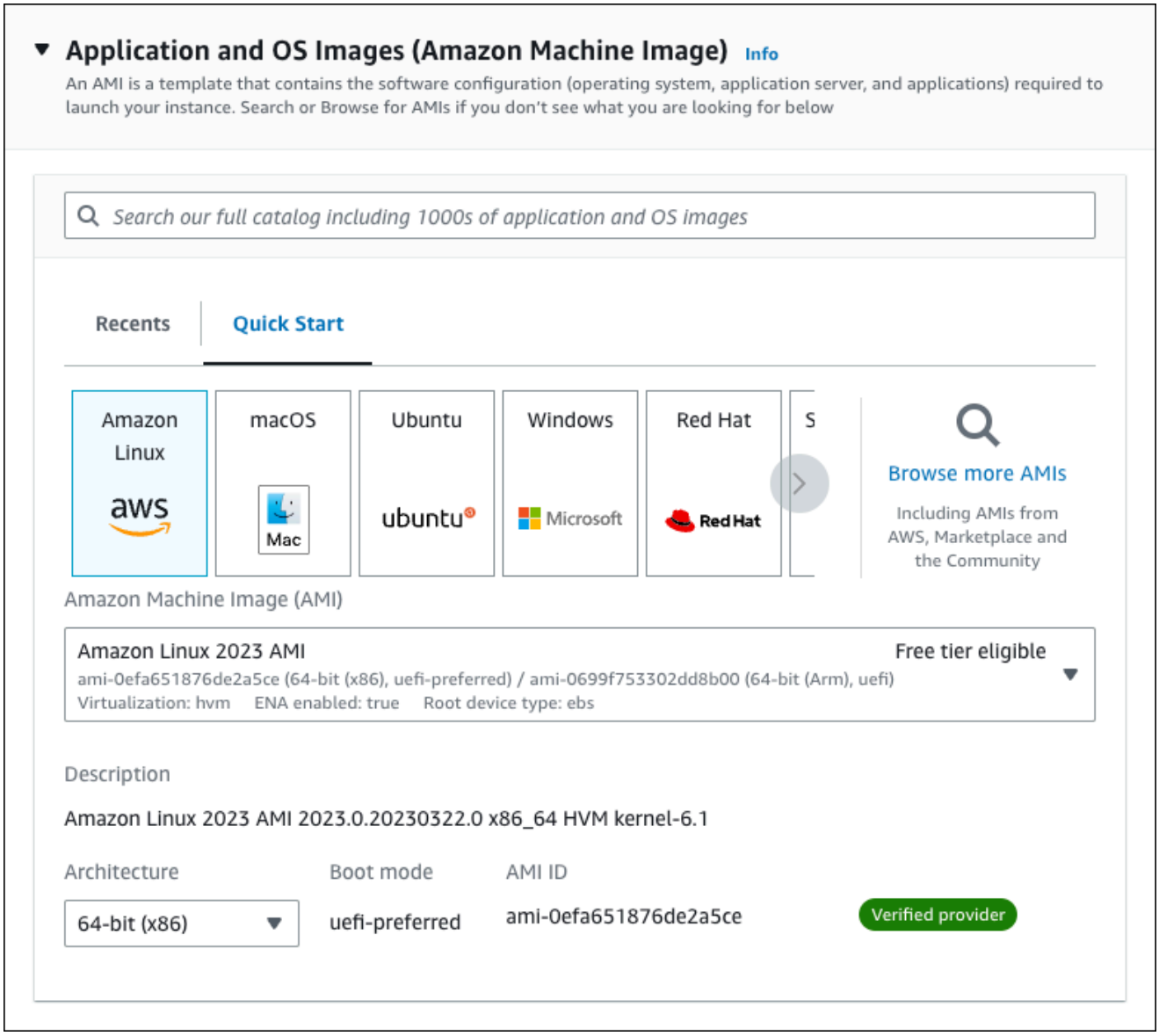


Die Seite Eine Instance starten wird geöffnet.

4. Wählen Sie auf der Seite Eine Instance starten die folgenden Einstellungen aus.

   a. Geben Sie unter Name and tags (Name und Tags) als Name den Namen **`ec2-database-connect`** ein.

b. Wählen Sie unter Anwendungs- und Betriebssystem-Images (Amazon Machine Image) die Option Amazon Linux und dann die Registerkarte Amazon Linux 2023 AMI aus. Übernehmen Sie für alle anderen Einstellungen die Standardwerte.



c. Wählen Sie unter Instance type (Instance-Typ) den Wert t2.micro aus.

d. Wählen Sie unter Key pair (login) (Schlüsselpaar (Anmeldung)) einen Key pair name (Schlüsselpaarname), um ein vorhandenes Schlüsselpaar zu verwenden. Um ein neues key pair für die EC2 Amazon-Instance zu erstellen, wählen Sie Neues key pair erstellen und verwenden Sie dann das Fenster key pair erstellen, um es zu erstellen.

   Weitere Informationen zum Erstellen eines neuen key pair finden Sie unter Erstellen eines key pair im EC2 Amazon-Benutzerhandbuch.

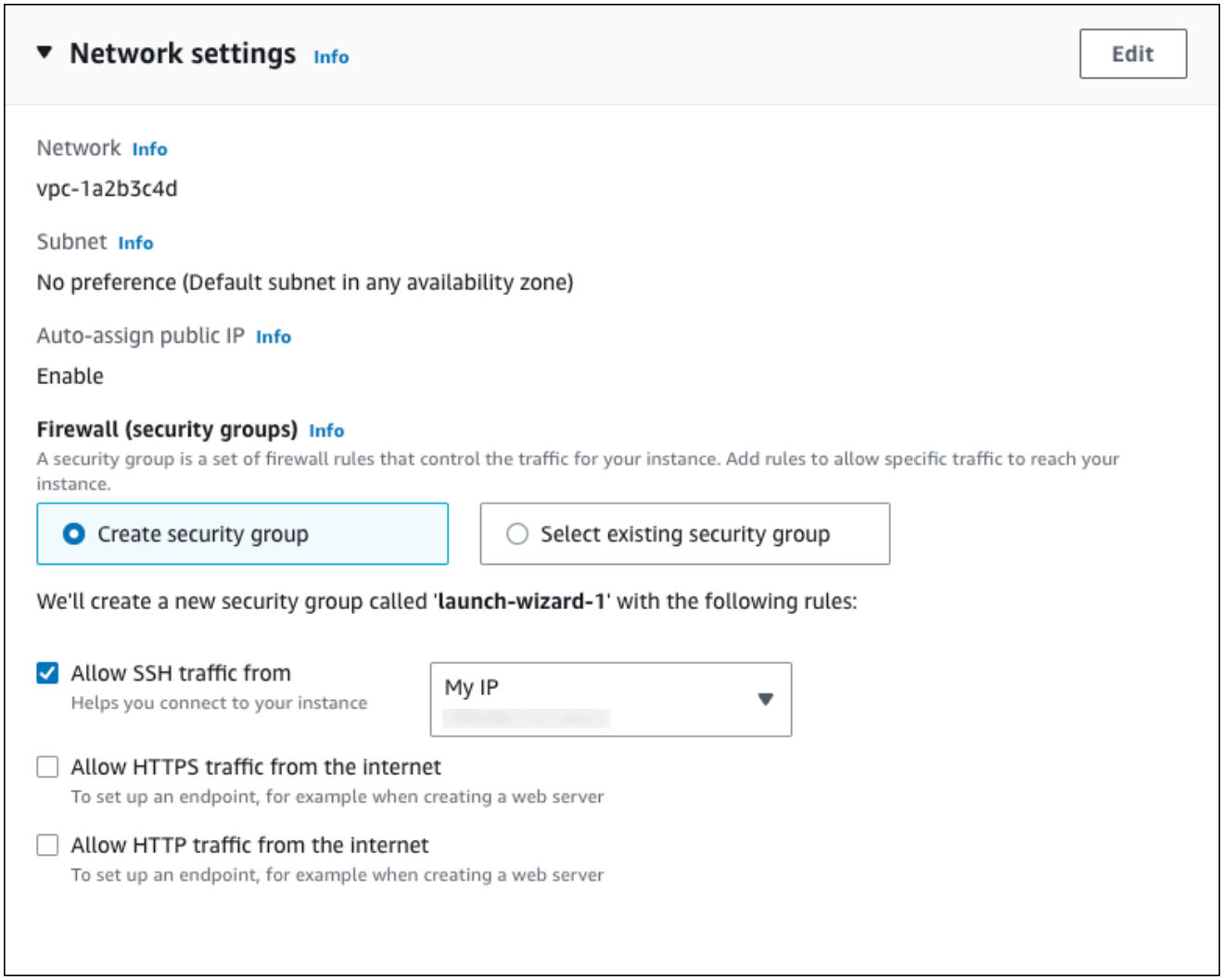
e. Wählen Sie unter „SSH-Verkehr zulassen“ in den Netzwerkeinstellungen die Quelle der SSH-Verbindungen zur Instance aus. EC2

   Sie können My IP (Meine IP) auswählen, wenn die angezeigte IP-Adresse für SSH-Verbindungen korrekt ist. Andernfalls können Sie mithilfe von Secure Shell (SSH) die IP-Adresse bestimmen, die für die Verbindung zu EC2 Instances in Ihrer VPC verwendet werden soll. Um Ihre öffentliche IP-Adresse in einem anderen Browserfenster oder einer anderen Registerkarte zu ermitteln, können Sie den Dienst unter verwenden. https://checkip.amazonaws.com Ein Beispiel für eine IP-Adresse ist 192.0.2.1/32.

   In vielen Fällen können Sie eine Verbindung über einen Internetdienstanbieter (ISP) oder hinter Ihrer Firewall ohne statische IP-Adresse herstellen. Bestimmen Sie in diesem Fall den Bereich der IP-Adressen, die von Client-Computern verwendet werden.

   > **Warning**
   >
   > Wenn Sie `0.0.0.0/0` den SSH-Zugriff verwenden, ermöglichen Sie allen IP-Adressen den Zugriff auf Ihre öffentlichen EC2 Instanzen über SSH. Dieser Ansatz ist zwar für kurze Zeit in einer Testumgebung zulässig, aber für Produktionsumgebungen sehr unsicher. Autorisieren Sie in der Produktion nur eine bestimmte IP-Adresse oder einen bestimmten Adressbereich für den Zugriff auf Ihre EC2 Instances über SSH.

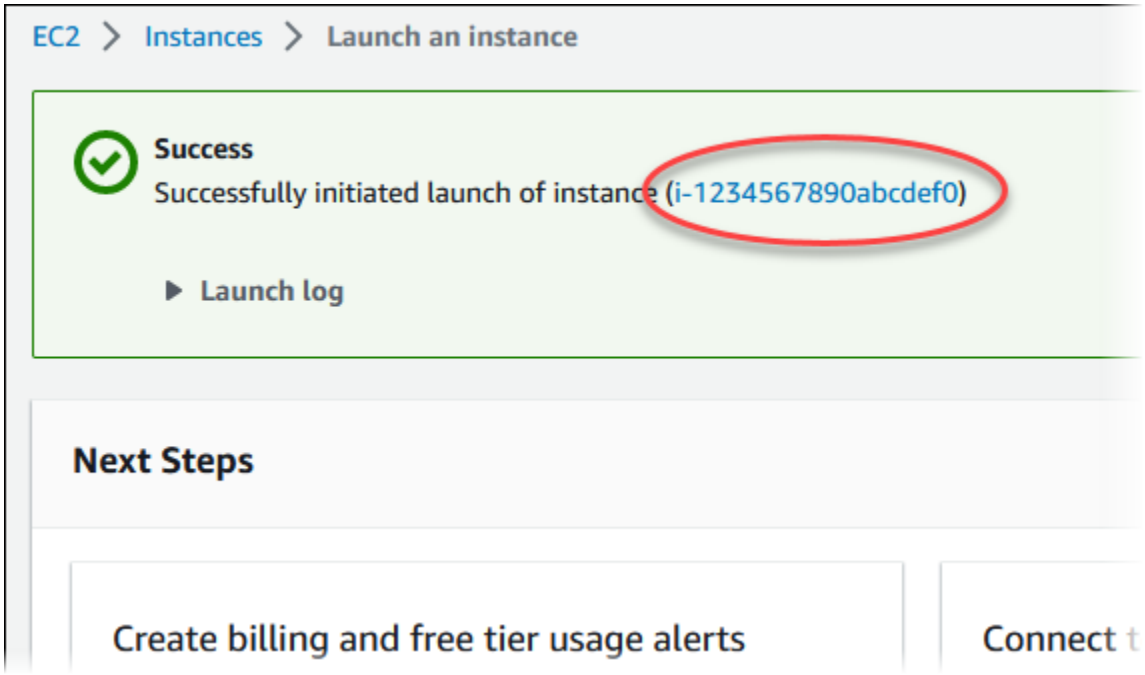
   Die folgende Abbildung zeigt ein Beispiel für den Bereich Netzwerkeinstellungen.



f. Übernehmen Sie die Standardwerte für die übrigen Abschnitte.

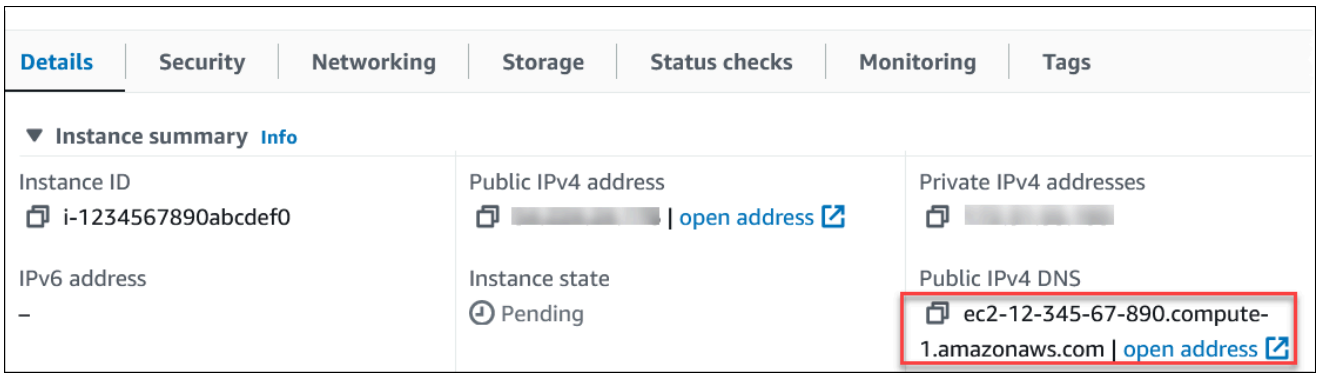
g. Sehen Sie sich im Bereich Zusammenfassung eine Zusammenfassung Ihrer EC2 Instance-Konfiguration an. Wenn Sie bereit sind, wählen Sie Launch instance aus.

5. Notieren Sie sich auf der Seite Startstatus die ID für Ihre neue EC2 Instance, zum Beispiel:`i-1234567890abcdef0`.

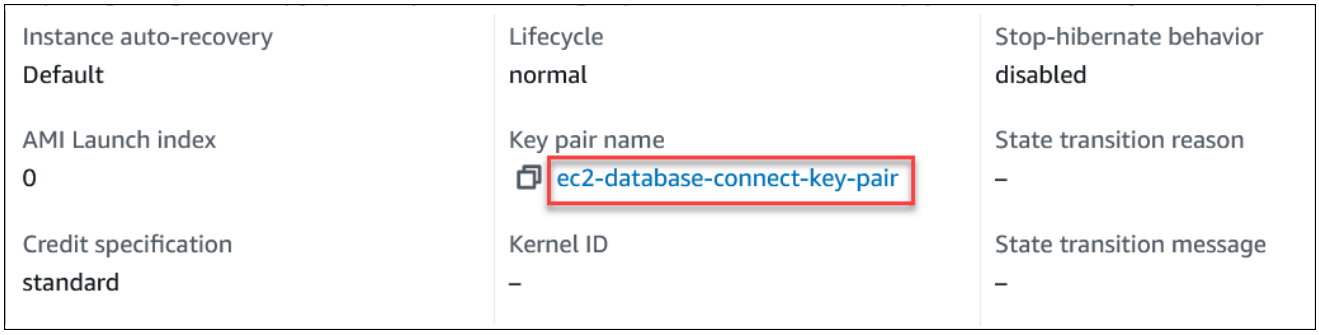


6. Wählen Sie die EC2 Instance-ID, um die Liste der EC2 Instances zu öffnen, und wählen Sie dann Ihre EC2 Instance aus.
7. Notieren Sie sich die folgenden Werte auf der Registerkarte Details. Diese benötigen Sie, wenn Sie eine Verbindung über SSH herstellen:

   a. Notieren Sie sich in der Instanzübersicht den Wert für Public IPv4 DNS.



   b. Notieren Sie sich unter Instance-Details den Wert für Schlüsselpaarname.



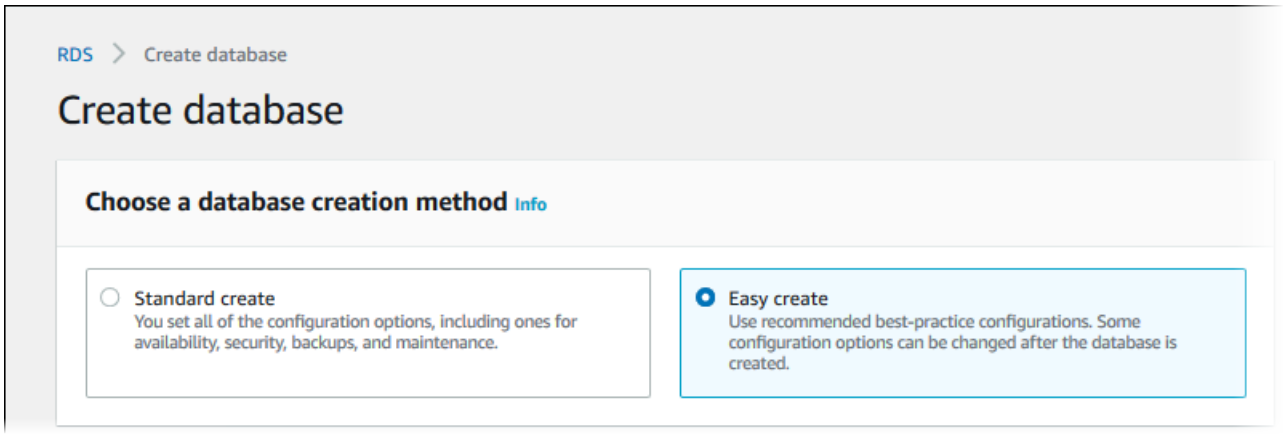
8. Warten Sie, bis der Instanzstatus Ihrer EC2 Instance den Status Running hat, bevor Sie fortfahren.

Schritt 2:Erstellen eines DB-Clusters von Aurora MySQL

In diesem Beispiel verwenden Sie Einfache Erstellung, um einen DB-Cluster von Aurora MySQL mit einer DB-Instance-Klasse vom Typ „db.r6g.large“ zu erstellen.

So erstellen Sie einen DB-Cluster von Aurora MySQL mit der Option „Einfache Erstellung“

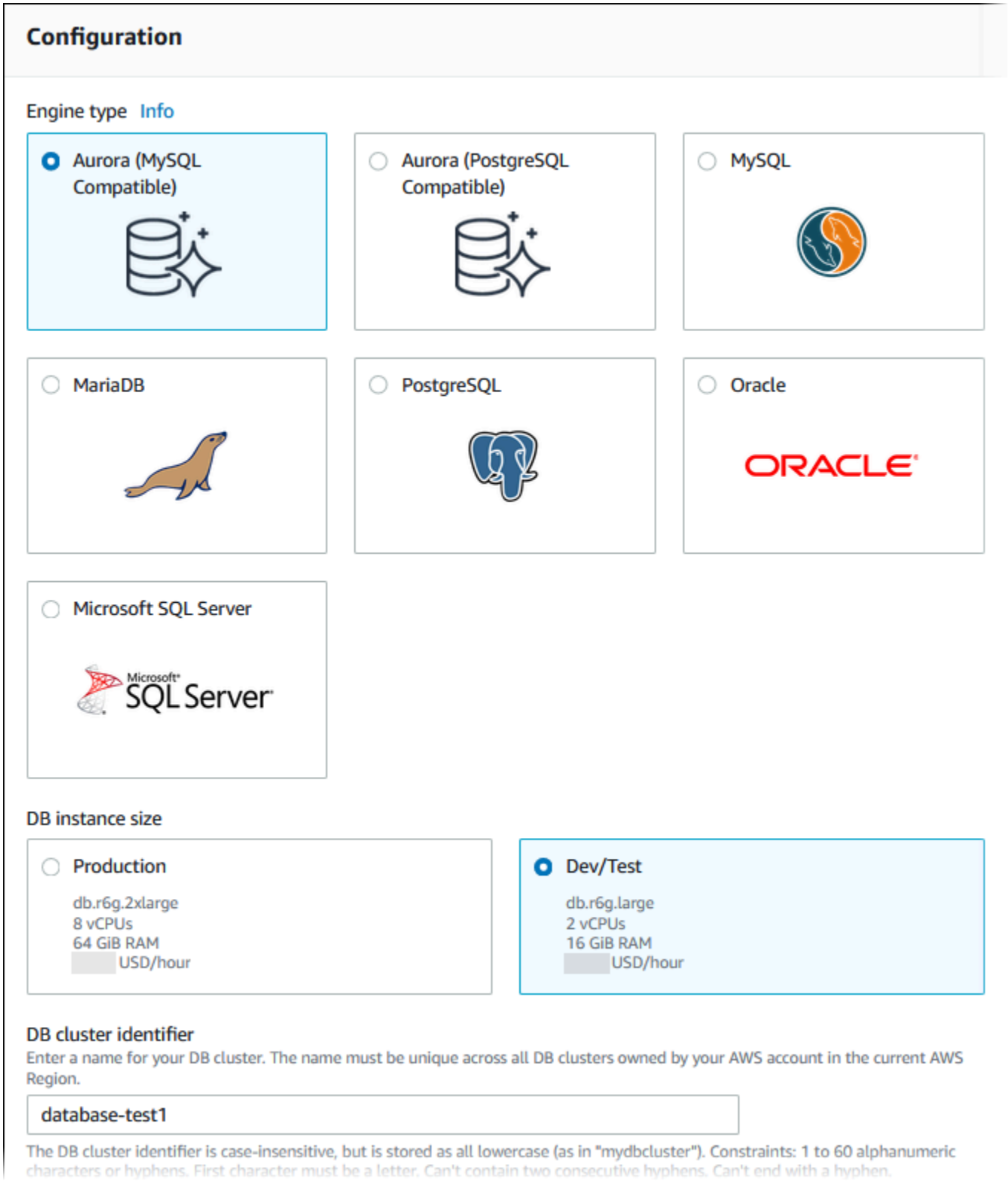
1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der oberen rechten Ecke der Amazon RDS-Konsole den aus, AWS-Region in dem Sie den DB-Cluster erstellen möchten.
3. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
4. Wählen Sie Datenbank erstellen aus und vergewissern Sie sich, dass Einfache Erstellung ausgewählt ist.



5. Wählen Sie unter Konfiguration die Option Aurora (MySQL-kompatibel) als Engine-Typ aus.

6. Wählen Sie in DB instance size (DB-Instance-Größe) die Option Dev/Test (Entwicklung/Testen) aus.
7. Geben Sie unter DB-Cluster-Kennung die Zeichenfolge **`database-test1`** ein.

   Die Seite Datenbank erstellen sollte ähnlich wie in der folgenden Abbildung gezeigt aussehen.



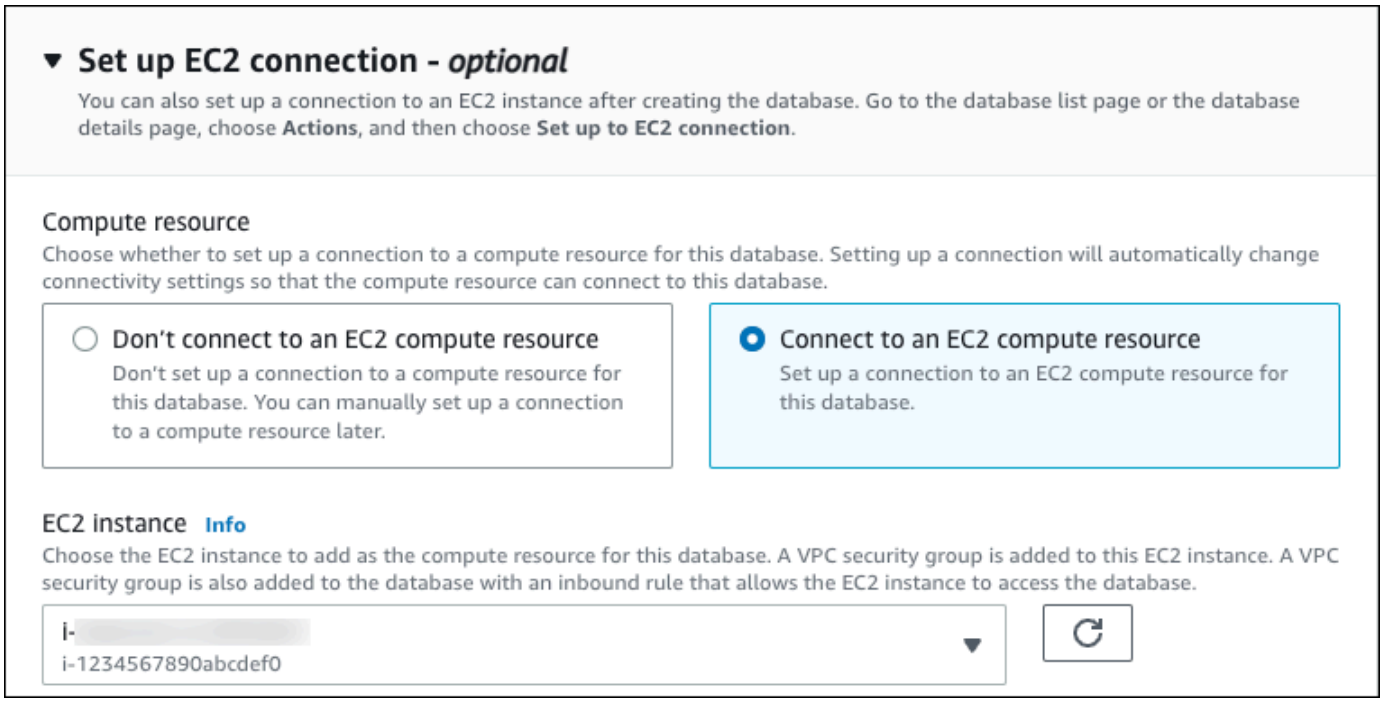
8. Geben Sie unter Hauptbenutzername einen Namen für den Hauptbenutzer ein oder behalten Sie den Standardnamen bei.

9. Um für den DB-Cluster ein automatisch generiertes Hauptpasswort zu verwenden, wählen Sie Passwort automatisch generieren aus.

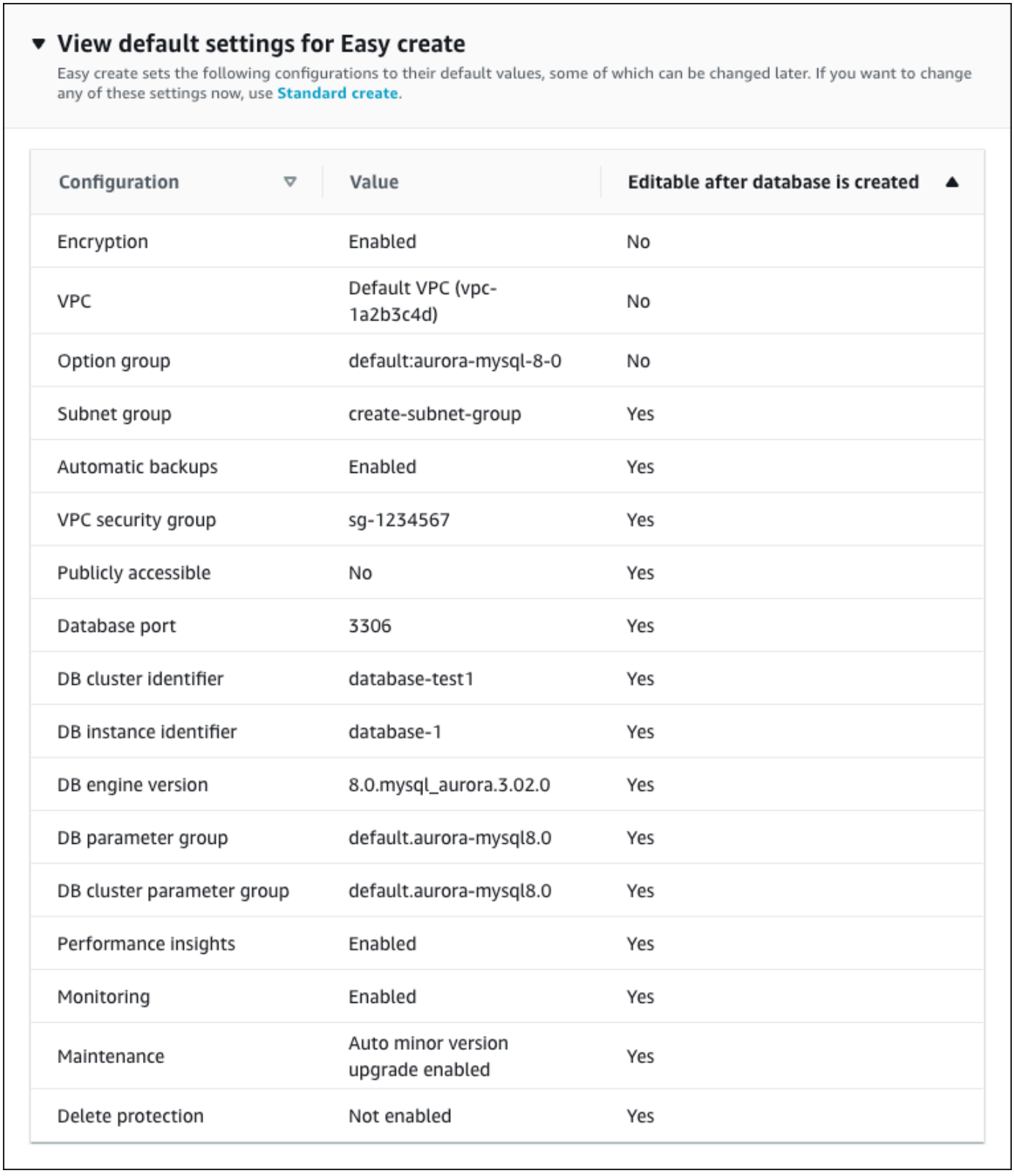
   Um das Hauptpasswort einzugeben, deaktivieren Sie das Kontrollkästchen Passwort automatisch generieren und geben Sie anschließend dasselbe Passwort in Hauptpasswort und Passwort bestätigen ein.

10. Um eine Verbindung mit der EC2 Instance einzurichten, die Sie zuvor erstellt haben, öffnen Sie EC2 Verbindung einrichten — optional.

    Wählen Sie Mit einer EC2 Rechenressource verbinden aus. Wählen Sie die EC2 Instanz aus, die Sie zuvor erstellt haben.



11. (Optional) Öffnen Sie Anzeigen von Standardeinstellungen für eine einfache Erstellung.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-mysql-8-0	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	3306	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	8.0.mysql_aurora.3.02.0	Yes
DB parameter group	default.aurora-mysql8.0	Yes
DB cluster parameter group	default.aurora-mysql8.0	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

Sie können die Standardeinstellungen von Einfache Erstellung einsehen. Die Spalte Nach Erstellung der Datenbank editierbar zeigt, welche Optionen Sie nach der Datenbankerstellung ändern können.

- Wenn für eine Einstellung Nein in dieser Spalte steht und Sie eine andere Einstellung verwenden möchten, können Sie Standarderstellung verwenden, um den DB-Cluster zu erstellen.
- Wenn für eine Einstellung Ja in dieser Spalte steht und Sie eine andere Einstellung verwenden möchten, können Sie entweder Standarderstellung auswählen, um den DB-Cluster zu erstellen, oder den DB-Cluster nach der Erstellung ändern, um die Einstellung zu ändern.

12. Wählen Sie Datenbank erstellen aus.

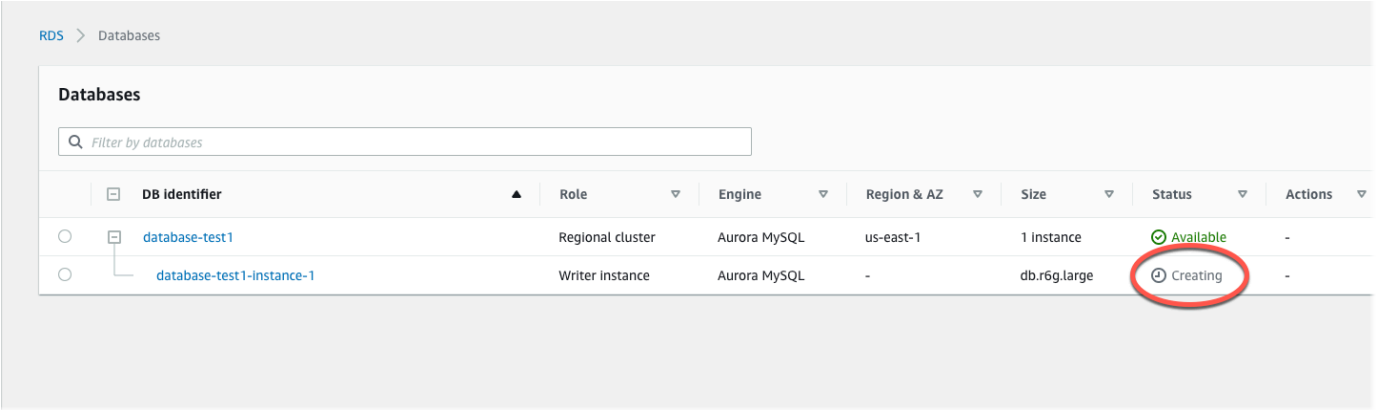
    Um den Hauptbenutzernamen und das Passwort für den DB-Cluster anzuzeigen, wählen Sie Anmeldeinformationen anzeigen aus.

    Sie können den angezeigten Benutzernamen und das angezeigte Passwort verwenden, um als Hauptbenutzer eine Verbindung mit dem DB-Cluster herzustellen.

    > **Important**
    >
    > Sie können dieses Passwort für den Hauptbenutzer nicht erneut anzeigen. Wenn Sie es nicht notieren, müssen Sie es möglicherweise ändern.
    > Wenn Sie das Passwort für den Hauptbenutzer ändern müssen, nachdem der DB-Cluster verfügbar wurde, können Sie den DB-Cluster entsprechend ändern. Weitere Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

13. Wählen Sie in der Liste Datenbanken den Namen des neuen DB-Clusters von Aurora MySQL aus.

    Die Writer-Instance hat den Status Wird erstellt, bis der DB-Cluster bereit ist und verwendet werden kann.



Wenn sich der Status der Writer-Instance in Verfügbar ändert, können Sie die Verbindung mit dem DB-Cluster herstellen. Je nach Klasse und Speicherort der DB-Instance kann es bis zu 20 Minuten dauern, bis der neue DB-Cluster verfügbar ist.

(Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora MySQL-Cluster mit CloudFormation

Anstatt die Konsole zur Erstellung Ihrer VPC, EC2 Instance und Ihres Aurora MySQL-DB-Clusters CloudFormation zu verwenden, können Sie AWS Ressourcen bereitstellen, indem Sie die Infrastruktur als Code behandeln. Um Ihnen zu helfen, Ihre AWS Ressourcen in kleinere und besser verwaltbare Einheiten zu organisieren, können Sie die CloudFormation Nested-Stack-Funktionalität verwenden. Weitere Informationen finden Sie unter Einen Stack auf der CloudFormation Konsole erstellen und Mit verschachtelten Stacks arbeiten.

> **Important**
>
> CloudFormation ist kostenlos, aber die Ressourcen, die CloudFormation erstellt werden, sind live. Es fallen die standardmäßigen Nutzungsgebühren für diese Ressourcen an, bis sie von Ihnen beendet werden. Weitere Informationen dazu finden Sie unter Amazon Aurora – Preise.

Gehen Sie wie folgt vor, um Ihre Ressourcen mithilfe der CloudFormation Konsole zu erstellen:

- Schritt 1: Laden Sie die CloudFormation Vorlage herunter
- Schritt 2: Konfigurieren Sie Ihre Ressourcen mit CloudFormation

Laden Sie die CloudFormation Vorlage herunter

Eine CloudFormation Vorlage ist eine JSON- oder YAML-Textdatei, die die Konfigurationsinformationen zu den Ressourcen enthält, die Sie im Stack erstellen möchten. Diese Vorlage erstellt neben dem Aurora-Cluster auch eine VPC und einen Bastion-Host für Sie.

Um die Vorlagendatei herunterzuladen, öffnen Sie den folgenden Link: Aurora MySQL CloudFormation template.

Klicken Sie auf der Github-Seite auf die Schaltfläche Rohdatei herunterladen, um die YAML-Vorlagendatei zu speichern.

Konfigurieren Sie Ihre Ressourcen mit CloudFormation

> **Note**
>
> Bevor Sie diesen Vorgang starten, stellen Sie sicher, dass Sie ein Schlüsselpaar für eine EC2 Instance in Ihrem haben AWS-Konto. Weitere Informationen finden Sie unter EC2 Amazon-Schlüsselpaare und Linux-Instances.

Wenn Sie die CloudFormation Vorlage verwenden, müssen Sie die richtigen Parameter auswählen, um sicherzustellen, dass Ihre Ressourcen ordnungsgemäß erstellt werden. Führen Sie die folgenden Schritte aus:

1. Melden Sie sich bei https://console.aws.amazon.com/cloudformation an AWS-Managementkonsole und öffnen Sie die CloudFormation Konsole.
2. Wählen Sie Stack erstellen aus.
3. Wählen Sie im Abschnitt „Vorlage angeben“ die Option Vorlagendatei von Ihrem Computer hochladen und dann Weiter aus.
4. Legen Sie auf der Seite Stack-Details angeben die folgenden Parameter fest:
   a. Setzen Sie den Stack-Namen auf AurMy SQLTest Stack.
   b. Legen Sie unter Parameter Availability Zones fest, indem Sie zwei Availability Zones auswählen.
   c. Wählen Sie unter Linux Bastion Host configuration für Key Name ein key pair aus, um sich bei Ihrer EC2 Instance anzumelden.
   d. Stellen Sie in den Bastion-Host-Konfigurationseinstellungen in Linux den zulässigen IP-Bereich auf Ihre IP-Adresse ein. Um mithilfe von Secure Shell (SSH) eine Verbindung zu EC2 Instances

in Ihrer VPC herzustellen, ermitteln Sie Ihre öffentliche IP-Adresse mithilfe des Dienstes unter. https://checkip.amazonaws.com Ein Beispiel für eine IP-Adresse ist 192.0.2.1/32.

> **Warning**
>
> Wenn Sie `0.0.0.0/0` den SSH-Zugriff verwenden, ermöglichen Sie allen IP-Adressen den Zugriff auf Ihre öffentlichen EC2 Instances über SSH. Dieser Ansatz ist zwar für kurze Zeit in einer Testumgebung zulässig, aber für Produktionsumgebungen sehr unsicher. Autorisieren Sie in der Produktion nur eine bestimmte IP-Adresse oder einen bestimmten Adressbereich für den Zugriff auf Ihre EC2 Instances über SSH.

e. Stellen Sie unter Allgemeine Datenbankkonfiguration die Datenbank-Instance-Klasse auf db.r6g.large ein.

f. Setzen Sie den Datenbanknamen auf **`database-test1`**.

g. Geben Sie unter Hauptbenutzername der Datenbank den Namen des Hauptbenutzers ein.

h. Setzen Sie für dieses Tutorial die Option DB-Hauptbenutzerpasswort mit Secrets Manager verwalten auf `false`.

i. Legen Sie für Datenbankpasswort ein Passwort Ihrer Wahl fest. Notieren Sie sich dieses Passwort für weitere Schritte im Tutorial.

j. Setzen Sie Multi-AZ-Bereitstellung auf `false`.

k. Lassen Sie alle anderen Einstellungen auf den Standardwerten. Klicken Sie auf Weiter, um fortzufahren.

5. Behalten Sie auf der Seite Stack-Optionen konfigurieren alle Standardoptionen bei. Klicken Sie auf Weiter, um fortzufahren.
6. Wählen Sie auf der Seite Stack überprüfen die Option Absenden aus, nachdem Sie die Optionen für die Datenbank und den Linux-Bastion-Host überprüft haben.

Sehen Sie sich nach Abschluss der Stack-Erstellung die Stacks mit Namen BastionStackund AMSNS an, um die Informationen zu notieren, die Sie für die Verbindung mit der Datenbank benötigen. Weitere Informationen finden Sie unter CloudFormation Stack-Daten und Ressourcen anzeigen auf der. AWS-Managementkonsole

Schritt 3: Herstellen einer Verbindung mit einem DB-Cluster von Aurora MySQL

Sie können für die Verbindung mit dem DB-Cluster eine beliebige SQL-Client-Standardanwendung verwenden. In diesem Beispiel stellen Sie eine Verbindung mit einem DB-Cluster von Aurora MySQL mithilfe des mysql-Befehlszeilen-Clients her.

So stellen Sie eine Verbindung mit dem DB-Cluster von Aurora MySQL her

1. Suchen Sie nach dem Endpunkt (DNS-Name) und der Portnummer der Writer-Instance für Ihren DB-Cluster.

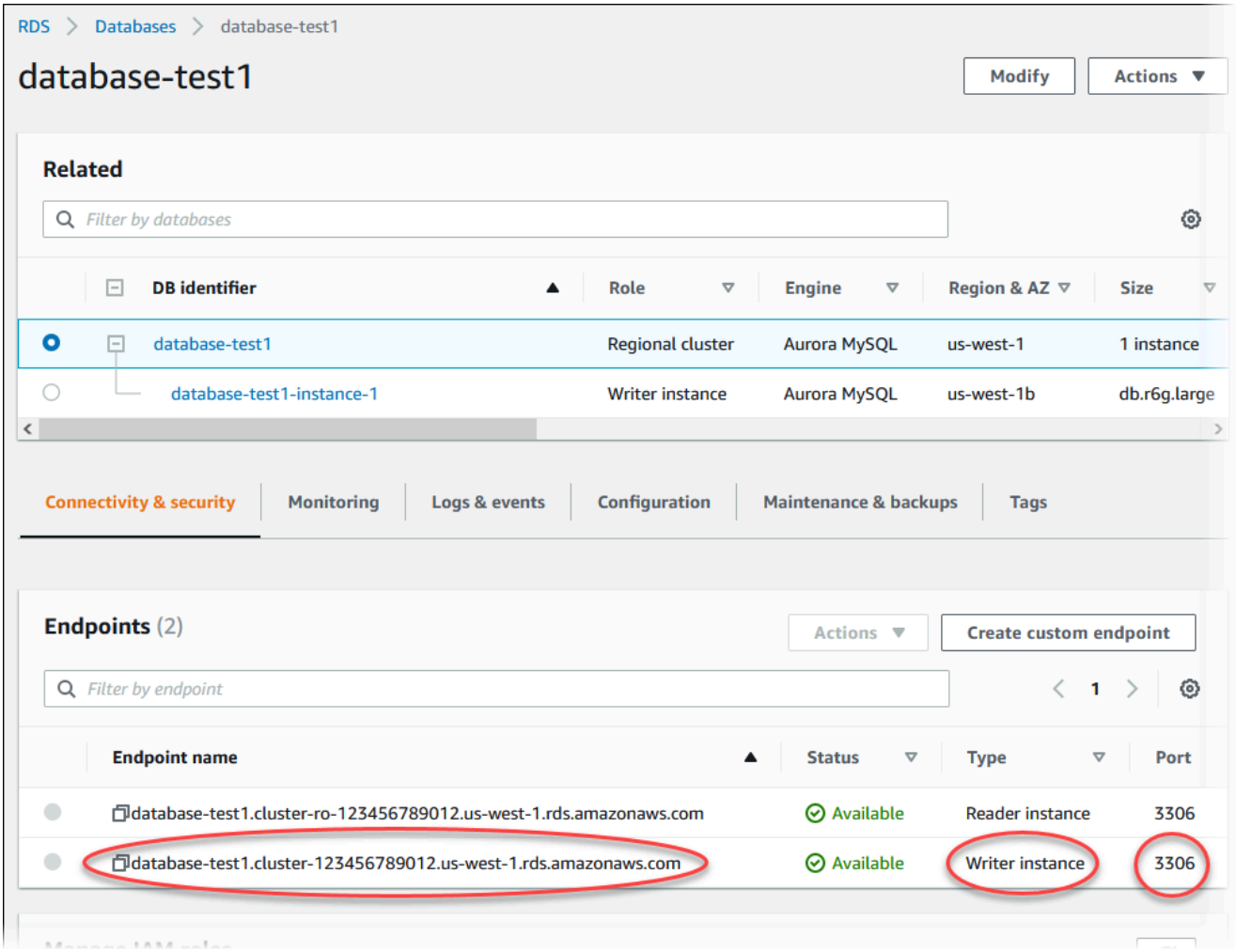
    a. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.

    b. Wählen Sie oben rechts in der Amazon-RDS-Konsole die AWS-Region für den DB-Cluster aus.

    c. Wählen Sie im Navigationsbereich Datenbanken aus.

    d. Wählen Sie den Namen des DB-Clusters von Aurora MySQL aus, um dessen Details anzuzeigen.

    e. Kopieren Sie auf der Registerkarte Konnektivität und Sicherheit den Endpunkt der Writer-Instance. Notieren Sie sich auch die Portnummer. Sie benötigen sowohl den Endpunkt als auch die Portnummer, um die Verbindung mit dem DB-Cluster herzustellen.



2. Stellen Sie eine Verbindung zu der EC2 Instance her, die Sie zuvor erstellt haben, indem Sie den Schritten unter Connect to your Linux Instance im EC2 Amazon-Benutzerhandbuch folgen.

   Wir empfehlen, dass Sie über SSH eine Verbindung zu Ihrer EC2 Instance herstellen. Wenn das SSH-Client-Dienstprogramm unter Windows, Linux oder Mac installiert ist, können Sie mit dem folgenden Befehlsformat eine Verbindung mit der Instance herstellen:

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   Nehmen wir zum Beispiel an, das `ec2-database-connect-key-pair.pem` ist `/dir1` unter Linux gespeichert und das öffentliche IPv4 DNS für Ihre EC2 Instance ist `esec2-12-345-678-90.compute-1.amazonaws.com`. In diesem Fall würde Ihr SSH-Befehl wie folgt aussehen:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Holen Sie sich die neuesten Bugfixes und Sicherheitsupdates, indem Sie die Software auf Ihrer EC2 Instance aktualisieren. Führen Sie dazu den folgenden Befehl aus.

   > **Note**
   >
   > Mit der Option `-y` werden die Updates installiert, ohne um Bestätigung zu bitten. Um Updates vor der Installation zu überprüfen, lassen Sie diese Option aus.

   ```
   sudo dnf update -y
   ```

4. Führen Sie den folgenden Befehl aus, um den mysql-Befehlszeilen-Client von MariaDB auf Amazon Linux 2023 zu installieren:

   ```
   sudo dnf install mariadb105
   ```

5. Stellen Sie eine Verbindung mit dem DB-Cluster von Aurora MySQL her. Geben Sie z. B. den folgenden Befehl ein. Mit dieser Aktion können Sie eine Verbindung mit dem DB-Cluster von Aurora MySQL mithilfe des MySQL-Clients herstellen.

   Ersetzen Sie den Endpunkt der Writer-Instance für *`endpoint`* und den Hauptbenutzernamen, den Sie für *`admin`* verwendet haben. Geben Sie das Master-Passwort ein, das Sie bei der Aufforderung zur Eingabe eines Passworts verwendet haben.

   ```
   mysql -h endpoint -P 3306 -u admin -p
   ```

   Nachdem Sie das Passwort für den Benutzer eingegeben haben, sollte eine Ausgabe wie die folgende angezeigt werden.

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 217
   Server version: 8.0.23 Source distribution

   Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
   ```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Weitere Informationen zum Herstellen einer Verbindung mit einem DB-Cluster von Aurora MySQL finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora MySQL-DB-Cluster. Wenn Sie sich nicht mit Ihrem DB-Cluster verbinden können, erhalten Sie unter Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden Hilfe.

Aus Sicherheitsgründen empfiehlt es sich, verschlüsselte Verbindungen zu verwenden. Verwenden Sie eine unverschlüsselte MySQL Verbindung nur, wenn sich Client und Server in derselben VPC befinden und das Netzwerk vertrauenswürdig ist. Weitere Informationen zur Verwendung verschlüsselter Verbindungen finden Sie unter Herstellen einer Verbindung mit Aurora MySQL mithilfe von SSL.

6. SQL-Befehle ausführen

   Der folgende SQL-Befehl zeigt z B. das aktuelle Datum und die aktuelle Zeit an:

   ```
   SELECT CURRENT_TIMESTAMP;
   ```

Schritt 4: Löschen Sie die EC2 Instance und den DB-Cluster

Nachdem Sie sich mit der EC2 Beispiel-Instance und dem DB-Cluster, die Sie erstellt haben, verbunden und diese erkundet haben, löschen Sie sie, damit Ihnen keine Gebühren mehr berechnet werden.

Wenn Sie früher CloudFormation Ressourcen erstellt haben, überspringen Sie diesen Schritt und fahren Sie mit dem nächsten Schritt fort.

Um die EC2 Instanz zu löschen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die EC2 Amazon-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie im Navigationsbereich Instances aus.
3. Wählen Sie die EC2 Instance aus und wählen Sie dann Instance-Status, Instanz beenden aus.
4. Wenn Sie zur Bestätigung aufgefordert werden, wählen Sie Beenden aus.

Weitere Informationen zum Löschen einer EC2 Instance finden Sie unter Kündigen Ihrer Instance im EC2 Amazon-Benutzerhandbuch.

So löschen Sie den DB-Cluster

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) und anschließend die DB-Instance aus, die mit dem DB-Cluster verknüpft ist.
3. Klicken Sie bei Actions auf Delete.
4. Deaktivieren Sie das Kontrollkästchen Abschließenden Snapshot erstellen?.
5. Bestätigen Sie, und wählen Sie Löschen.

Wenn alle mit einem DB-Cluster verknüpften DB-Instances gelöscht sind, wird der DB-Cluster automatisch gelöscht.

(Optional) Löschen Sie die EC2 Instance und den DB-Cluster, die mit erstellt wurden CloudFormation

Wenn Sie früher CloudFormation Ressourcen erstellt haben, löschen Sie den CloudFormation Stack, nachdem Sie sich mit der EC2 Beispiel-Instance und dem DB-Cluster verbunden und diese erkundet haben, sodass Ihnen diese nicht mehr in Rechnung gestellt werden.

Um die CloudFormation Ressourcen zu löschen

1. Öffnen Sie die CloudFormation Konsole.
2. Wählen Sie auf der Seite Stacks in der CloudFormation Konsole den Root-Stack aus (den Stack ohne Namen VPCStack BastionStack oder AMSNS).
3. Wählen Sie Löschen aus.
4. Wenn Sie zur Bestätigung aufgefordert werden, wählen Sie Stack löschen aus.

Weitere Informationen zum Löschen eines Stacks in CloudFormation finden Sie im Benutzerhandbuch unter Löschen eines Stacks auf der CloudFormationAWS CloudFormation Konsole.

(Optional) Verbinden Sie Ihren DB-Cluster mit einer Lambda-Funktion

Sie können Ihren DB-Cluster von Aurora MySQL auch mit einer Lambda-Serverless-Rechenressource verbinden. Mit Lambda-Funktionen können Sie Code ausführen, ohne die Infrastruktur bereitstellen oder verwalten zu müssen. Eine Lambda-Funktion ermöglicht es Ihnen auch, automatisch auf Codeausführungsanfragen jeder Größenordnung zu reagieren, von einem Dutzend Ereignissen pro Tag bis hin zu Hunderten von Ereignissen pro Sekunde. Weitere Informationen finden Sie unter Automatisches Verbinden einer Lambda-Funktion mit einem Aurora-DB-Cluster.

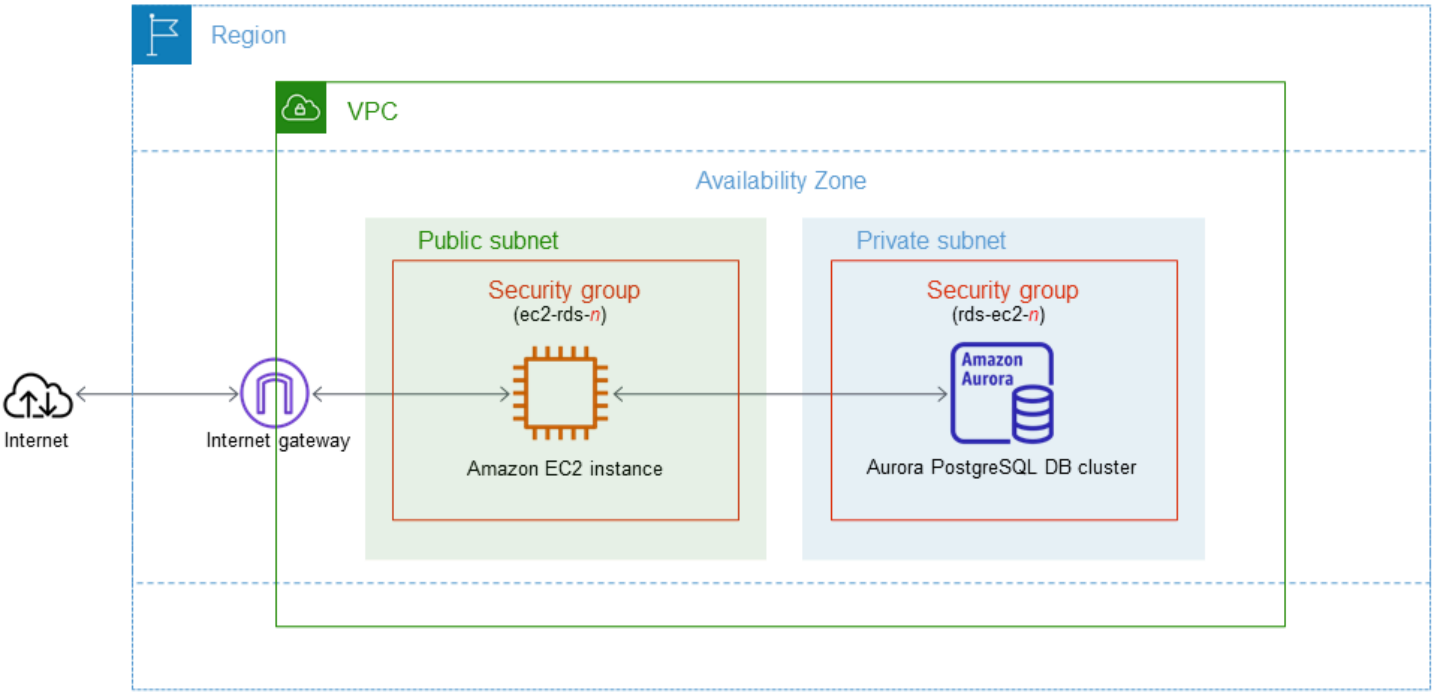
Erstellen eines DB-Clusters von Aurora PostgreSQL und Herstellen einer Verbindung

In diesem Tutorial werden eine EC2 Instance und ein Aurora PostgreSQL-DB-Cluster erstellt. Das Tutorial zeigt Ihnen, wie Sie mit einem Standard-PostgreSQL-Client von der EC2 Instance aus auf den DB-Cluster zugreifen. Als bewährte Methode erstellt dieses Tutorial einen privaten DB-Cluster in einer Virtual Private Cloud (VPC). In den meisten Fällen können andere Ressourcen in derselben VPC, z. B. EC2 Instances, auf den DB-Cluster zugreifen, aber Ressourcen außerhalb der VPC können nicht darauf zugreifen.

Nach Abschluss des Tutorials gibt es in jeder Availability Zone im VPC ein öffentliches und ein privates Subnetz. In einer Availability Zone befindet sich die EC2 Instance im öffentlichen Subnetz und die DB-Instance im privaten Subnetz.

> **Important**
>
> Für die Erstellung eines Kontos fallen keine Gebühren an AWS . Wenn Sie dieses Tutorial abschließen, können Ihnen jedoch Kosten für die AWS Ressourcen entstehen, die Sie verwenden. Sie können diese Ressourcen nach Abschluss des Tutorials löschen, wenn sie nicht mehr benötigt werden.

Das folgende Diagramm zeigt die Konfiguration nach Abschluss des Tutorials.



Dieses Tutorial ermöglicht es Ihnen, Ihre Ressourcen auf eine der folgenden Weisen zu erstellen:

1. Verwenden Sie das - und AWS-Managementkonsole Schritt 1: Eine EC2 Instanz erstellen Schritt 2: Erstellen eines DB-Clusters von Aurora PostgreSQL
2. Verwenden Sie CloudFormation , um die Datenbankinstanz und die EC2 Instanz zu erstellen - (Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora PostgreSQL-Cluster mit CloudFormation

Die erste Methode verwendet Einfache Erstellung, um einen privaten DB-Cluster von Aurora PostgreSQL mit der AWS-Managementkonsole zu erstellen. Geben Sie hier nur den DB-Engine-Typ, die DB-Instance-Größe und die DB-Cluster-Kennung an. Einfache Erstellung verwendet für die anderen Konfigurationsoptionen die Standardeinstellung.

Wenn Sie stattdessen Standarderstellung verwenden, können Sie beim Erstellen eines DB-Clusters weitere Konfigurationsoptionen wählen. Zu diesen Optionen gehören Einstellungen für Verfügbarkeit, Sicherheit, Backups und Wartung. Wenn Sie einen öffentliche DB-Cluster erstellen möchten, müssen Sie Standarderstellung verwenden. Weitere Informationen finden Sie unter the section called “Erstellen eines DB-Clusters”.

Themen

- Voraussetzungen

- Schritt 1: Eine EC2 Instanz erstellen
- Schritt 2: Erstellen eines DB-Clusters von Aurora PostgreSQL
- (Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora PostgreSQL-Cluster mit CloudFormation
- Schritt 3: Herstellen einer Verbindung mit einem DB-Cluster von Aurora PostgreSQL
- Schritt 4: Löschen Sie die EC2 Instance und den DB-Cluster
- (Optional) Löschen Sie die EC2 Instance und den DB-Cluster, die mit erstellt wurden CloudFormation
- (Optional) Verbinden Sie Ihren DB-Cluster mit einer Lambda-Funktion

Voraussetzungen

Bevor Sie die Schritte in diesem Abschnitt abschließen, stellen Sie sicher, dass Sie folgende Voraussetzungen erfüllen:

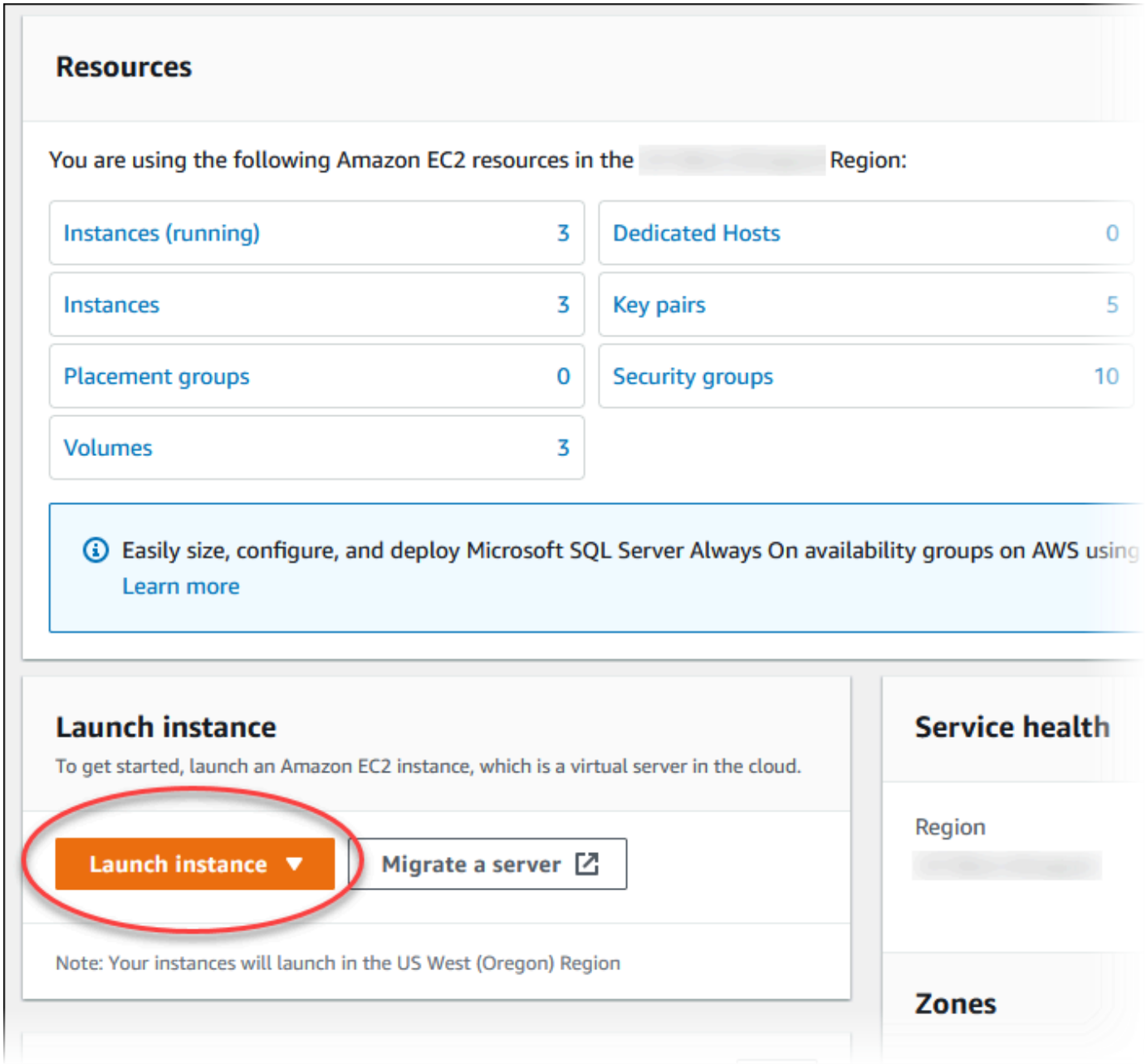
- Melden Sie sich an für ein AWS-Konto
- Erstellen eines Benutzers mit Administratorzugriff

Schritt 1: Eine EC2 Instanz erstellen

Erstellen Sie eine EC2 Amazon-Instance, mit der Sie eine Verbindung zu Ihrer Datenbank herstellen.

Um eine EC2 Instance zu erstellen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die EC2 Amazon-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie in der oberen rechten Ecke von die aus AWS-Managementkonsole, AWS-Region in der Sie die Instance erstellen möchten. EC2
3. Wählen Sie EC2 Dashboard und dann Launch instance aus, wie in der folgenden Abbildung gezeigt.

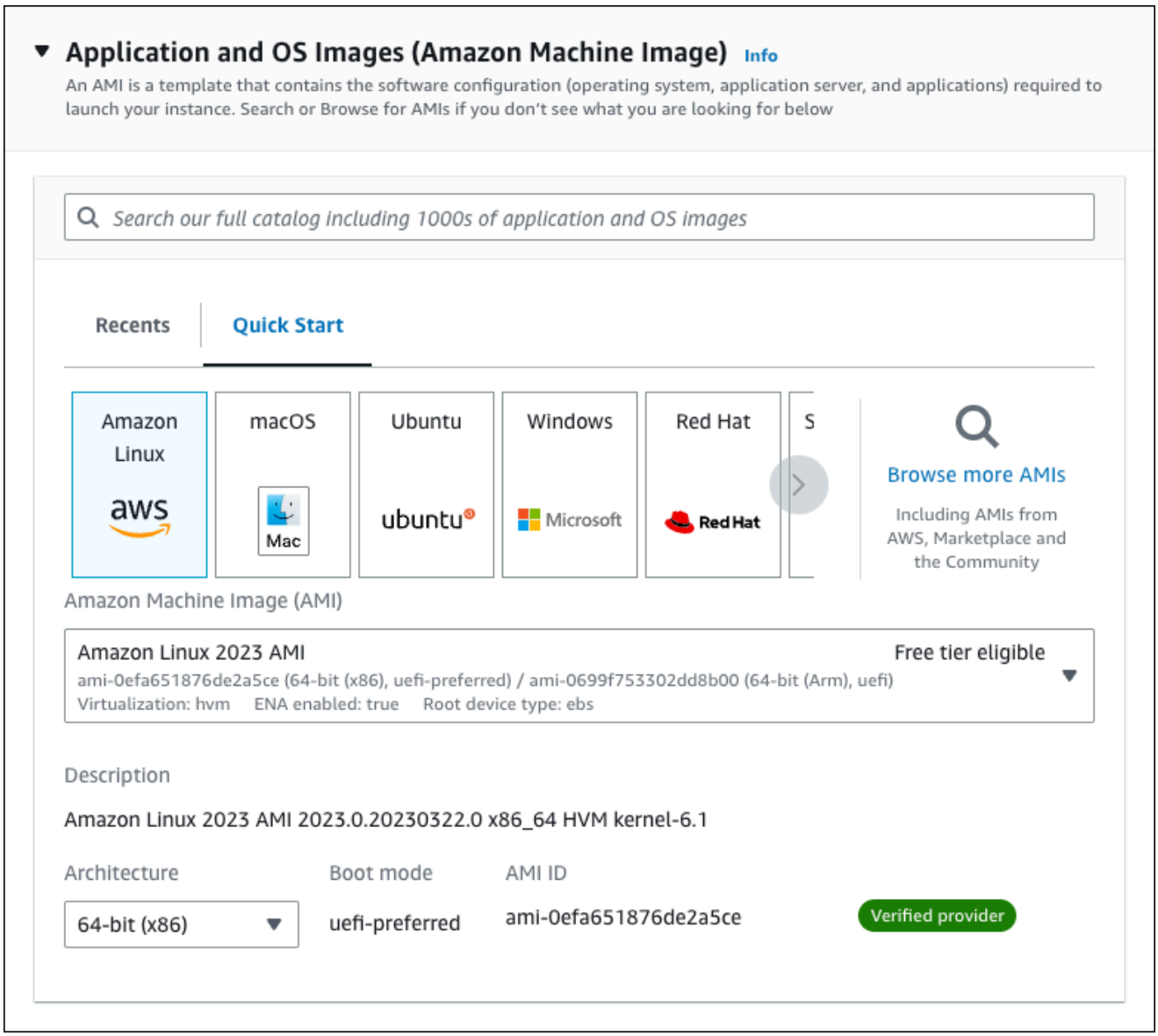


Die Seite Eine Instance starten wird geöffnet.

4. Wählen Sie auf der Seite Eine Instance starten die folgenden Einstellungen aus.

   a. Geben Sie unter Name and tags (Name und Tags) als Name den Namen **`ec2-database-connect`** ein.

   b. Wählen Sie unter Anwendungs- und Betriebssystem-Images (Amazon Machine Image) die Option Amazon Linux und dann die Registerkarte Amazon Linux 2023 AMI aus. Übernehmen Sie für alle anderen Einstellungen die Standardwerte.



c. Wählen Sie unter Instance type (Instance-Typ) den Wert t2.micro aus.

d. Wählen Sie unter Key pair (login) (Schlüsselpaar (Anmeldung)) einen Key pair name (Schlüsselpaarname), um ein vorhandenes Schlüsselpaar zu verwenden. Um ein neues key pair für die EC2 Amazon-Instance zu erstellen, wählen Sie Neues key pair erstellen und verwenden Sie dann das Fenster key pair erstellen, um es zu erstellen.

   Weitere Informationen zum Erstellen eines neuen key pair finden Sie unter Erstellen eines key pair im EC2 Amazon-Benutzerhandbuch.

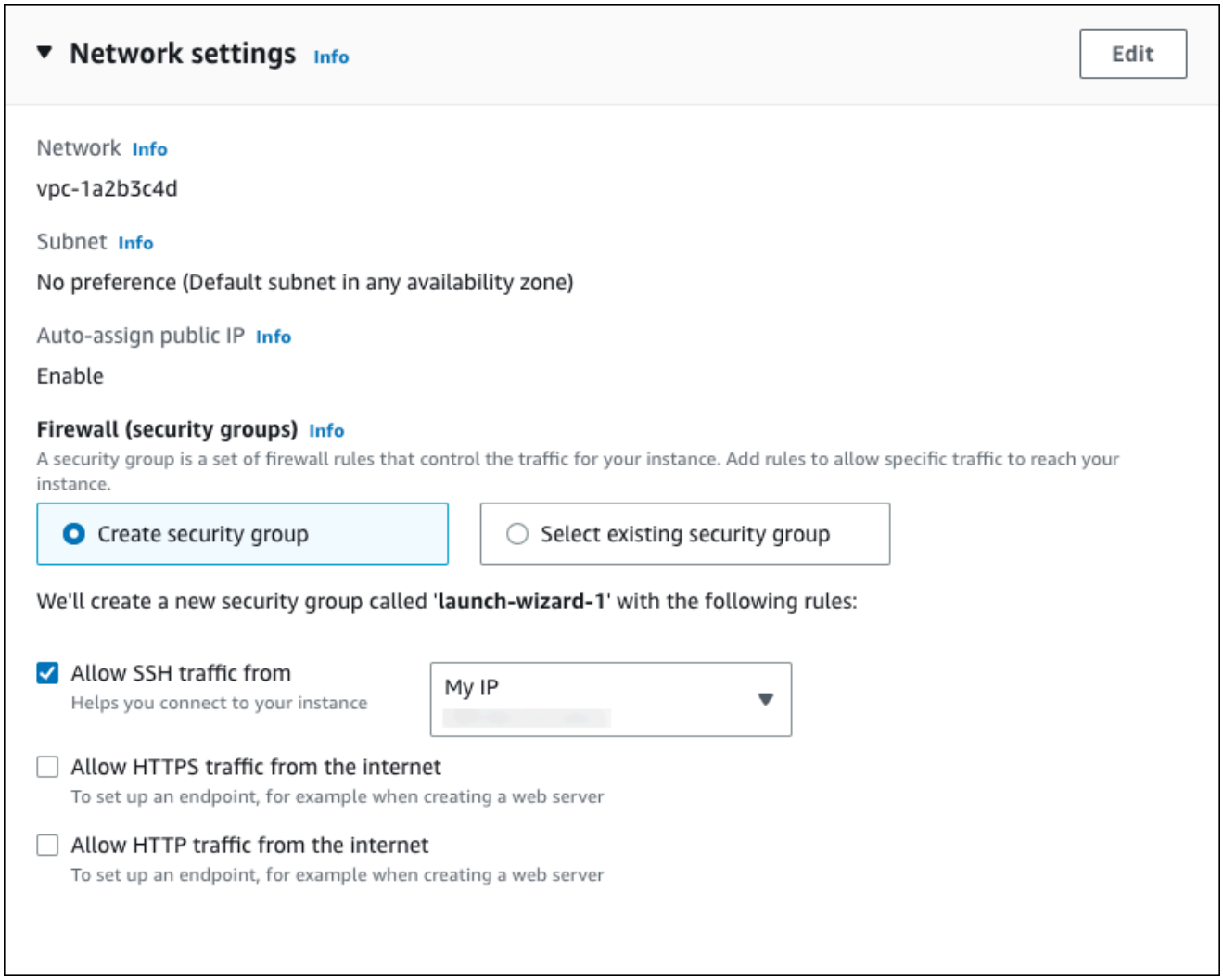
e. Wählen Sie unter „SSH-Verkehr zulassen“ in den Netzwerkeinstellungen die Quelle der SSH-Verbindungen zur Instance aus. EC2

Sie können My IP (Meine IP) auswählen, wenn die angezeigte IP-Adresse für SSH-Verbindungen korrekt ist. Andernfalls können Sie mithilfe von Secure Shell (SSH) die IP-Adresse bestimmen, die für die Verbindung zu EC2 Instances in Ihrer VPC verwendet werden soll. Um Ihre öffentliche IP-Adresse in einem anderen Browserfenster oder einer anderen Registerkarte zu ermitteln, können Sie den Dienst unter verwenden. https://checkip.amazonaws.com Ein Beispiel für eine IP-Adresse ist 192.0.2.1/32.

In vielen Fällen können Sie eine Verbindung über einen Internetdienstanbieter (ISP) oder hinter Ihrer Firewall ohne statische IP-Adresse herstellen. Bestimmen Sie in diesem Fall den Bereich der IP-Adressen, die von Client-Computern verwendet werden.

> **Warning**
>
> Wenn Sie `0.0.0.0/0` den SSH-Zugriff verwenden, ermöglichen Sie allen IP-Adressen den Zugriff auf Ihre öffentlichen EC2 Instanzen über SSH. Dieser Ansatz ist zwar für kurze Zeit in einer Testumgebung zulässig, aber für Produktionsumgebungen sehr unsicher. Autorisieren Sie in der Produktion nur eine bestimmte IP-Adresse oder einen bestimmten Adressbereich für den Zugriff auf Ihre EC2 Instances über SSH.

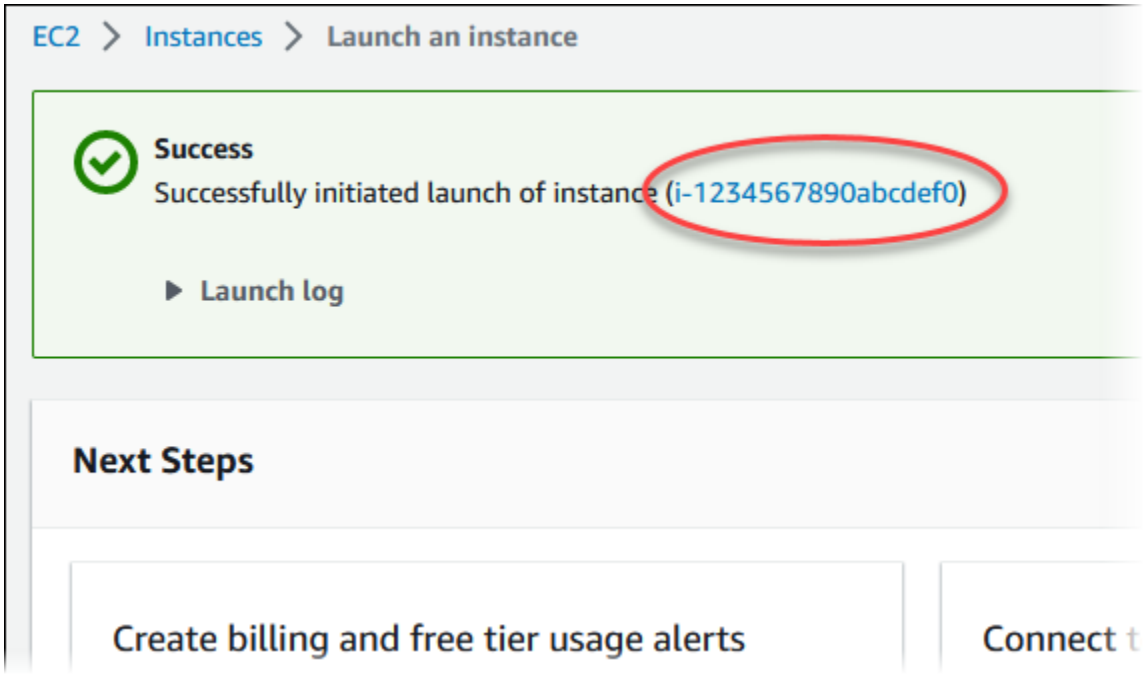
Die folgende Abbildung zeigt ein Beispiel für den Bereich Netzwerkeinstellungen.



f. Übernehmen Sie die Standardwerte für die übrigen Abschnitte.

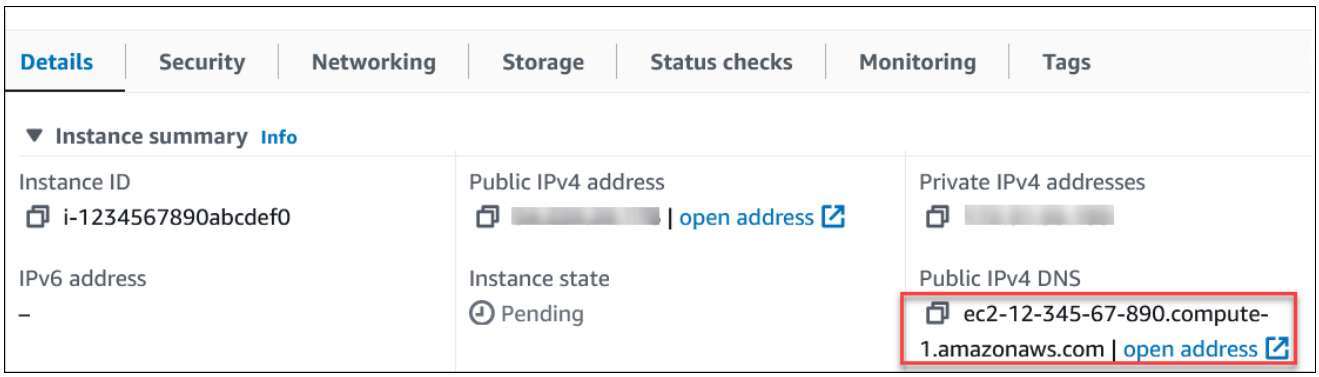
g. Sehen Sie sich im Bereich Zusammenfassung eine Zusammenfassung Ihrer EC2 Instance-Konfiguration an. Wenn Sie bereit sind, wählen Sie Launch instance aus.

5. Notieren Sie sich auf der Seite Startstatus die ID für Ihre neue EC2 Instance, zum Beispiel:`i-1234567890abcdef0`.

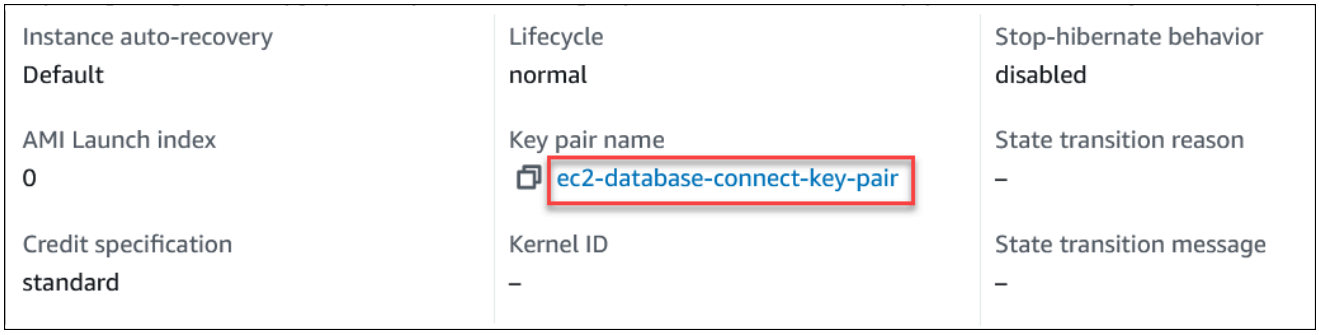


6. Wählen Sie die EC2 Instance-ID, um die Liste der EC2 Instances zu öffnen, und wählen Sie dann Ihre EC2 Instance aus.
7. Notieren Sie sich die folgenden Werte auf der Registerkarte Details. Diese benötigen Sie, wenn Sie eine Verbindung über SSH herstellen:

   a. Notieren Sie sich in der Instanzübersicht den Wert für Public IPv4 DNS.



   b. Notieren Sie sich unter Instance-Details den Wert für Schlüsselpaarname.



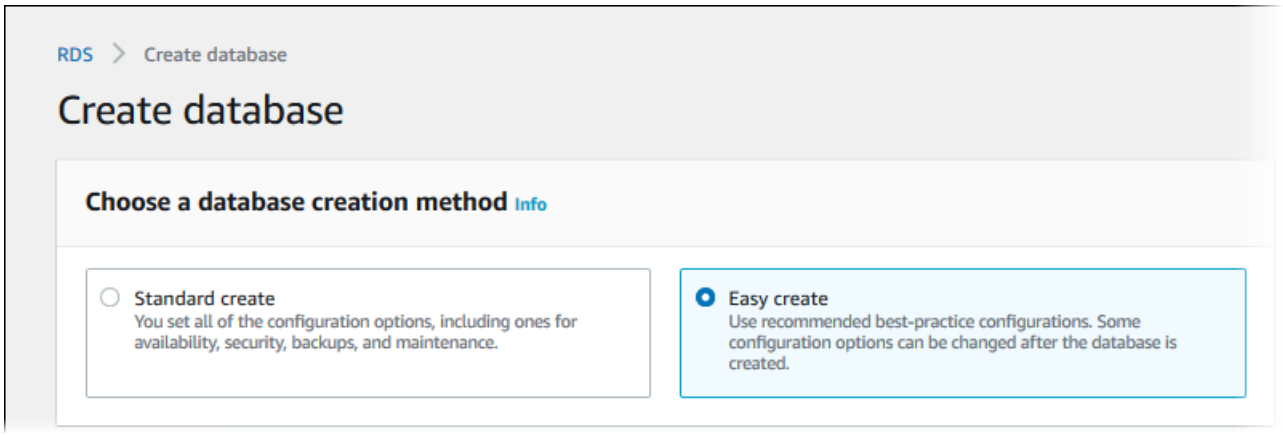
8. Warten Sie, bis der Instanzstatus Ihrer EC2 Instance den Status Running hat, bevor Sie fortfahren.

Schritt 2: Erstellen eines DB-Clusters von Aurora PostgreSQL

In diesem Beispiel verwenden Sie Einfache Erstellung, um einen DB-Cluster von Aurora PostgreSQL mit einer DB-Instance-Klasse vom Typ „db.t4g.large“ zu erstellen.

So erstellen Sie einen DB-Cluster von Aurora PostgreSQL der Option „Einfache Erstellung“

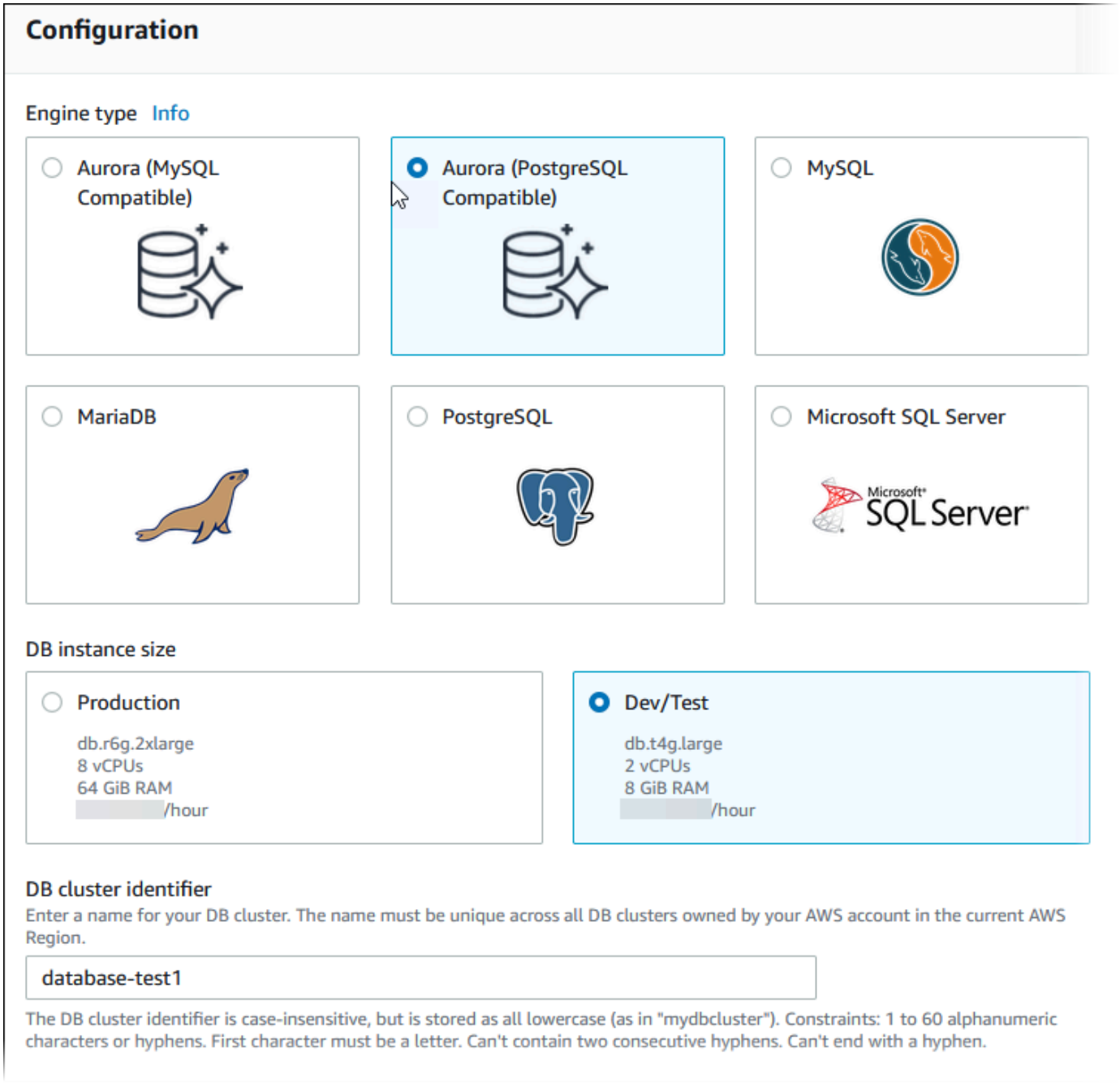
1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie rechts oben in der Amazon-RDS-Konsole die AWS-Region aus, in der Sie den DB-Cluster erstellen möchten.
3. Wählen Sie im Navigationsbereich Datenbanken aus.
4. Wählen Sie Datenbank erstellen aus und vergewissern Sie sich, dass Einfache Erstellung ausgewählt ist.



5. Wählen Sie unter Konfiguration die Option Aurora (PostgreSQL-kompatibel) als Engine-Typ aus.

6. Wählen Sie in DB instance size (DB-Instance-Größe) die Option Dev/Test (Entwicklung/Testen) aus.
7. Geben Sie unter DB-Cluster-Kennung die Zeichenfolge **`database-test1`** ein.

   Die Seite Datenbank erstellen sollte ähnlich wie in der folgenden Abbildung gezeigt aussehen.



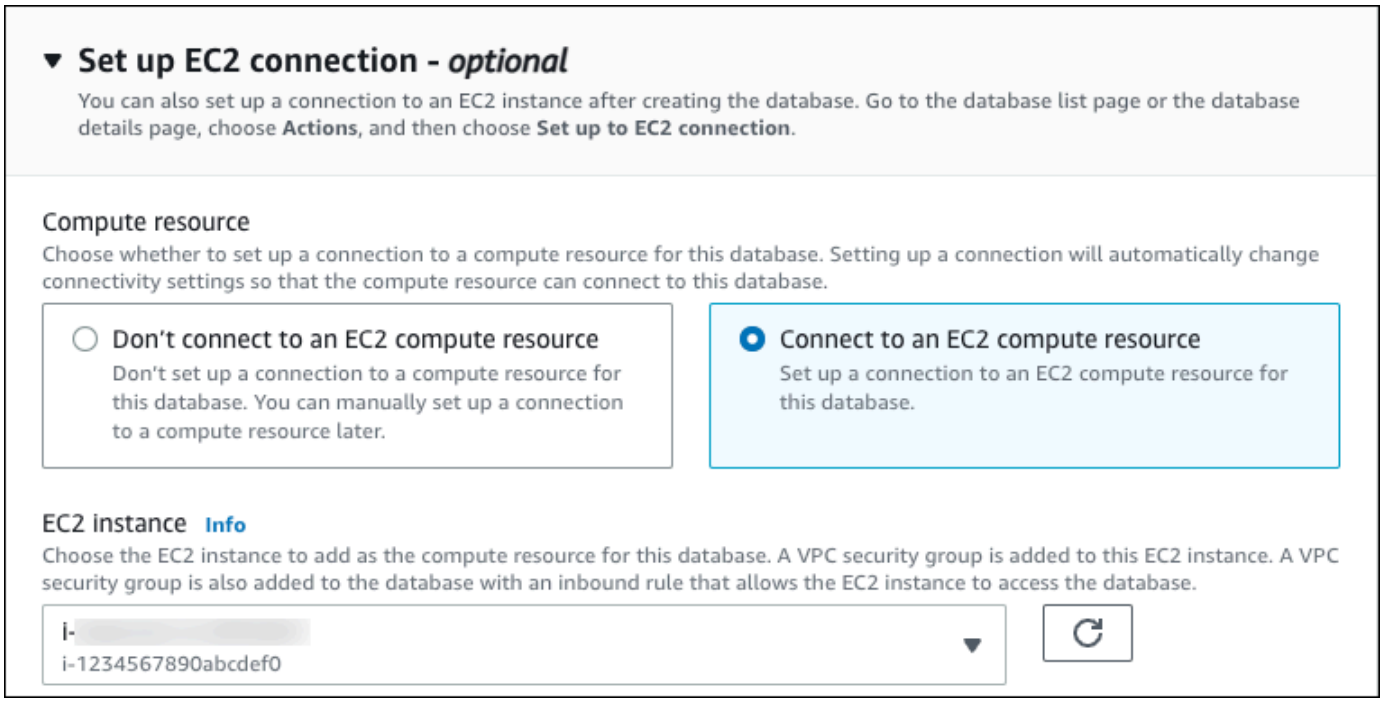
8. Geben Sie im Feld Hauptbenutzername einen Namen für den Benutzer ein oder behalten Sie den Standardnamen (**`postgres`**) bei.

9. Um für den DB-Cluster ein automatisch generiertes Hauptpasswort zu verwenden, wählen Sie Passwort automatisch generieren aus.

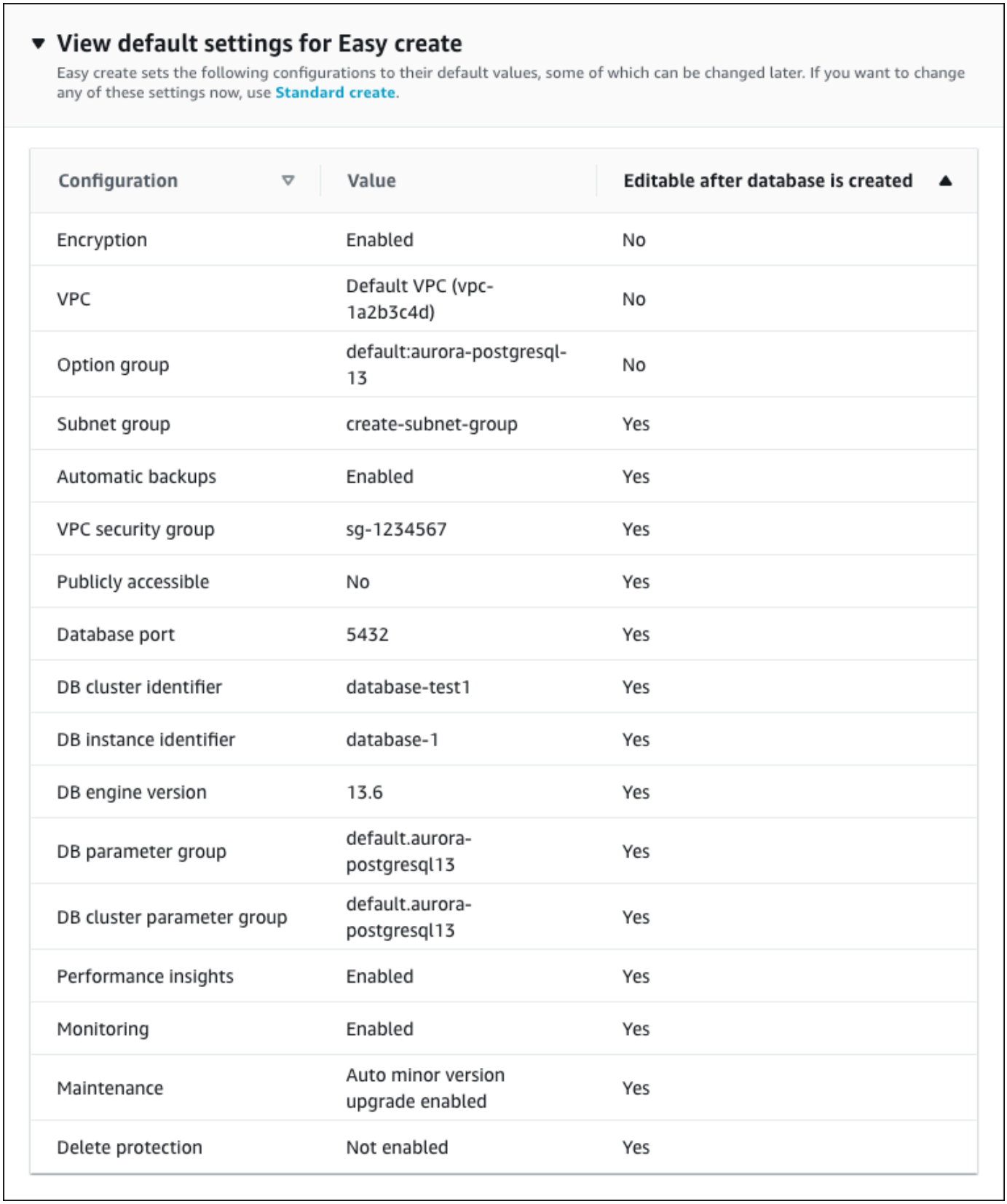
   Um das Hauptpasswort einzugeben, deaktivieren Sie das Kästchen Passwort automatisch generieren und geben Sie anschließend dasselbe Passwort in Hauptpasswort und Passwort bestätigen ein.

10. Um eine Verbindung mit der EC2 Instance einzurichten, die Sie zuvor erstellt haben, öffnen Sie EC2 Verbindung einrichten — optional.

    Wählen Sie Mit einer EC2 Rechenressource verbinden aus. Wählen Sie die EC2 Instanz aus, die Sie zuvor erstellt haben.



11. (Optional) Öffnen Sie Anzeigen von Standardeinstellungen für eine einfache Erstellung.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-postgresql-13	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	5432	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	13.6	Yes
DB parameter group	default.aurora-postgresql13	Yes
DB cluster parameter group	default.aurora-postgresql13	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

Sie können die Standardeinstellungen von Einfache Erstellung einsehen. Die Spalte Nach Erstellung der Datenbank editierbar zeigt, welche Optionen Sie nach der Datenbankerstellung ändern können.

- Wenn für eine Einstellung Nein in dieser Spalte steht und Sie eine andere Einstellung verwenden möchten, können Sie Standarderstellung verwenden, um den DB-Cluster zu erstellen.
- Wenn für eine Einstellung Ja in dieser Spalte steht und Sie eine andere Einstellung verwenden möchten, können Sie entweder Standarderstellung auswählen, um den DB-Cluster zu erstellen, oder den DB-Cluster nach der Erstellung ändern, um die Einstellung zu ändern.

12. Wählen Sie Datenbank erstellen aus.

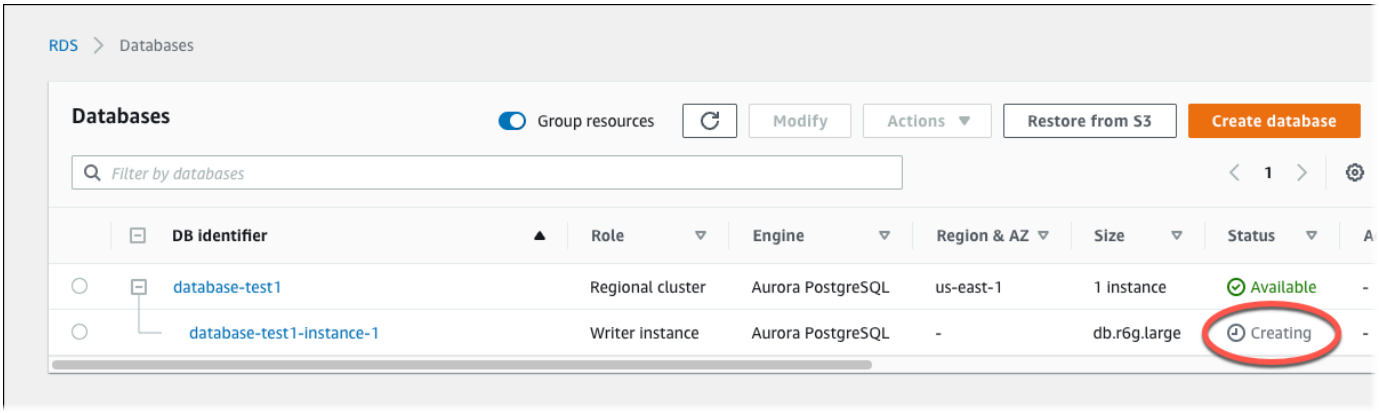
    Um den Hauptbenutzernamen und das Passwort für den DB-Cluster anzuzeigen, wählen Sie Anmeldeinformationen anzeigen aus.

    Sie können den angezeigten Benutzernamen und das angezeigte Passwort verwenden, um als Hauptbenutzer eine Verbindung mit dem DB-Cluster herzustellen.

    > **Important**
    >
    > Sie können dieses Passwort für den Hauptbenutzer nicht erneut anzeigen. Wenn Sie es nicht notieren, müssen Sie es möglicherweise ändern.
    > Wenn Sie das Passwort für den Hauptbenutzer ändern müssen, nachdem der DB-Cluster verfügbar wurde, können Sie den DB-Cluster entsprechend ändern. Weitere Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

13. Wählen Sie in der Liste Datenbanken den Namen des neuen DB-Clusters von Aurora PostgreSQL aus.

    Die Writer-Instance hat den Status Wird erstellt, bis der DB-Cluster bereit ist und verwendet werden kann.



Wenn sich der Status der Writer-Instance in Verfügbar ändert, können Sie die Verbindung mit dem DB-Cluster herstellen. Je nach Klasse und Speicherort der DB-Instance kann es bis zu 20 Minuten dauern, bis der neue DB-Cluster verfügbar ist.

(Optional) Erstellen Sie eine VPC, eine EC2 Instanz und einen Aurora PostgreSQL-Cluster mit CloudFormation

Anstatt die Konsole zur Erstellung Ihrer VPC, EC2 Instance und Ihres Aurora PostgreSQL-DB-Clusters CloudFormation zu verwenden, können Sie AWS Ressourcen bereitstellen, indem Sie die Infrastruktur als Code behandeln. Um Ihnen zu helfen, Ihre AWS Ressourcen in kleinere und besser verwaltbare Einheiten zu organisieren, können Sie die Nested-Stack-Funktionalität verwenden. CloudFormation Weitere Informationen finden Sie unter Einen Stack auf der CloudFormation Konsole erstellen und Mit verschachtelten Stacks arbeiten.

> **Important**
>
> CloudFormation ist kostenlos, aber die Ressourcen, die CloudFormation erstellt werden, sind live. Es fallen die standardmäßigen Nutzungsgebühren für diese Ressourcen an, bis sie von Ihnen beendet werden. Weitere Informationen dazu finden Sie unter Amazon Aurora – Preise.

Gehen Sie wie folgt vor, um Ihre Ressourcen mithilfe der CloudFormation Konsole zu erstellen:

- Schritt 1: Laden Sie die CloudFormation Vorlage herunter
- Schritt 2: Konfigurieren Sie Ihre Ressourcen mit CloudFormation

Laden Sie die CloudFormation Vorlage herunter

Eine CloudFormation Vorlage ist eine JSON- oder YAML-Textdatei, die die Konfigurationsinformationen zu den Ressourcen enthält, die Sie im Stack erstellen möchten. Diese Vorlage erstellt neben dem Aurora-Cluster auch eine VPC und einen Bastion-Host für Sie.

Um die Vorlagendatei herunterzuladen, öffnen Sie den folgenden Link: Aurora CloudFormation PostgreSQL-Vorlage.

Klicken Sie auf der Github-Seite auf die Schaltfläche Rohdatei herunterladen, um die YAML-Vorlagendatei zu speichern.

Konfigurieren Sie Ihre Ressourcen mit CloudFormation

> **Note**
>
> Bevor Sie diesen Vorgang starten, stellen Sie sicher, dass Sie ein Schlüsselpaar für eine EC2 Instance in Ihrem haben AWS-Konto. Weitere Informationen finden Sie unter EC2 Amazon-Schlüsselpaare und Linux-Instances.

Wenn Sie die CloudFormation Vorlage verwenden, müssen Sie die richtigen Parameter auswählen, um sicherzustellen, dass Ihre Ressourcen ordnungsgemäß erstellt werden. Führen Sie die folgenden Schritte aus:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die CloudFormation Konsole unter https://console.aws.amazon.com/cloudformation.
2. Wählen Sie Stack erstellen aus.
3. Wählen Sie im Abschnitt „Vorlage angeben“ die Option Vorlagendatei von Ihrem Computer hochladen und dann Weiter aus.
4. Legen Sie auf der Seite Stack-Details angeben die folgenden Parameter fest:
   a. Setzen Sie den Stack-Namen auf AurPostgre SQLTest Stack.
   b. Legen Sie unter Parameter Availability Zones fest, indem Sie zwei Availability Zones auswählen.
   c. Wählen Sie unter Linux Bastion Host configuration für Key Name ein key pair aus, um sich bei Ihrer EC2 Instance anzumelden.
   d. Stellen Sie in den Bastion-Host-Konfigurationseinstellungen in Linux den zulässigen IP-Bereich auf Ihre IP-Adresse ein. Um mithilfe von Secure Shell (SSH) eine Verbindung zu EC2 Instances

in Ihrer VPC herzustellen, ermitteln Sie Ihre öffentliche IP-Adresse mithilfe des Dienstes unter. https://checkip.amazonaws.com Ein Beispiel für eine IP-Adresse ist 192.0.2.1/32.

> **Warning**
>
> Wenn Sie `0.0.0.0/0` den SSH-Zugriff verwenden, ermöglichen Sie allen IP-Adressen den Zugriff auf Ihre öffentlichen EC2 Instances über SSH. Dieser Ansatz ist zwar für kurze Zeit in einer Testumgebung zulässig, aber für Produktionsumgebungen sehr unsicher. Autorisieren Sie in der Produktion nur eine bestimmte IP-Adresse oder einen bestimmten Adressbereich für den Zugriff auf Ihre EC2 Instances über SSH.

e. Stellen Sie unter Allgemeine Datenbankkonfiguration die Datenbank-Instance-Klasse auf db.t4g.large ein.

f. Setzen Sie den Datenbanknamen auf **`database-test1`**.

g. Geben Sie unter Hauptbenutzername der Datenbank den Namen des Hauptbenutzers ein.

h. Setzen Sie für dieses Tutorial die Option DB-Hauptbenutzerpasswort mit Secrets Manager verwalten auf `false`.

i. Legen Sie für Datenbankpasswort ein Passwort Ihrer Wahl fest. Notieren Sie sich dieses Passwort für weitere Schritte im Tutorial.

j. Setzen Sie Multi-AZ-Bereitstellung auf `false`.

k. Lassen Sie alle anderen Einstellungen auf den Standardwerten. Klicken Sie auf Weiter, um fortzufahren.

5. Behalten Sie auf der Seite Stack-Optionen konfigurieren alle Standardoptionen bei. Klicken Sie auf Weiter, um fortzufahren.
6. Wählen Sie auf der Seite Stack überprüfen die Option Absenden aus, nachdem Sie die Optionen für die Datenbank und den Linux-Bastion-Host überprüft haben.

Sehen Sie sich nach Abschluss der Stack-Erstellung die Stacks mit Namen BastionStackund APGNS an, um die Informationen zu notieren, die Sie für die Verbindung mit der Datenbank benötigen. Weitere Informationen finden Sie unter CloudFormation Stack-Daten und Ressourcen anzeigen auf der. AWS-Managementkonsole

Schritt 3: Herstellen einer Verbindung mit einem DB-Cluster von Aurora PostgreSQL

Sie können für die Verbindung mit dem DB-Cluster eine beliebige PostgreSQL-Client-Standardanwendung verwenden. In diesem Beispiel stellen Sie eine Verbindung mit einem DB-Cluster von Aurora PostgreSQL mithilfe des psql-Befehlszeilen-Clients her.

So stellen Sie eine Verbindung mit einem DB-Cluster von Aurora PostgreSQL her

1. Suchen Sie nach dem Endpunkt (DNS-Name) und der Portnummer der Writer-Instance für Ihren DB-Cluster.

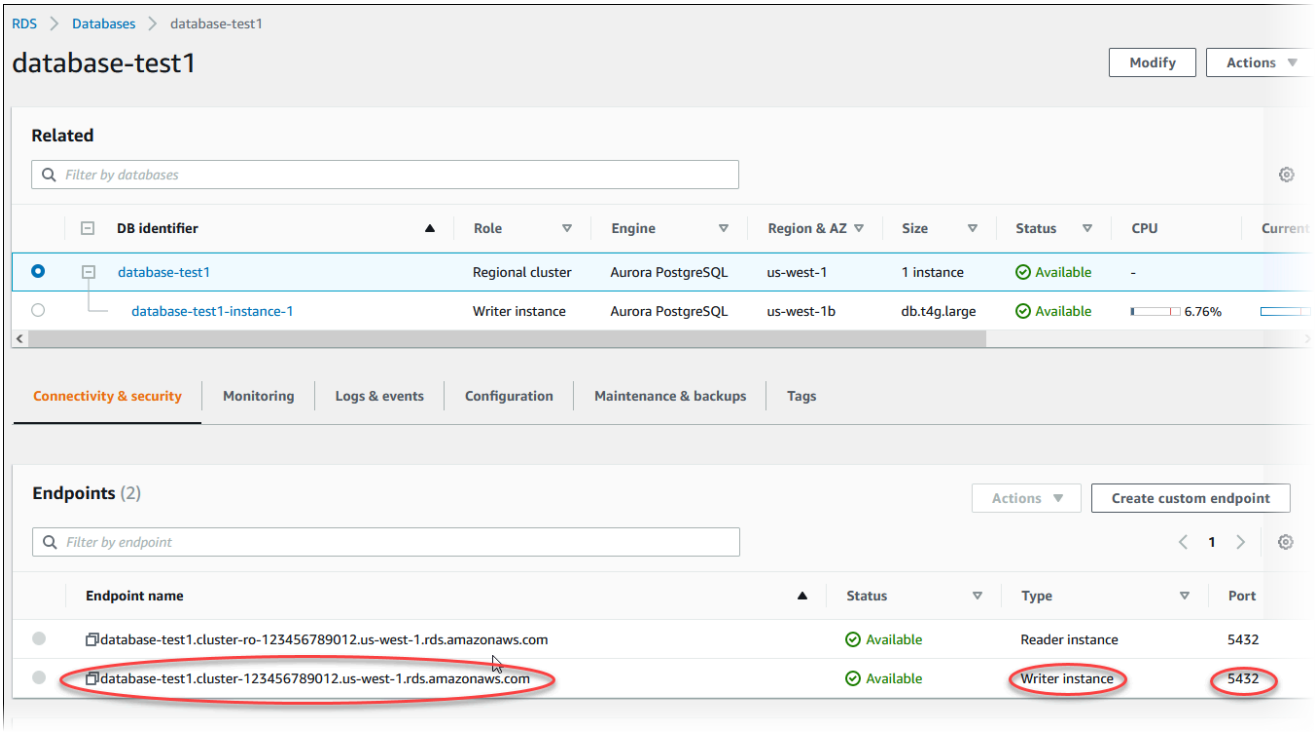
   a. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.

   b. Wählen Sie in der oberen rechten Ecke der Amazon RDS-Konsole den AWS-Region für den DB-Cluster aus.

   c. Wählen Sie im Navigationsbereich Datenbanken aus.

   d. Wählen Sie den Namen des DB-Clusters von Aurora PostgreSQL aus, um dessen Details anzuzeigen.

   e. Kopieren Sie auf der Registerkarte Konnektivität und Sicherheit den Endpunkt der Writer-Instance. Notieren Sie sich auch die Portnummer. Sie benötigen sowohl den Endpunkt als auch die Portnummer, um die Verbindung mit dem DB-Cluster herzustellen.



2. Stellen Sie eine Verbindung zu der EC2 Instance her, die Sie zuvor erstellt haben, indem Sie den Schritten unter Connect to your Linux Instance im EC2 Amazon-Benutzerhandbuch folgen.

   Wir empfehlen, dass Sie über SSH eine Verbindung zu Ihrer EC2 Instance herstellen. Wenn das SSH-Client-Dienstprogramm unter Windows, Linux oder Mac installiert ist, können Sie mit dem folgenden Befehlsformat eine Verbindung mit der Instance herstellen:

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   Gehen Sie beispielsweise davon aus, dass dies `/dir1` unter Linux gespeichert `ec2-database-connect-key-pair.pem` ist und das öffentliche IPv4 DNS für Ihre EC2 Instance bereits. `ec2-12-345-678-90.compute-1.amazonaws.com` Ihr SSH-Befehl würde wie folgt aussehen:

   ```
   ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
   user@ec2-12-345-678-90.compute-1.amazonaws.com
   ```

3. Holen Sie sich die neuesten Bugfixes und Sicherheitsupdates, indem Sie die Software auf Ihrer EC2 Instance aktualisieren. Führen Sie dazu den folgenden Befehl aus.

> **Note**
>
> Mit der Option `-y` werden die Updates installiert, ohne um Bestätigung zu bitten. Um Updates vor der Installation zu überprüfen, lassen Sie diese Option aus.

```
sudo dnf update -y
```

4. Installieren Sie den psql-Befehlszeilen-Client von PostgreSQL auf Amazon Linux 2023 mit dem folgenden Befehl:

```
sudo dnf install postgresql15
```

5. Stellen Sie eine Verbindung mit dem DB-Cluster von Aurora PostgreSQL her. Geben Sie z. B. den folgenden Befehl ein. Mit dieser Aktion können Sie mithilfe des psql-Clients eine Verbindung mit dem DB-Cluster von Aurora PostgreSQL herstellen.

   Ersetzen Sie den Endpunkt der Writer-Instance für *`endpoint`*, den Namen der Datenbank `--dbname`, mit der Sie eine Verbindung für *`postgres`* herstellen möchten, und den Hauptbenutzernamen, den Sie für *`postgres`* verwendet haben. Geben Sie das Master-Passwort ein, das Sie bei der Aufforderung zur Eingabe eines Passworts verwendet haben.

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

   Nachdem Sie das Passwort für den Benutzer eingegeben haben, sollte eine Ausgabe wie die folgende angezeigt werden.

```
psql (14.3, server 14.6)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

   Weitere Informationen zum Herstellen einer Verbindung mit einem DB-Cluster von Aurora PostgreSQL finden Sie unter Herstellen einer Verbindung mit einem Amazon-Aurora-PostgreSQL-DB-Cluster. Wenn Sie sich nicht mit Ihrem DB-Cluster verbinden können, erhalten Sie unter Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden Hilfe.

Aus Sicherheitsgründen empfiehlt es sich, verschlüsselte Verbindungen zu verwenden. Verwenden Sie eine unverschlüsselte PostgreSQL-Verbindung nur, wenn sich Client und Server in derselben VPC befinden und das Netzwerk vertrauenswürdig ist. Weitere Informationen zur Verwendung verschlüsselter Verbindungen finden Sie unter Sicherung von Aurora-PostgreSQL-Daten mit SSL/TLS.

6. SQL-Befehle ausführen

   Der folgende SQL-Befehl zeigt z B. das aktuelle Datum und die aktuelle Zeit an:

```
SELECT CURRENT_TIMESTAMP;
```

Schritt 4: Löschen Sie die EC2 Instance und den DB-Cluster

Nachdem Sie sich mit der EC2 Beispiel-Instance und dem DB-Cluster, die Sie erstellt haben, verbunden und diese erkundet haben, löschen Sie sie, damit Ihnen keine Gebühren mehr berechnet werden.

Wenn Sie früher CloudFormation Ressourcen erstellt haben, überspringen Sie diesen Schritt und fahren Sie mit dem nächsten Schritt fort.

Um die EC2 Instanz zu löschen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die EC2 Amazon-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie im Navigationsbereich Instances aus.
3. Wählen Sie die EC2 Instance aus und wählen Sie dann Instance-Status, Instanz beenden aus.
4. Wenn Sie zur Bestätigung aufgefordert werden, wählen Sie Beenden aus.

Weitere Informationen zum Löschen einer EC2 Instance finden Sie unter Kündigen Ihrer Instance im EC2 Amazon-Benutzerhandbuch.

So löschen Sie einen DB-Cluster

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie Databases (Datenbanken) und anschließend die DB-Instance aus, die mit dem DB-Cluster verknüpft ist.
3. Klicken Sie bei Actions auf Delete.
4. Wählen Sie Delete (Löschen).

   Wenn alle mit einem DB-Cluster verknüpften DB-Instances gelöscht sind, wird der DB-Cluster automatisch gelöscht.

(Optional) Löschen Sie die EC2 Instance und den DB-Cluster, die mit erstellt wurden CloudFormation

Wenn Sie früher CloudFormation Ressourcen erstellt haben, löschen Sie den CloudFormation Stack, nachdem Sie sich mit der EC2 Beispiel-Instance und dem DB-Cluster verbunden und diese erkundet haben, sodass Ihnen diese nicht mehr in Rechnung gestellt werden.

Um die CloudFormation Ressourcen zu löschen

1. Öffnen Sie die CloudFormation Konsole.
2. Wählen Sie auf der Seite Stacks in der CloudFormation Konsole den Root-Stack aus (den Stack ohne Namen VPCStack BastionStack oder APGNS).
3. Wählen Sie Löschen aus.
4. Wenn Sie zur Bestätigung aufgefordert werden, wählen Sie Stack löschen aus.

Weitere Informationen zum Löschen eines Stacks in CloudFormation finden Sie im Benutzerhandbuch unter Löschen eines Stacks auf der CloudFormationAWS CloudFormation Konsole.

(Optional) Verbinden Sie Ihren DB-Cluster mit einer Lambda-Funktion

Sie können Ihren DB-Cluster von Aurora PostgreSQL auch mit einer Lambda-Serverless-Rechenressource verbinden. Mit Lambda-Funktionen können Sie Code ausführen, ohne die Infrastruktur bereitstellen oder verwalten zu müssen. Eine Lambda-Funktion ermöglicht es Ihnen auch, automatisch auf Codeausführungsanfragen jeder Größenordnung zu reagieren, von einem Dutzend Ereignissen pro Tag bis hin zu Hunderten von Ereignissen pro Sekunde. Weitere Informationen finden Sie unter Automatisches Verbinden einer Lambda-Funktion mit einem Aurora-DB-Cluster.

Tutorial: Erstellen eines Webservers und einer eines Amazon Aurora-DB-Clusters

Dieses Tutorial veranschaulicht, wie Sie einen Apache-Webserver mit PHP installieren und eine MariaDB-, MySQL- oder PostgreSQL-Datenbank erstellen. Der Webserver wird auf einer Amazon-EC2-Instance unter Verwendung von Amazon Linux 2023 ausgeführt und Sie können zwischen einem Aurora-MySQL- oder Aurora-PostgreSQL-DB-Cluster wählen. Die Amazon EC2-Instance und die / der DB-Cluster werden beide in einer virtuellen privaten Cloud (VPC) auf der Basis des Amazon VPC-Service ausgeführt.

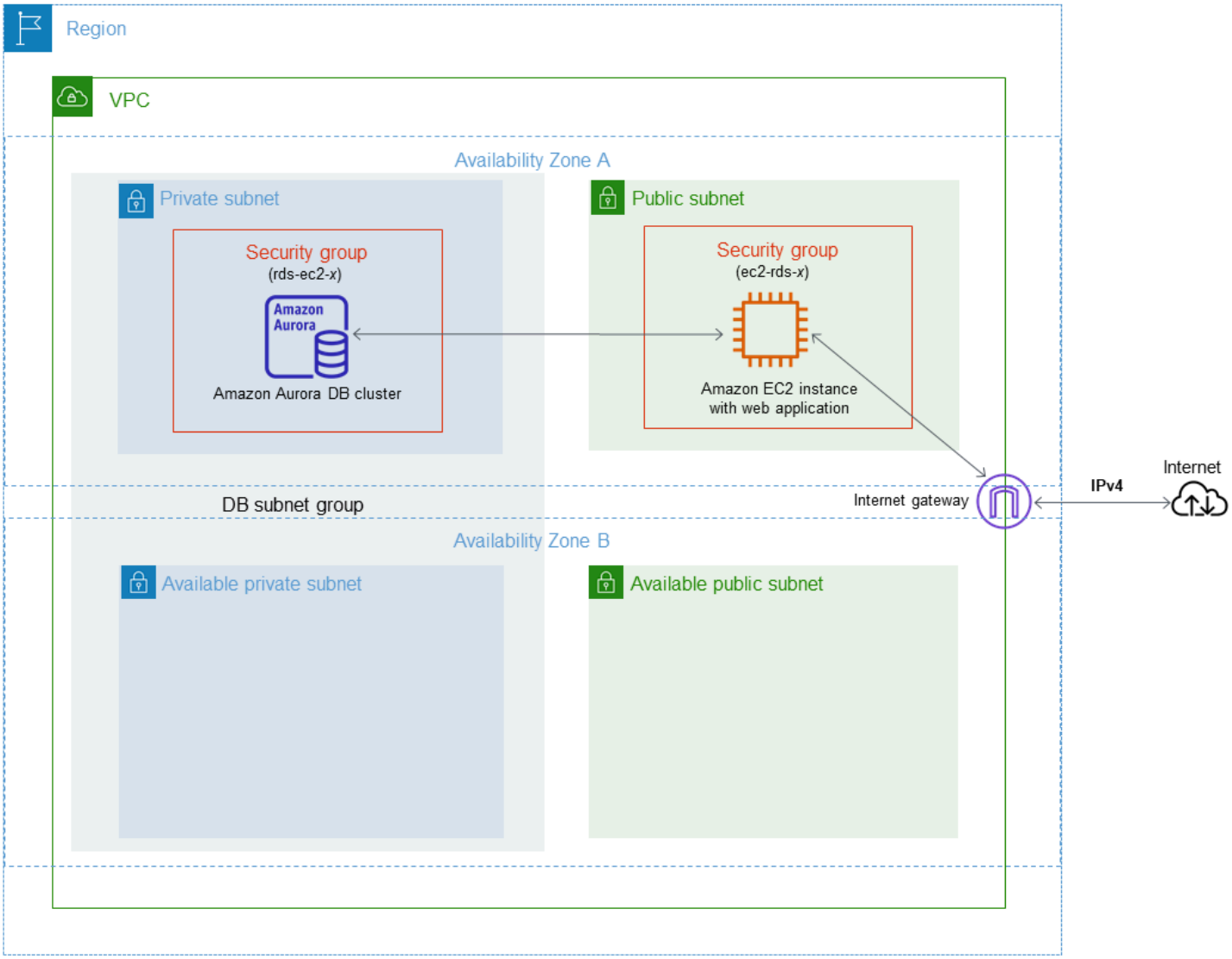
> **Important**
>
> Die Einrichtung eines AWS-Kontos ist kostenlos. Bei Durchführung dieses Tutorials können jedoch Kosten für die von Ihnen verwendeten AWS-Ressourcen anfallen. Sie können diese Ressourcen nach Abschluss des Tutorials löschen, wenn sie nicht mehr benötigt werden.

> **Note**
>
> Dieses Tutorial funktioniert mit Amazon Linux 2023 und möglicherweise nicht mit anderen Linux-Versionen.

Im folgenden Tutorial erstellen Sie eine EC2-Instance, die die Standard-VPC, Subnetze und die Sicherheitsgruppe für Ihr AWS-Konto verwendet. In diesem Tutorial erhalten Sie Informationen zum Erstellen des DB-Clusters und richten die Verbindung mit der EC2-Instance, die Sie erstellt haben, automatisch ein. Das Tutorial zeigt Ihnen dann, wie Sie den Webserver auf der EC2-Instance installieren. Sie verbinden Ihren Webserver in der VPC mit Ihrem DB-Cluster in der VPC mithilfe des DB-Cluster-Writer-Endpunkts.

1. Starten Sie eine EC2-Instance, um eine Verbindung mit Ihrem DB-Cluster herzustellen.
2. Erstellen eines Amazon Aurora-DB-Clusters
3. Installieren eines Webservers auf Ihrer EC2-Instance

Das folgende Diagramm zeigt die Konfiguration nach Abschluss des Tutorials.



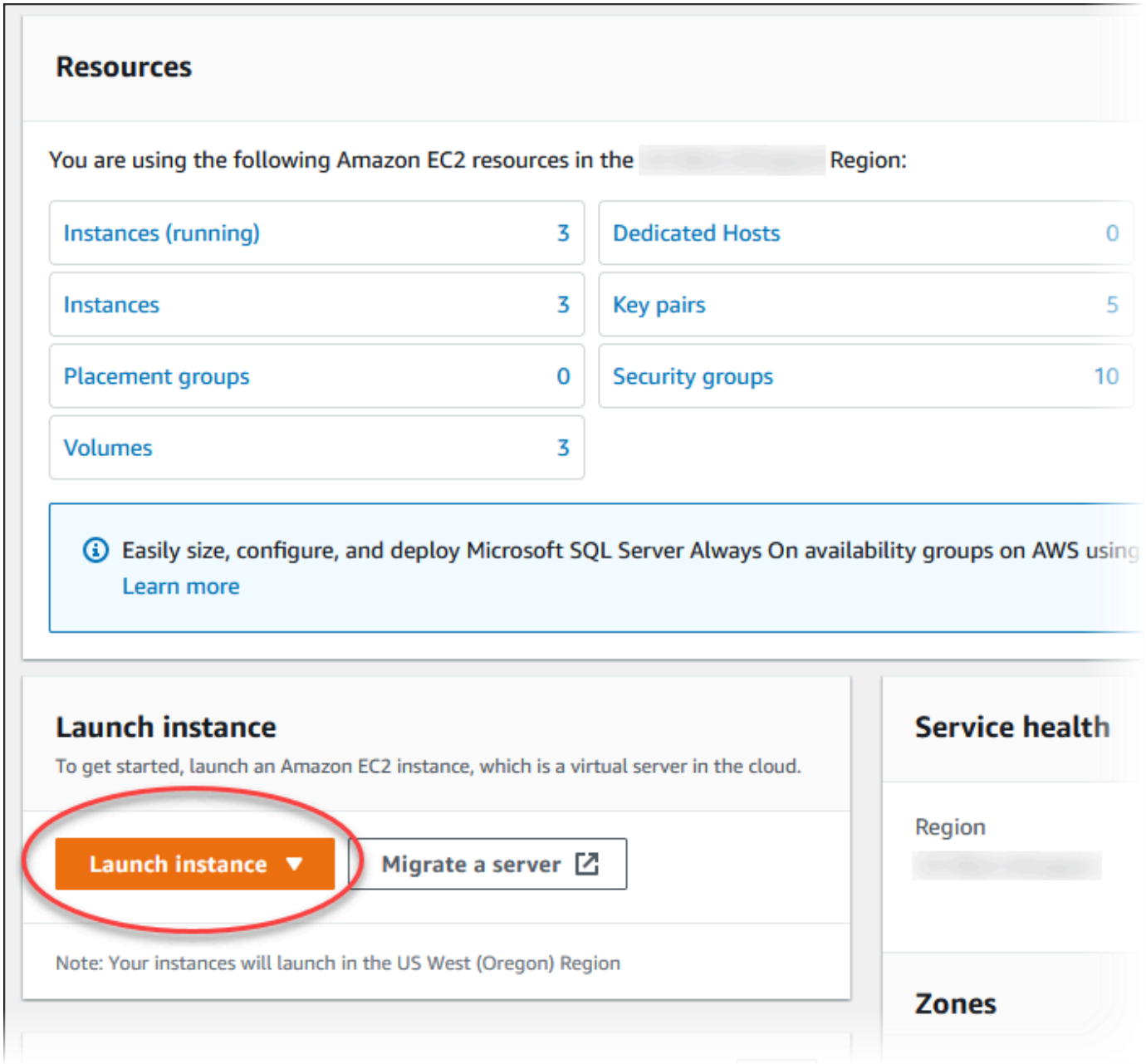
> **Note**
>
> Nach Abschluss des Tutorials gibt es in jeder Availability Zone im VPC ein öffentliches und ein privates Subnetz. Dieses Tutorial verwendet die Standard-VPC für Ihr AWS-Kontound richtet automatisch die Verbindung zwischen Ihrer EC2-Instance und dem DB-Cluster ein. Wenn Sie stattdessen lieber eine neue VPC für dieses Szenario konfigurieren möchten, führen Sie die Aufgaben in Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4) aus.

Starten Sie eine EC2-Instance, um eine Verbindung mit Ihrem DB-Cluster herzustellen.

Erstellen Sie im öffentlichen Subnetz Ihrer VPC eine Amazon-EC2-Instance.

Starten Sie EC2-Instances wie folgt:

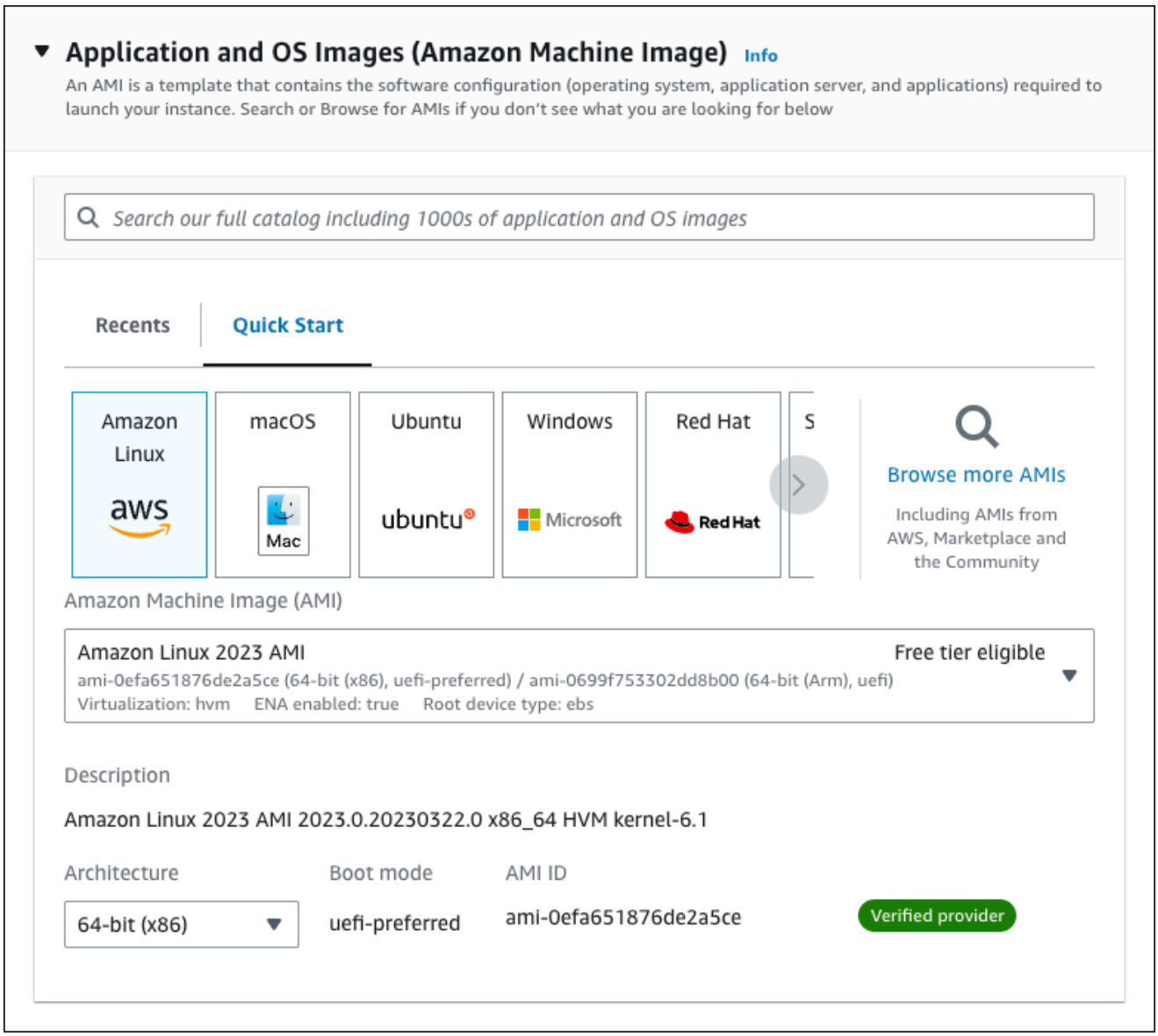
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/EC2/.
2. Wählen Sie in der oberen rechten Ecke der AWS-Managementkonsole die AWS-Region aus, in der Sie die DB-Instance erstellen möchten.
3. Wählen Sie EC2-Dashboard und anschließend Instance starten wie im Folgenden gezeigt.



4. Wählen Sie auf der Seite Eine Instance starten die folgenden Einstellungen aus.

   a. Geben Sie unter Name and tags (Name und Tags) als Name den Namen **`tutorial-ec2-instance-web-server`** ein.

   b. Wählen Sie unter Anwendungs- und Betriebssystem-Images (Amazon Machine Image) die Option Amazon Linux und dann die Registerkarte Amazon Linux 2023 AMI aus. Übernehmen Sie für alle anderen Einstellungen die Standardwerte.



c. Wählen Sie unter Instance type (Instance-Typ) den Wert t2.micro aus.

d. Wählen Sie unter Key pair (login) (Schlüsselpaar (Anmeldung)) einen Key pair name (Schlüsselpaarname), um ein vorhandenes Schlüsselpaar zu verwenden. Wenn Sie ein neues Schlüsselpaar für die Amazon-EC2-Instance erstellen möchten, wählen Sie Create new key pair (Neues Schlüsselpaar erstellen) aus und erstellen sie das Schlüsselpaar im Fenster Create key pair (Schlüsselpaar erstellen).

   Weitere Informationen zum Erstellen eines neuen Schlüsselpaars finden Sie unter Erstellen eines Schlüsselpaars im Amazon-EC2-Benutzerhandbuch.

e. Legen Sie unter Network settings (Netzwerkeinstellungen) die folgenden Werte fest und übernehmen Sie für die anderen Einstellungen die Standardwerte:

- Wählen Sie für Allow SSH traffic from (SSH-Verkehr zulassen von) die Quelle von SSH-Verbindungen mit der EC2-Instance aus.

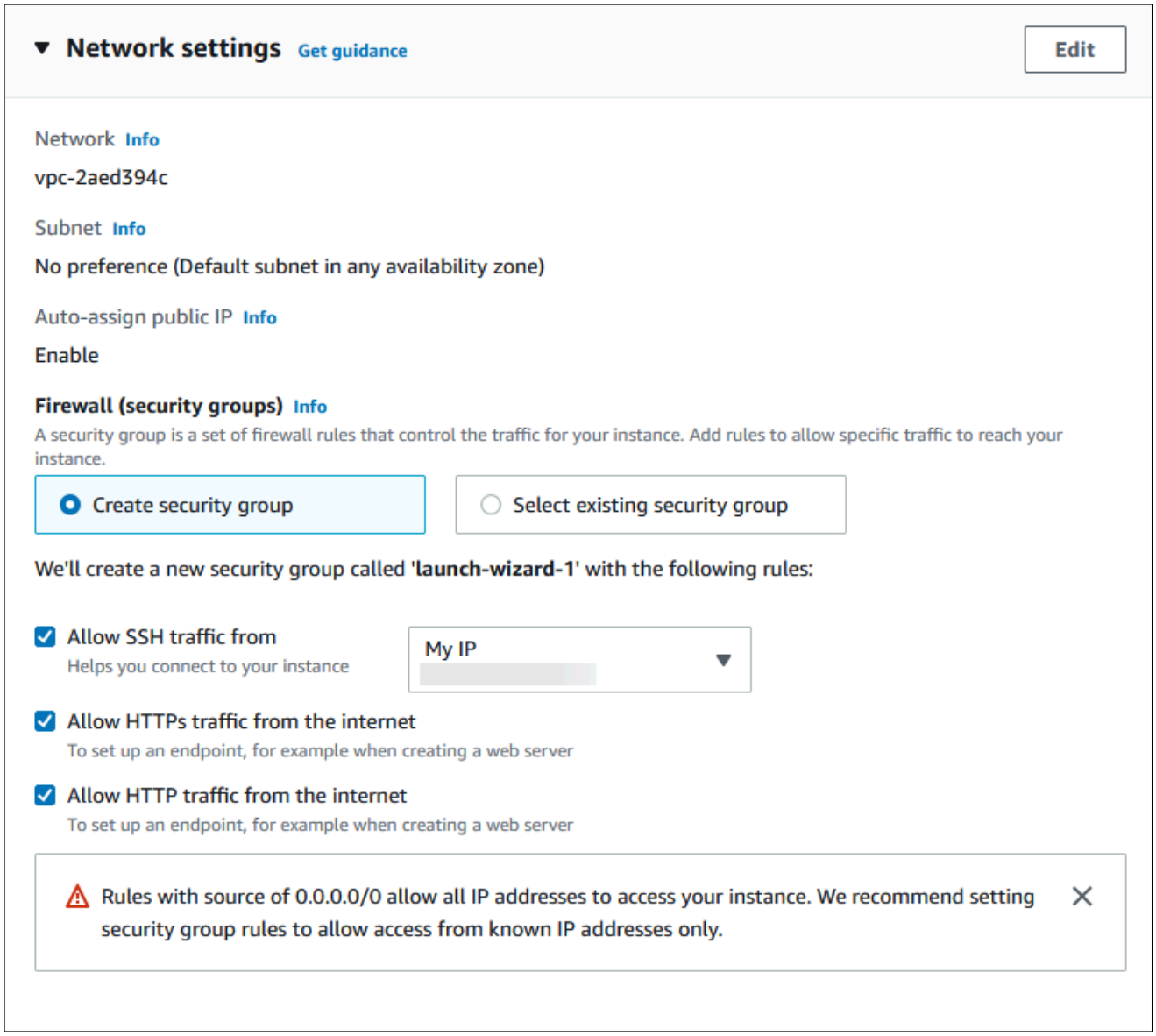
  Sie können My IP (Meine IP) auswählen, wenn die angezeigte IP-Adresse für SSH-Verbindungen korrekt ist.

  Andernfalls können Sie die IP-Adresse, die für die Verbindung mit EC2-Instances in Ihrer VPC verwendet werden soll, mit Secure Shell (SSH) ermitteln. Um Ihre öffentliche IP-Adresse zu ermitteln, können Sie in einem anderen Browserfenster oder einer anderen Registerkarte den Service unter https://checkip.amazonaws.com verwenden. Ein Beispiel für eine IP-Adresse ist `203.0.113.25/32`.

  In vielen Fällen können Sie eine Verbindung über einen Internetdienstanbieter (ISP) oder hinter Ihrer Firewall ohne statische IP-Adresse herstellen. Bestimmen Sie in diesem Fall den Bereich der IP-Adressen, die von Client-Computern verwendet werden.

  > **Warning**
  >
  > Wenn Sie `0.0.0.0/0` für SSH-Zugriff verwenden, ermöglichen Sie für alle IP-Adressen den Zugriff auf Ihre öffentlichen Instances. Dieser Ansatz ist zwar für kurze Zeit in einer Testumgebung zulässig, aber für Produktionsumgebungen sehr unsicher. Für die Produktion wird nur eine bestimmte IP-Adresse bzw. ein bestimmter Adressbereich für den Zugriff auf Ihre Instances autorisiert.

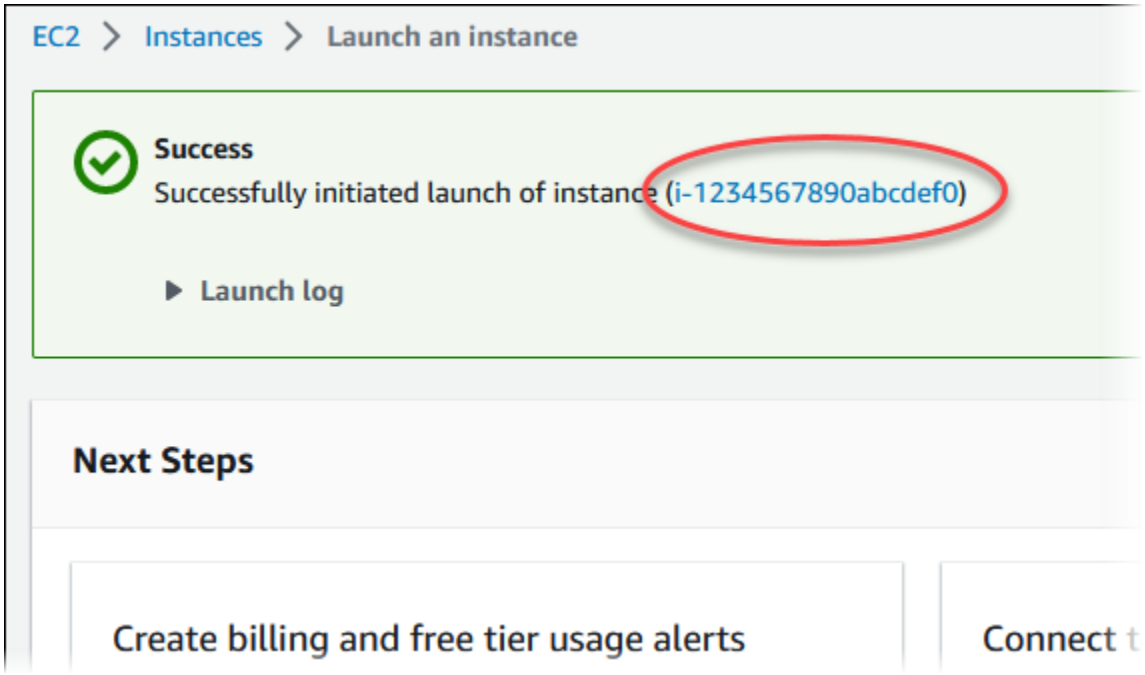
- Aktivieren Sie Allow HTTPs traffic from the internet (HTTPs-Verkehr aus dem Internet zulassen).
- Aktivieren Sie Allow HTTP traffic from the internet (HTTP-Verkehr aus dem Internet zulassen).



f. Übernehmen Sie die Standardwerte für die übrigen Abschnitte.

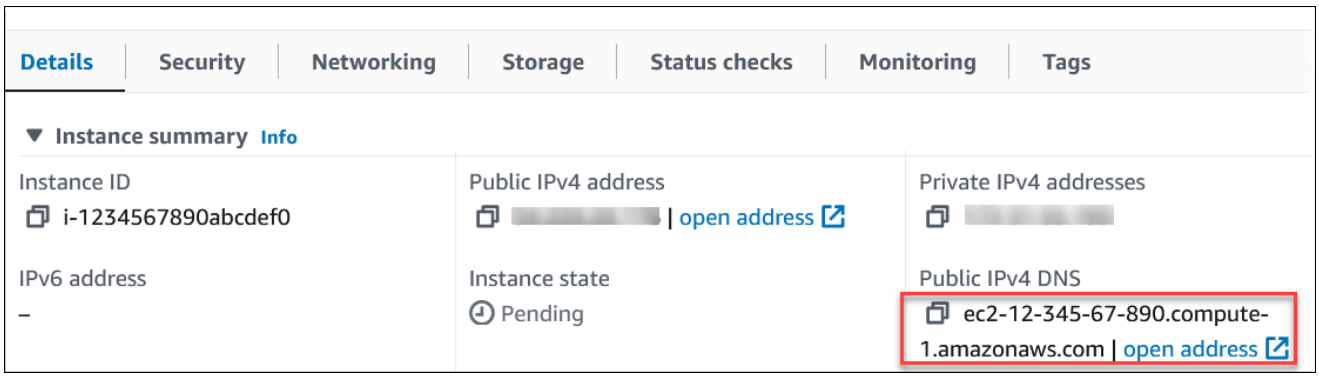
g. Überprüfen Sie Ihre Instance-Konfiguration im Bereich Summary (Übersicht). Wenn alles in Ordnung ist, klicken Sie auf Launch instance (Instance starten).

5. Notieren Sie auf der im Folgenden gezeigten Seite Startstatus die Kennung für die neue EC2-Instance, beispielsweise: `i-1234567890abcdef0`.

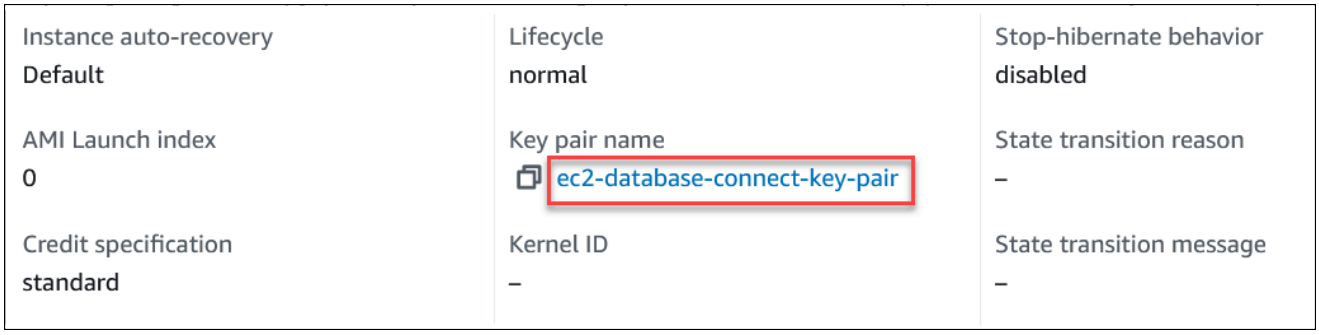


6. Wählen Sie die EC2-Instance-Kennung aus, um die Liste der EC2-Instances zu öffnen. Wählen Sie dann Ihre EC2-Instance aus.
7. Notieren Sie sich die folgenden Werte auf der Registerkarte Details. Diese benötigen Sie, wenn Sie eine Verbindung über SSH herstellen:

   a. Notieren Sie sich unter Instance-Zusammenfassung den Wert für Public IPv4 DNS.



   b. Notieren Sie sich unter Instance-Details den Wert für Schlüsselpaarname.



8. Warten Sie, bis der Instance state (Instance-Status) für Ihre Instance als Running (Wird ausgeführt) angezeigt wird, bevor Sie fortfahren.
9. Schließen Sie Erstellen eines Amazon Aurora-DB-Clusters ab.

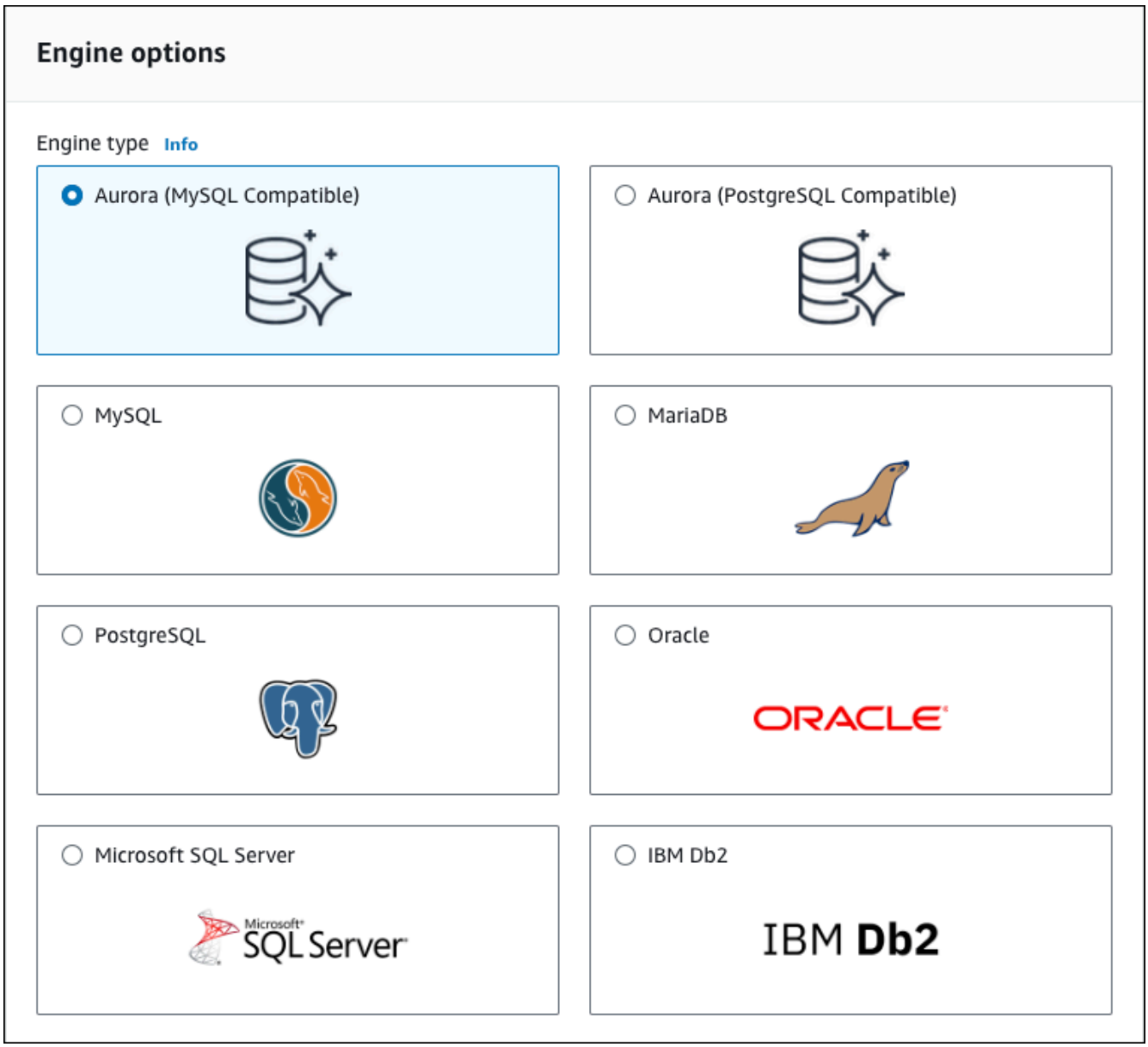
Erstellen eines Amazon Aurora-DB-Clusters

Erstellen eines DB-Clusters von Amazon Aurora MySQL oder Aurora PostgreSQL, der die von einer Webanwendung verwendeten Daten enthält.

Aurora MySQL

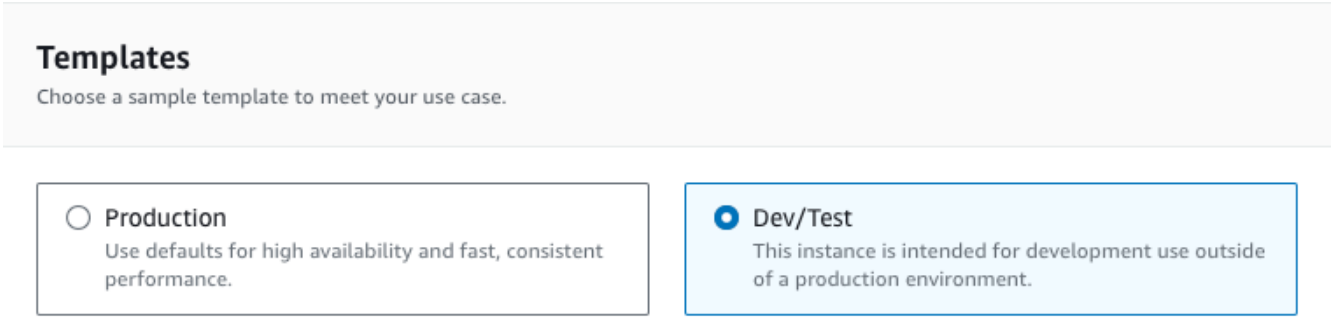
Erstellen eines Aurora MySQL-DB-Clusters

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Vergewissern Sie sich, dass in der oberen rechten Ecke desAWS-Managementkonsole derselbe AWS-Region steht wie der, an dem Sie Ihre EC2-Instance erstellt haben.
3. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
4. Wählen Sie Create database (Datenbank erstellen) aus.
5. Wählen Sie auf der Seite Datenbank erstellen die Option Standarderstellung aus.
6. Wählen Sie unter Engine-Optionen die Option Aurora (MySQL-kompatibel) aus.



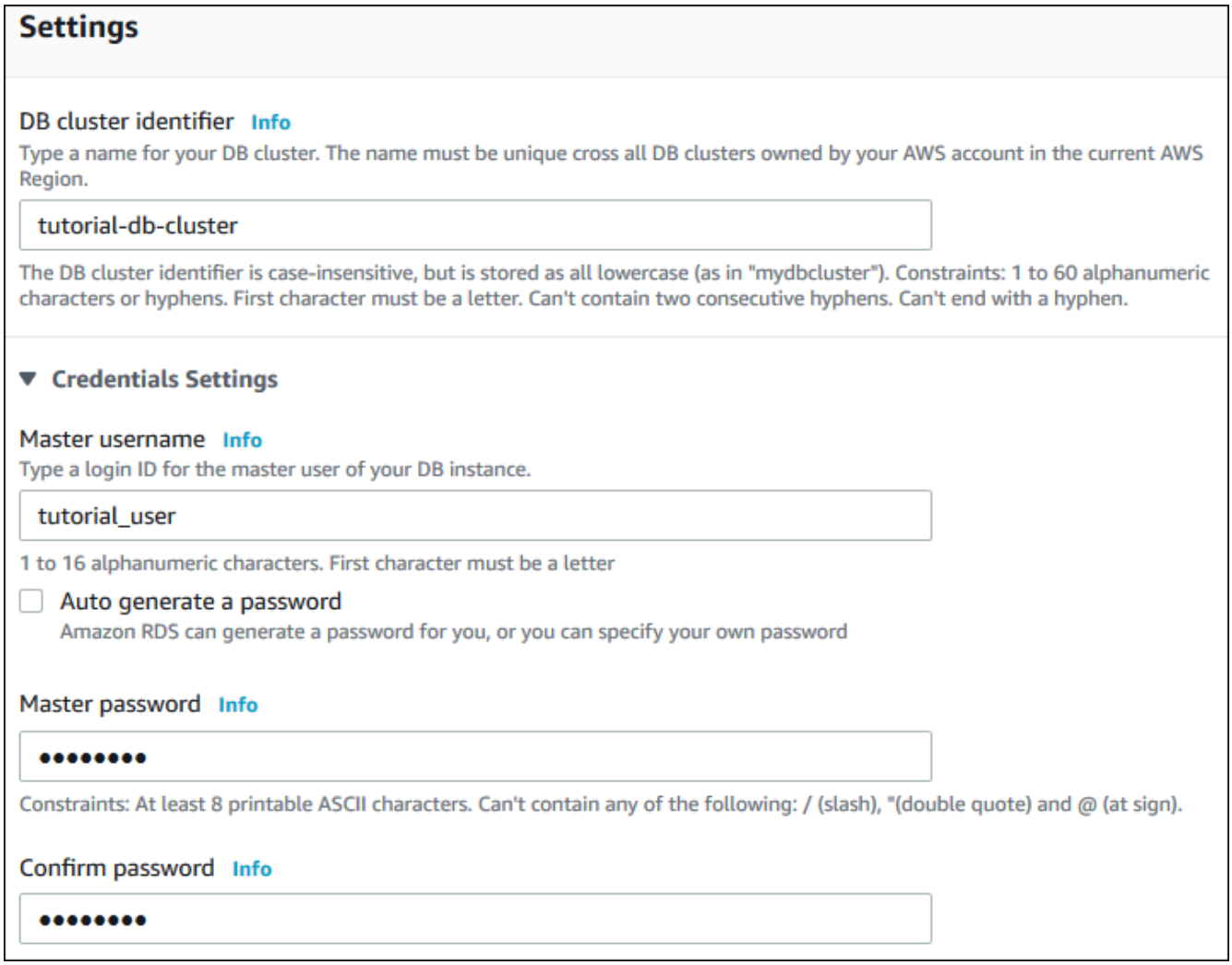
Behalten Sie die Standardwerte für Version und die anderen Engine-Optionen bei.

7. Wählen Sie unter Templates (Vorlagen) die Option Dev/Test (Entwicklung/Testing).



8. Legen Sie im Abschnitt Settings (Einstellungen) die folgenden Werte fest:

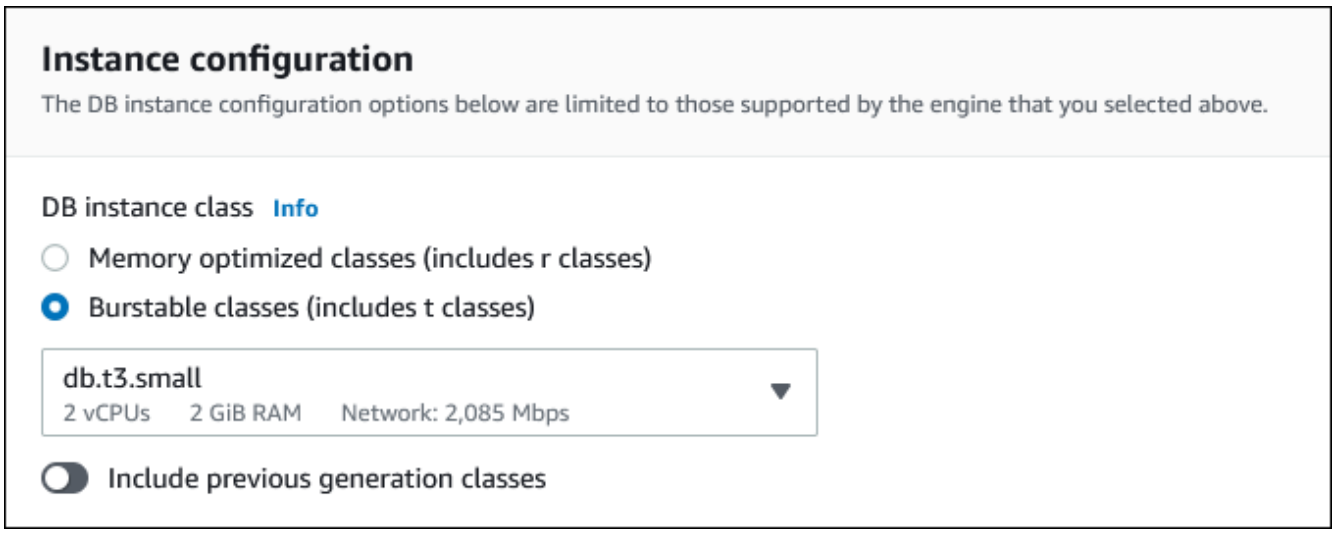
   - DB cluster identifier (DB-Cluster-Kennung) – Geben Sie **`tutorial-db-cluster`** ein.
   - Master username (Masterbenutzername) – Typ **`tutorial_user`**.
   - Automatisch ein Passwort generieren - Lassen Sie die Option ausgeschaltet.
   - Master-Passwort - Geben Sie ein Passwort ein.
   - Confirm password (Passwort bestätigen): Geben Sie das Passwort erneut ein.



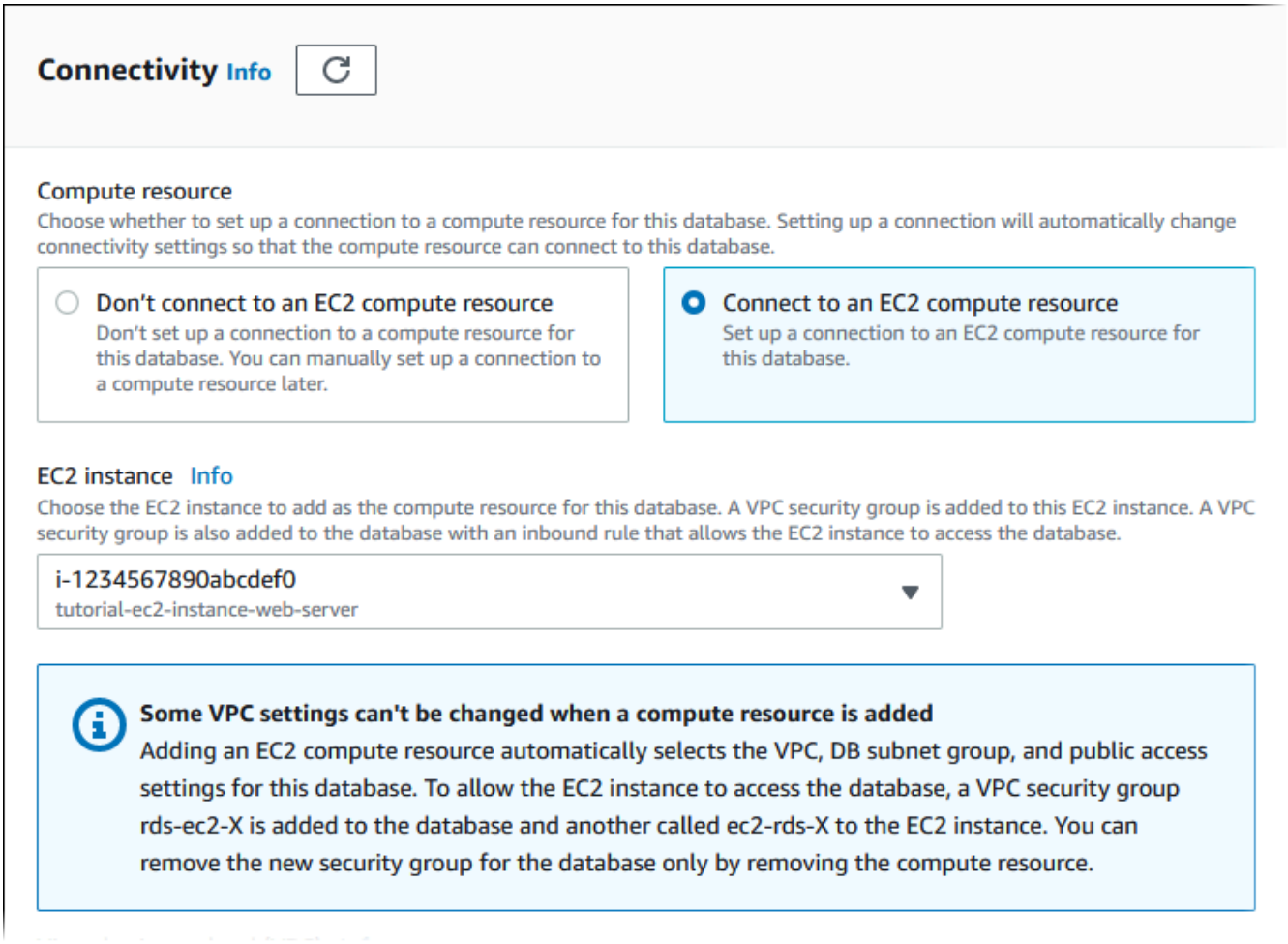
9. Legen Sie im Abschnitt Instance Configuration folgende Werte fest:

   - Burst-fähige Klassen (einschließlich t-Klassen)
   - db.t3.small oder db.t3.medium

> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-Instance-Klassenarten.

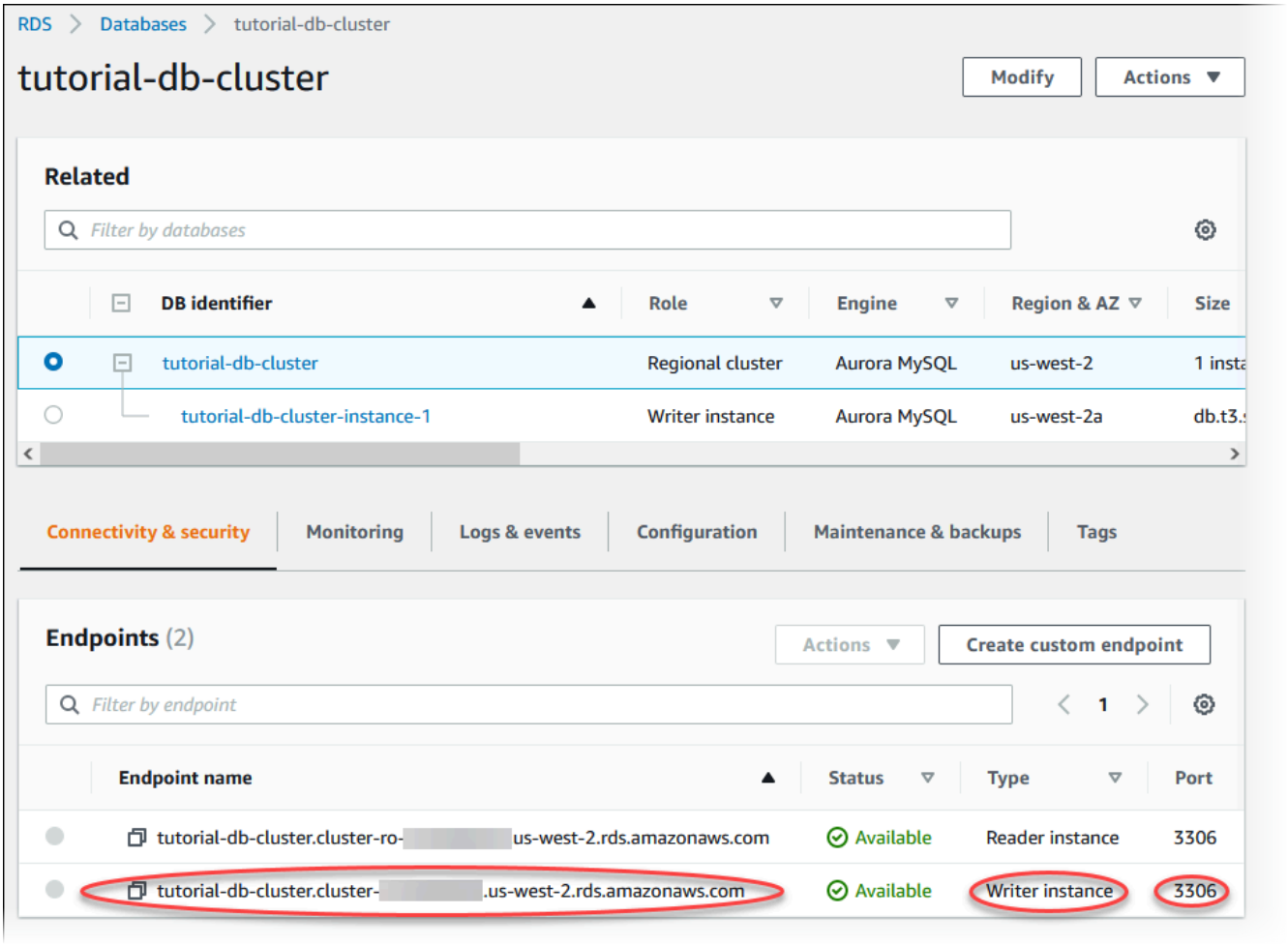


10. Verwenden Sie im Abschnitt Availability and durability (Verfügbarkeit und Beständigkeit) die Standardwerte.
11. Legen Sie im Abschnitt Connectivity (Konnektivität) die folgenden Werte fest und übernehmen Sie für die anderen Einstellungen die Standardwerte:
    - Wählen Sie unter Compute-Ressource die Option Connect to an EC2 compute resource (Verbinden mit einer EC2 Compute-Ressource).
    - Als EC2-Instance wählen Sie die EC2-Instance, die Sie zuvor erstellt haben, z. B. tutorial-ec2-instance-web-server.



12. Öffnen Sie den Abschnitt Additional configuration (Zusätzliche Konfiguration) und geben Sie **`sample`** für Initial database name (Erster Datenbankname) ein. Behalten Sie für die anderen Optionen die Standardeinstellungen bei.
13. Zum Erstellen Ihres Aurora MySQL-DB-Clusters wählen Sie Create database (Datenbank erstellen).

    Ihr neuer DB-Cluster wird in der Liste Databases (Datenbanken) mit dem Status Creating (Wird erstellt) angezeigt.
14. Warten Sie, bis der Status Ihres neuen DB-Clusters als Available (Verfügbar) angezeigt wird. Wählen Sie dann den Namen des DB-Clusters aus, um dessen Details anzuzeigen.
15. Zeigen Sie im Abschnitt Connectivity & security (Konnektivität und Sicherheit) den Endpoint (Endpunkt) und den Port der DB-Writer-Instance an.



Notieren Sie den Endpunkt und den Port Ihrer DB-Writer-Instance. Sie verwenden diese Informationen, um Ihren Webserver mit Ihrem DB-Cluster zu verbinden.

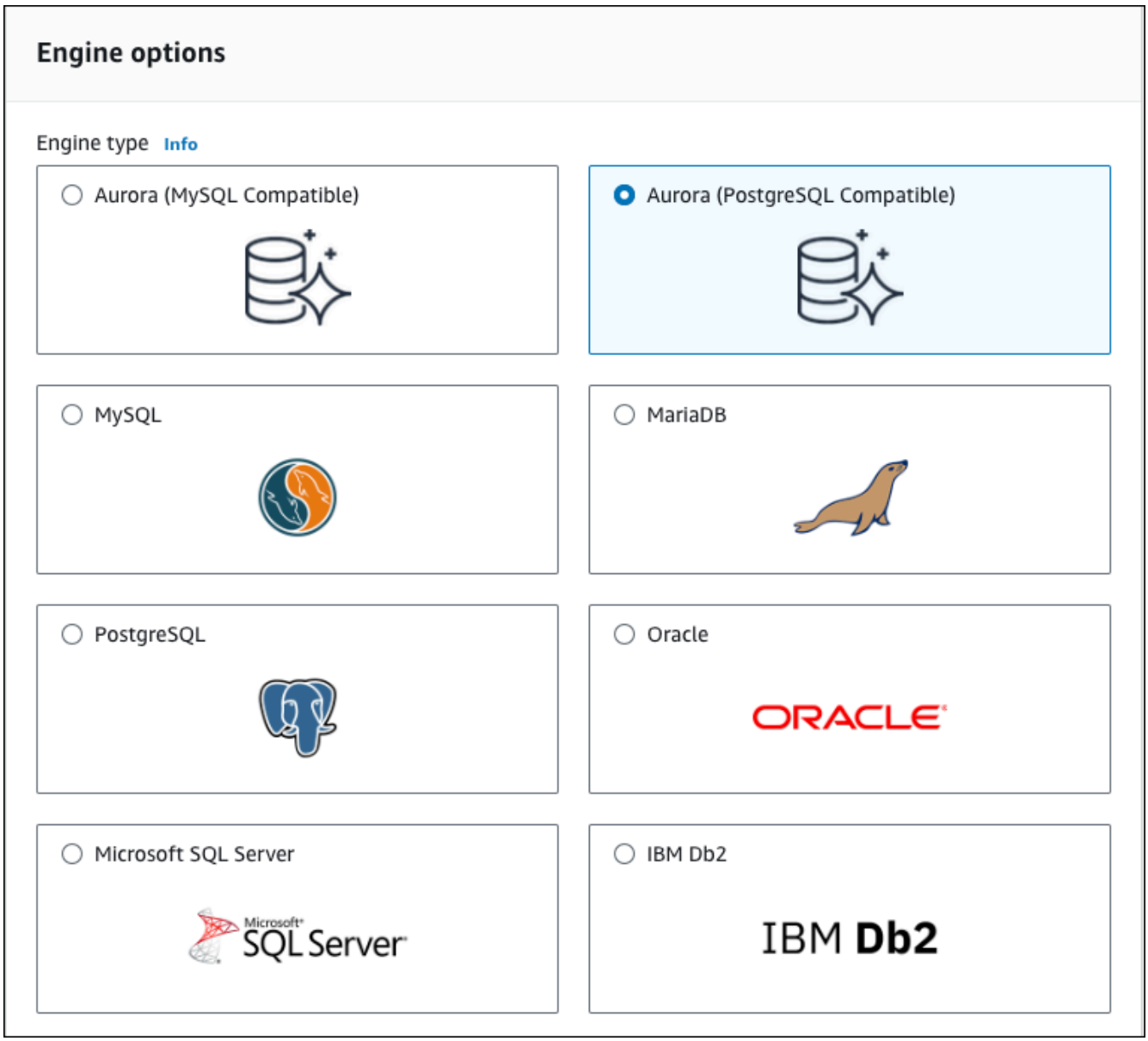
16. Schließen Sie Installieren eines Webservers auf Ihrer EC2-Instance ab.

Aurora PostgreSQL

Erstellen eines Aurora PostgreSQL-DB-Clusters

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Vergewissern Sie sich, dass in der oberen rechten Ecke desAWS-Managementkonsole derselbe AWS-Region steht wie der, an dem Sie Ihre EC2-Instance erstellt haben.
3. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
4. Wählen Sie Create database (Datenbank erstellen) aus.

5. Wählen Sie auf der Seite Datenbank erstellen die Option Standarderstellung aus.
6. Wählen Sie unter Engine-Optionen Aurora (PostgreSQL-kompatibel).

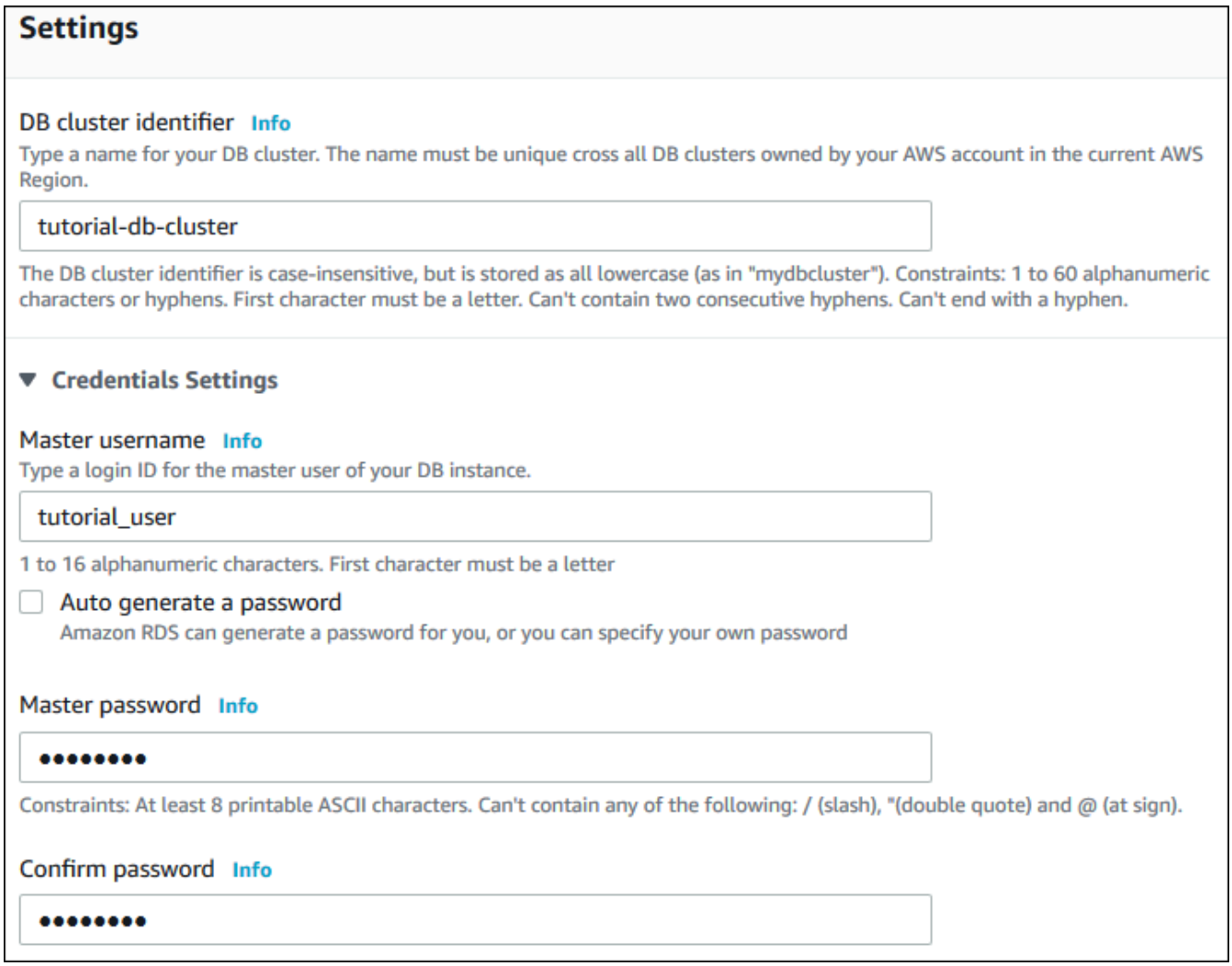


Behalten Sie die Standardwerte für Version und die anderen Engine-Optionen bei.

7. Wählen Sie unter Templates (Vorlagen) die Option Dev/Test (Entwicklung/Testing).

8. Legen Sie im Abschnitt Settings (Einstellungen) die folgenden Werte fest:

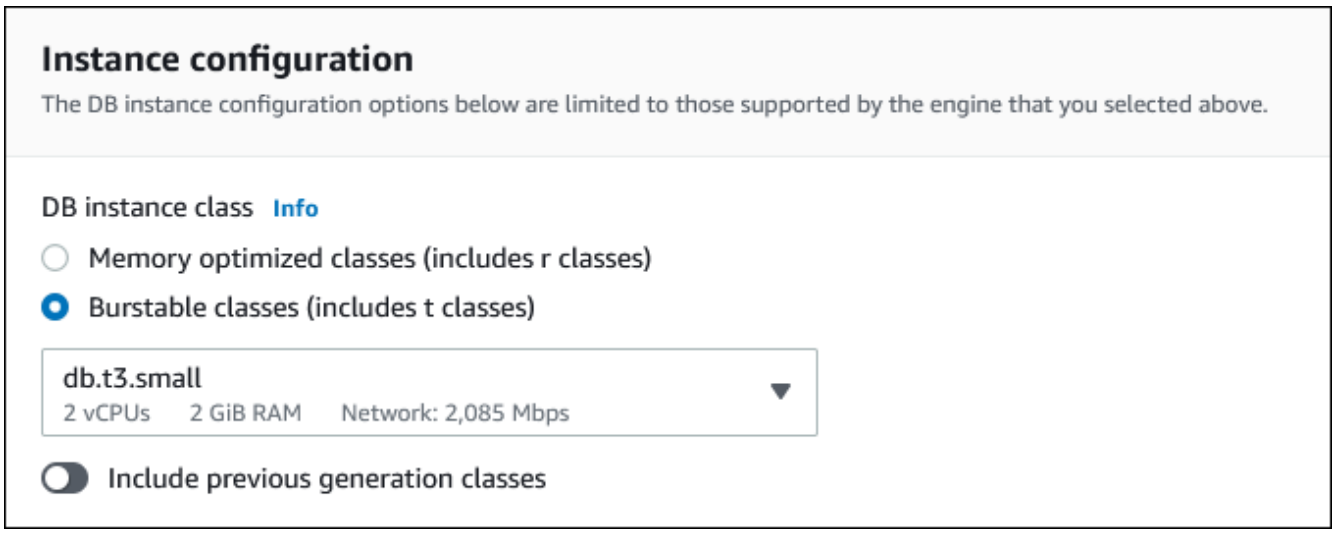
   - DB cluster identifier (DB-Cluster-Kennung) – Geben Sie **`tutorial-db-cluster`** ein.
   - Master username (Masterbenutzername) – Typ **`tutorial_user`**.
   - Automatisch ein Passwort generieren - Lassen Sie die Option ausgeschaltet.
   - Master-Passwort - Geben Sie ein Passwort ein.
   - Confirm password (Passwort bestätigen): Geben Sie das Passwort erneut ein.



9. Legen Sie im Abschnitt Instance Configuration folgende Werte fest:

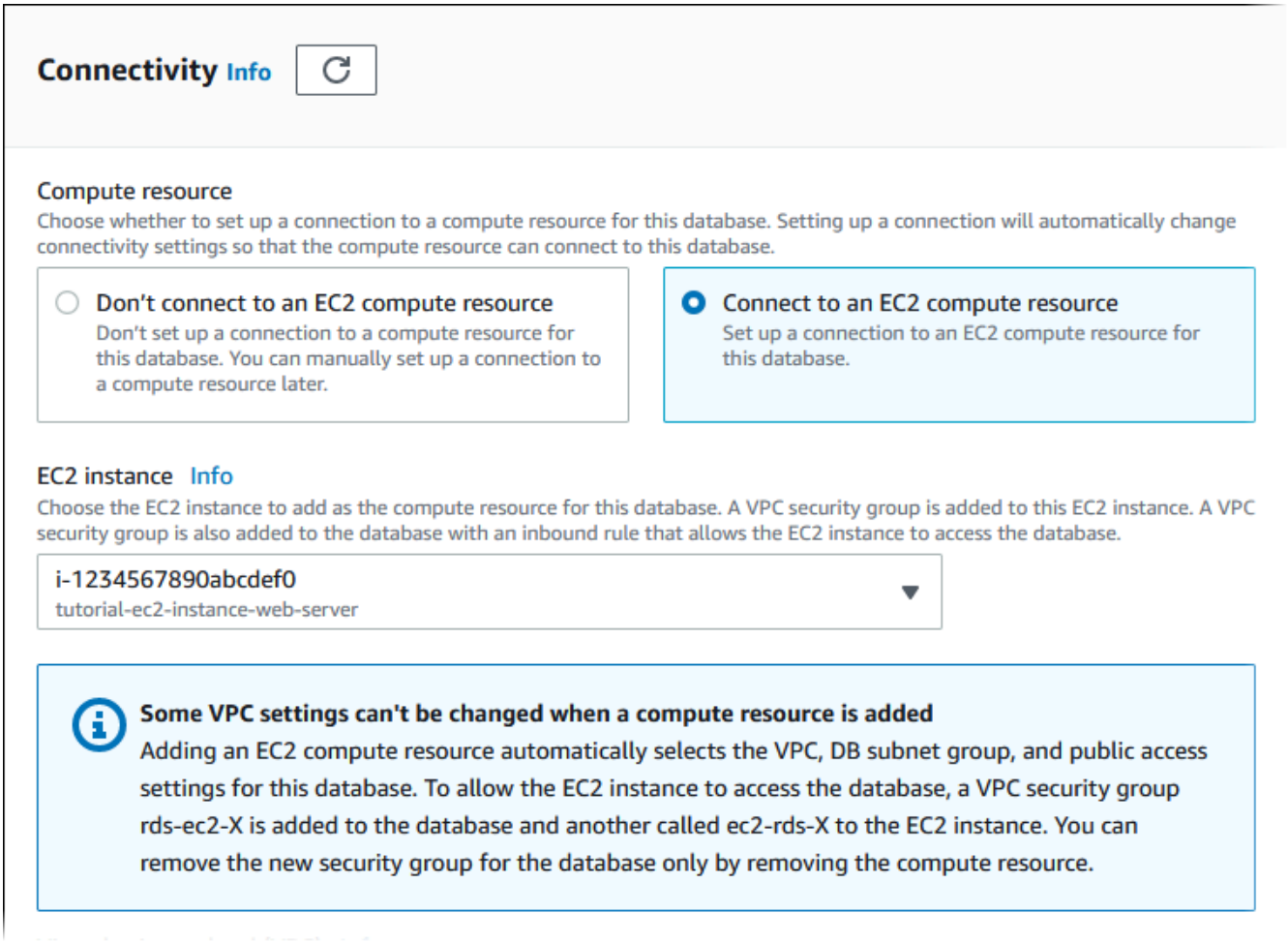
   - Burst-fähige Klassen (einschließlich t-Klassen)
   - db.t3.small oder db.t3.medium

   > **Note**
   >
   > Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-Instance-Klassenarten.



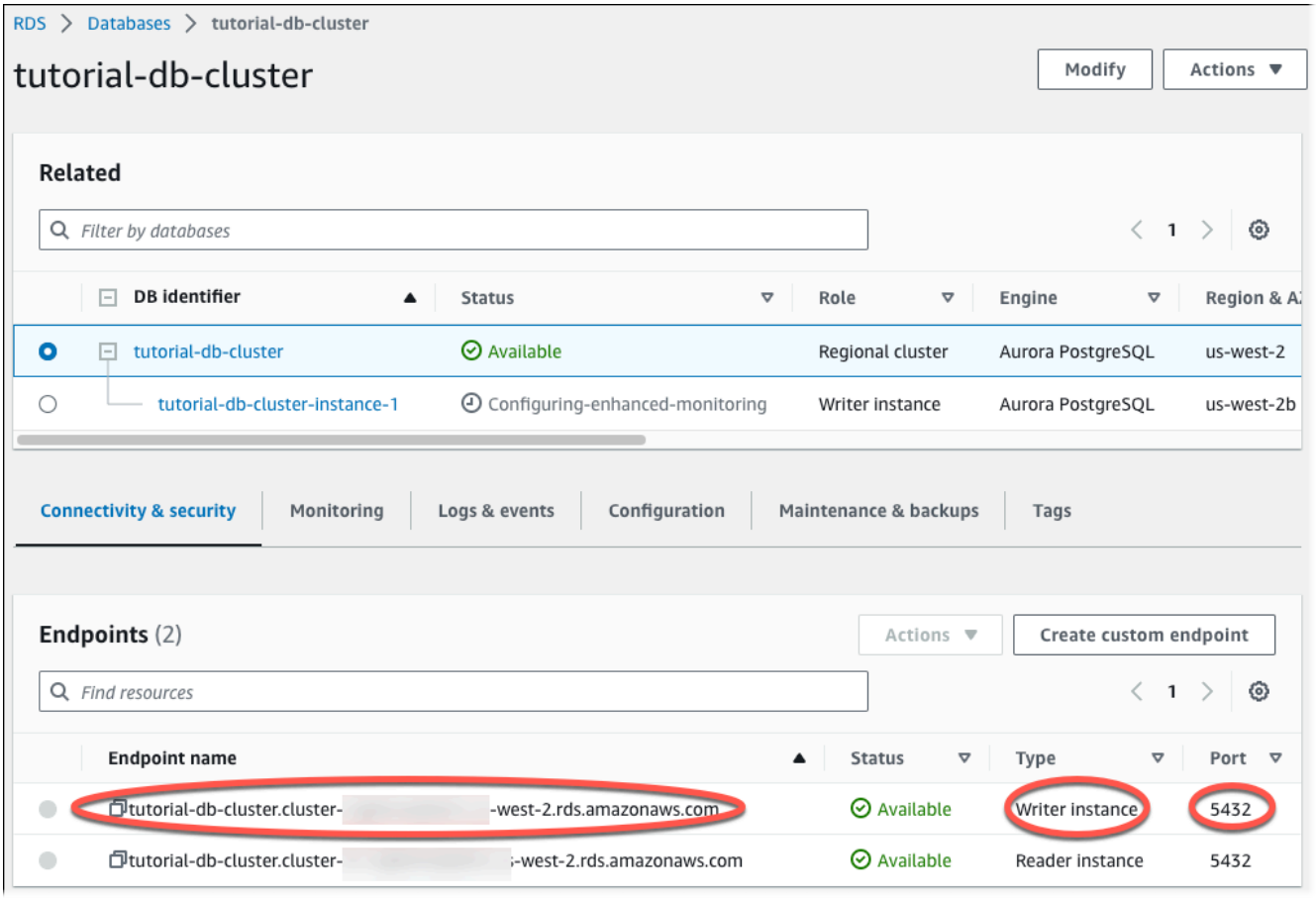
10. Verwenden Sie im Abschnitt Availability and durability (Verfügbarkeit und Beständigkeit) die Standardwerte.
11. Legen Sie im Abschnitt Connectivity (Konnektivität) die folgenden Werte fest und übernehmen Sie für die anderen Einstellungen die Standardwerte:

    - Wählen Sie unter Compute-Ressource die Option Connect to an EC2 compute resource (Verbinden mit einer EC2 Compute-Ressource).
    - Als EC2-Instance wählen Sie die EC2-Instance, die Sie zuvor erstellt haben, z. B. tutorial-ec2-instance-web-server.



12. Öffnen Sie den Abschnitt Additional configuration (Zusätzliche Konfiguration) und geben Sie **`sample`** für Initial database name (Erster Datenbankname) ein. Behalten Sie für die anderen Optionen die Standardeinstellungen bei.
13. Zum Erstellen Ihres Aurora PostgreSQL-DB-Clusters wählen Sie Datenbank erstellen.

    Ihr neuer DB-Cluster wird in der Liste Databases (Datenbanken) mit dem Status Creating (Wird erstellt) angezeigt.
14. Warten Sie, bis der Status Ihres neuen DB-Clusters als Available (Verfügbar) angezeigt wird. Wählen Sie dann den Namen des DB-Clusters aus, um dessen Details anzuzeigen.
15. Zeigen Sie im Abschnitt Connectivity & security (Konnektivität und Sicherheit) den Endpoint (Endpunkt) und den Port der DB-Writer-Instance an.



Notieren Sie den Endpunkt und den Port Ihrer DB-Writer-Instance. Sie verwenden diese Informationen, um Ihren Webserver mit Ihrem DB-Cluster zu verbinden.

16. Schließen Sie a Installieren eines Webservers auf Ihrer EC2-Instance.

Installieren eines Webservers auf Ihrer EC2-Instance

Installieren Sie einen Webserver auf der EC2-Instance, die Sie in Starten Sie eine EC2-Instance, um eine Verbindung mit Ihrem DB-Cluster herzustellen. erstellt haben. Der Webserver verbindet sich mit dem DB-Cluster von Amazon Aurora, den Sie in Erstellen eines Amazon Aurora-DB-Clusters erstellt haben.

Installieren eines Apache-Webservers mit PHP und MariaDB

Stellen Sie eine Verbindung mit Ihrer EC2-Instance her und installieren Sie den Webserver.

So stellen Sie eine Verbindung mit Ihrer EC2-Instance her und installieren den Apache-Webserver mit PHP

1. Stellen Sie eine Verbindung mit der von Ihnen zuvor erstellten EC2-Instance her, indem Sie die Schritte unter Herstellen einer Verbindung zur Linux-Instance im Amazon-EC2-Benutzerhandbuch befolgen.

   Wir empfehlen, dass Sie eine Verbindung mit Ihrer EC2-Instance mithilfe von SSH herstellen. Wenn das SSH-Client-Dienstprogramm unter Windows, Linux oder Mac installiert ist, können Sie mit dem folgenden Befehlsformat eine Verbindung mit der Instance herstellen:

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   Nehmen wir zum Beispiel an, das `ec2-database-connect-key-pair.pem` in `/dir1` unter Linux gespeichert und das öffentliche IPv4-DNS für Ihre EC2-Instance `ec2-12-345-678-90.compute-1.amazonaws.com` ist. Ihr SSH-Befehl würde wie folgt aussehen:

   ```
   ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
   user@ec2-12-345-678-90.compute-1.amazonaws.com
   ```

2. Installieren Sie die neuesten Bugfixes und Sicherheitsupdates, indem Sie die Software auf Ihrer EC2-Instance aktualisieren. Verwenden Sie dazu den folgenden Befehl.

   > **Note**
   >
   > Mit der Option `-y` werden die Updates installiert, ohne um Bestätigung zu bitten. Um Updates vor der Installation zu überprüfen, lassen Sie diese Option aus.

   ```
   sudo dnf update -y
   ```

3. Nachdem die Aktualisierungen abgeschlossen sind, installieren Sie die Software von Apache-Webserver, PHP und MariaDB mit den folgenden Befehlen. Mit diesem Befehl werden gleichzeitig mehrere Softwarepakete und zugehörige Abhängigkeiten installiert.

MariaDB & MySQL

```
sudo dnf install -y httpd php php-mysqli mariadb105
```

PostgreSQL

```
sudo dnf install -y httpd php php-pgsql postgresql15
```

Wenn Sie eine Fehlermeldung erhalten, wurde Ihre Instance wahrscheinlich nicht mit einem Amazon-Linux-2023-AMI gestartet. Stattdessen verwenden Sie möglicherweise das Amazon-Linux-2-AMI. Sie können Ihre Version von Amazon Linux mit dem folgenden Befehl anzeigen.

```
cat /etc/system-release
```

Weitere Informationen finden Sie unter Aktualisieren der Software einer Instance.

4. Starten Sie den Webserver mithilfe des folgenden Befehls.

```
sudo systemctl start httpd
```

Sie können testen, ob Ihr Webserver richtig installiert und gestartet ist. Geben Sie dazu den öffentlichen DNS-Namen (Domain Name System) Ihrer EC2-Instance in die Adressleiste eines Webbrowsers ein, zum Beispiel: `http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`. Wenn Ihr Webserver ausgeführt wird, wird Ihnen die Apache-Testseite angezeigt.

Wenn Sie die Apache-Testseite nicht sehen, überprüfen Sie die Regeln für eingehenden Datenverkehr für die VPC-Sicherheitsgruppe, die Sie in erstellt habe Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4). Stellen Sie sicher, dass die Regeln für eingehenden Datenverkehr eine Regel enthalten, die den HTTP-Zugriff (Port 80) für die IP-Adresse ermöglicht, um eine Verbindung mit dem Webserver herzustellen.

> **Note**
>
> Die Apache-Testseite wird nur angezeigt, wenn im Dokumentstammverzeichnis `/var/www/html` keine Inhalte vorhanden sind. Wenn Sie dem Dokumentstammverzeichnis Inhalte hinzugefügt haben, werden unter der öffentlichen DNS-Adresse Ihrer EC2-

Instance Ihre Inhalte angezeigt. Bis zu diesem Zeitpunkt erscheinen sie auf der Apache-Testseite.

5. Konfigurieren Sie den Webserver so, dass er mit jedem Systemstart gestartet wird. Verwenden Sie hierzu den Befehl `systemctl`.

```
sudo systemctl enable httpd
```

Um für `ec2-user` die Verwaltung von Dateien im Standardstammverzeichnis Ihres Apache-Webservers zuzulassen, ändern Sie die Eigentümerschaft und die Berechtigungen für das Verzeichnis `/var/www`. Es gibt viele Möglichkeiten, um diese Aufgabe zu erfüllen. In diesem Tutorial fügen Sie `ec2-user` zur `apache`-Gruppe hinzu, geben der `apache`-Gruppe Eigentümerschaft über das Verzeichnis `/var/www` und weisen der Gruppe Schreibberechtigungen zu.

So legen Sie Dateiberechtigungen für den Apache-Webserver fest

1. Fügen Sie den Benutzer `ec2-user` zur Gruppe `apache` hinzu.

```
sudo usermod -a -G apache ec2-user
```

2. Melden Sie sich ab, um Ihre Berechtigungen zu aktualisieren und die neue Gruppe `apache` einzufügen.

```
exit
```

3. Melden Sie sich wieder an und überprüfen Sie mit dem Befehl `apache`, ob die Gruppe `groups` vorhanden ist.

```
groups
```

Die Ausgabe sieht folgendermaßen oder ähnlich aus:

```
ec2-user adm wheel apache systemd-journal
```

4. Ändern Sie die Besitzergruppe für das Verzeichnis `/var/www` und dessen Inhalte in die Gruppe `apache`.

```
sudo chown -R ec2-user:apache /var/www
```

5. Ändern Sie die Verzeichnisberechtigungen von `/var/www` und dessen Unterverzeichnissen, indem Sie Schreibberechtigungen für die Gruppe hinzufügen und die Gruppen-ID für zukünftige Unterverzeichnisse einrichten.

```
sudo chmod 2775 /var/www
find /var/www -type d -exec sudo chmod 2775 {} \;
```

6. Ändern Sie die Dateiberechtigungen für Dateien im Verzeichnis `/var/www` und dessen Unterverzeichnissen rekursiv, um Schreibberechtigungen für die Gruppe hinzuzufügen.

```
find /var/www -type f -exec sudo chmod 0664 {} \;
```

Jetzt kann `ec2-user` (und jedes künftige Mitglied der `apache`-Gruppe) im Dokumentstammverzeichnis von Apache Dateien hinzufügen, löschen und bearbeiten. Damit haben Sie die Möglichkeit, Inhalte hinzuzufügen, z. B. eine statische Website oder eine PHP-Anwendung.

> **Note**
>
> Ein Webserver, auf dem HTTP ausgeführt wird, bietet keine Transportsicherheit für die gesendeten oder empfangenen Daten. Wenn Sie über einen Webbrowser eine Verbindung mit einem HTTP-Server herstellen, sind viele Informationen überall auf dem Netzwerkpfad für Lauscher zugänglich. Zu diesen Informationen gehören die URLs, die Sie aufrufen, die Inhalte von Webseiten, die Sie empfangen, und die Inhalte (einschließlich Passwörtern) von HTML-Formularen.
> Die bewährte Methode, Ihren Webserver abzusichern, besteht darin, Unterstützung für HTTPS (HTTP Secure) zu installieren. Dieses Protokoll schützt Ihre Daten mit SSL-/TLS-Verschlüsselung. Weitere Informationen finden Sie unter Tutorial: SSL/TLS mit dem Amazon Linux AMI konfigurieren im Amazon EC2-Benutzerhandbuch .

Herstellen der Verbindung zwischen Ihrem Apache-Webserver und Ihrem DB-Cluster

Als Nächstes fügen Sie Ihrem Apache-Server, der mit Ihrer /dem Amazon Aurora-DB-Cluster verbunden ist, Inhalte hinzu.

So fügen Sie dem Apache-Server, der mit Ihrer /dem DB-Cluster verbunden ist, Inhalte hinzu

1. Während die Verbindung mit Ihrer EC2-Instance besteht, ändern Sie das Verzeichnis in `/var/www` und erstellen Sie ein neues Unterverzeichnis namens `inc`.

```
cd /var/www
mkdir inc
cd inc
```

2. Erstellen Sie im Verzeichnis `inc` eine neue Datei namens `dbinfo.inc` und bearbeiten Sie anschließend die Datei, indem Sie Nano (oder einen Editor Ihrer Wahl) aufrufen.

```
>dbinfo.inc
nano dbinfo.inc
```

3. Fügen Sie der Datei `dbinfo.inc` die folgenden Inhalte hinzu. In diesem Fall ist *`db_instance_endpoint`* Ihr DB-Cluster-Writer-Endpunkt ohne den Port für Ihre Ihren DB-Cluster.

> **Note**
>
> Es wird empfohlen, die Informationen zu Benutzername und Passwort in einem Ordner abzulegen, der nicht Teil des Dokumentstammverzeichnisses für Ihren Webserver ist. Auf diese Weise wird die Möglichkeit verringert, dass Ihre Sicherheitsinformationen offengelegt werden.
> Stellen Sie sicher, dass Sie `master password` in Ihrer Anwendung in ein geeignetes Passwort ändern.

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. Speichern und schließen Sie die Datei `dbinfo.inc`. Wenn Sie Nano verwenden, speichern und schließen Sie die Datei mit Strg+S und Strg+X.

5. Ändern Sie das Verzeichnis in `/var/www/html`.

```
cd /var/www/html
```

6. Erstellen Sie im Verzeichnis `html` eine neue Datei namens `SamplePage.php` und bearbeiten Sie anschließend die Datei, indem Sie Nano (oder einen Editor Ihrer Wahl) aufrufen.

```
>SamplePage.php
nano SamplePage.php
```

7. Fügen Sie der Datei `SamplePage.php` die folgenden Inhalte hinzu:

   MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
<body>
<h1>Sample page</h1>
<?php

  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);

  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
```

```
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php
```

```
  mysqli_free_result($result);
  mysqli_close($connection);

?>

</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);

   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
```

```
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");

  if(mysqli_num_rows($checktable) > 0) return true;

  return false;
}
?>
```

PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
<h1>Sample page</h1>
<?php

/* Connect to PostgreSQL and select the database. */
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
```

```
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php
```

```
  pg_free_result($result);
  pg_close($connection);
?>
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
```

```
  return false;

}
?>
```

8. Speichern und schließen Sie die Datei `SamplePage.php`.
9. Überprüfen Sie, ob Ihr Webserver erfolgreich eine Verbindung mit Ihrem DB-Cluster herstellen kann, indem Sie einen Webbrowser öffnen und zu `http://`*`EC2 instance endpoint`*`/SamplePage.php` navigieren, beispielsweise: `http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php`.

Sie können `SamplePage.php` verwenden, um Ihrem DB-Cluster Daten hinzuzufügen. Die von Ihnen hinzugefügten Daten werden anschließend auf der Seite angezeigt. Wenn Sie überprüfen möchten, ob die Daten in die Tabelle eingefügt wurden, installieren Sie den MySQL-Client auf der Amazon-EC2-Instance. Stellen Sie dann eine Verbindung mit dem DB-Cluster her und führen Sie eine Abfrage der Tabelle aus.

Informationen zum Herstellen einer Verbindung mit einem DB-Cluster finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

Um sicherzustellen, dass Ihr DB-Cluster so sicher wie möglich ist, müssen Sie sich vergewissern, dass Quellen außerhalb der VPC keine Verbindungen mit Ihrem DB-Cluster herstellen können.

Nachdem Sie Ihren Webserver und Ihre Datenbank getestet haben, sollten Sie Ihren DB-Cluster und Ihre Amazon EC2-Instance löschen.

- Um einen DB-Cluster zu löschen, folgen Sie den Anweisungen in Löschen von Aurora-DB-Clustern und -DB-Instances. Sie müssen keinen abschließenden Snapshot erstellen.
- Um eine Amazon EC2-Instance zu beenden, folgen Sie den Anweisungen in Beenden Ihrer Instance im Amazon EC2-Benutzerhandbuch.

Amazon Aurora Tutorials und Beispiel-Code

Die AWS Dokumentation enthält mehrere Tutorials, die Sie durch gängige Aurora Aurora-Anwendungsfälle führen. Viele dieser Tutorials zeigen Ihnen, wie Sie Aurora mit anderen AWS Diensten verwenden können. Außerdem können Sie in auf den Beispiel-Code zugreifen.

> **Note**
>
> Weitere Tutorials finden Sie im AWS -Datenbank-Blog. Informationen zu Schulungen finden Sie unter AWS Training and Certification.

Themen

- Tutorials in diesem Handbuch
- Tutorials in anderen AWS Leitfäden
- Tutorials und Beispielcode in GitHub
- AWS Datenbank-Kochbuch
- AWS Portal für Workshop- und Laborinhalte für Amazon Aurora PostgreSQL
- AWS Portal für Workshop- und Laborinhalte für Amazon Aurora MySQL
- Verwenden Sie diesen Dienst mit einem AWS SDK

Tutorials in diesem Handbuch

Die folgenden Tutorials in diesem Handbuch veranschaulichen, wie Sie mit Amazon Aurora gängige Aufgaben durchführen.

- Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4)

  Erfahren Sie, wie Sie einen DB-Cluster in eine Virtual Private Cloud (VPC) aufnehmen, die auf dem Amazon-VPC-Service basiert. In diesem Fall teilt die VPC Daten mit einem Webserver, der auf einer EC2 Amazon-Instance in derselben VPC läuft.
- Tutorial: Erstellen einer VPC zur Verwendung mit einer DB–einem DB-Cluster (Dual-Stack-Modus)

  Erfahren Sie, wie Sie einen DB-Cluster in eine Virtual Private Cloud (VPC) aufnehmen, die auf dem Amazon-VPC-Service basiert. In diesem Fall teilt die VPC Daten mit einer EC2 Amazon-

Instance in derselben VPC. In diesem Tutorial erstellen Sie die VPC für dieses Szenario, die mit einer Datenbank arbeitet, die im Dual-Stack-Modus ausgeführt wird.

- Tutorial: Erstellen eines Webservers und einer eines Amazon Aurora-DB-Clusters

  Weitere Informationen darüber, wie Sie einen Apache-Webserver mit PHP installieren und eine MySQL-Datenbank zu erstellen. Der Webserver läuft auf einer EC2 Amazon-Instance, die Amazon Linux verwendet, und die MySQL-Datenbank ist , ein Aurora MySQL-DB-Cluster. Sowohl die EC2 Amazon-Instance als auch der werden in einer Amazon VPC ausgeführt.

- Tutorial: Wiederherstellen eines DB-Clusters von Amazon Aurora aus einem DB-Cluster-Snapshot

  Erfahren Sie, wie Sie einen DB-Cluster aus einem DB-Cluster-Snapshot wiederherstellen.

- Tutorial: Verwenden Sie Tags, um festzulegen, welcher Aurora-DB-Cluster stoppt.

  Erfahren Sie, wie Sie Tags verwenden, um festzulegen, welcher Aurora-DB-Cluster stoppt.

- Tutorial: Protokollieren von Statusänderungen an einer DB-Instance mit Amazon EventBridge

  Erfahren Sie, wie Sie eine Änderung des DB-Instance-Status mithilfe von Amazon EventBridge und protokollieren AWS Lambda.

Tutorials in anderen AWS Leitfäden

Die folgenden Tutorials in anderen AWS Handbüchern zeigen Ihnen, wie Sie allgemeine Aufgaben mit Amazon Aurora ausführen:

> **Note**
>
> Einige der Tutorials verwenden Amazon-RDS-DB-Instances, aber sie können angepasst werden, so dass sie Aurora-DB-Cluster verwenden.

- Tutorial: Aurora Serverless in AWS AppSync -Entwicklerhandbuch

  Erfahren Sie, wie Sie AWS AppSync mit aktivierter Daten-API eine Datenquelle für die Ausführung von SQL-Befehlen für Aurora Serverless DB-Cluster bereitstellen können. Sie können AWS AppSync -Resolver nutzen, um SQL-Anweisungen für die Daten-API mit GraphQL-Abfragen, Mutationen und Abonnements auszuführen.

- Tutorial: Ein Geheimnis für eine AWS Datenbank rotieren im AWS Secrets Manager Benutzerhandbuch

  Erfahren Sie, wie Sie ein Geheimnis für eine AWS Datenbank erstellen und das Geheimnis so konfigurieren, dass es nach einem Zeitplan rotiert. Sie lösen eine Rotation manuell aus und bestätigen dann, dass die neue Version des Secrets weiterhin Zugriff bietet.
- Tutorials und Beispiele im AWS Elastic Beanstalk -Entwicklerhandbuch

  Erfahren Sie, wie Sie Anwendungen bereitstellen, die Amazon RDS-Datenbanken verwenden AWS Elastic Beanstalk.
- Verwenden von Daten aus einer Amazon-RDS-Datenbank zum Erstellen einer Amazon ML-Datenquelle im Amazon Machine Learning Developer Guide

  Erfahren Sie, wie Sie ein Amazon Machine Learning (Amazon ML)-Datenquellenobjekt aus Daten erstellen, die in einer MySQL-DB-Instance gespeichert sind.
- Manuelles Aktivieren des Zugriffs auf eine Amazon RDS-Instance in einer VPC im Amazon Quick Suite-Benutzerhandbuch

  Erfahren Sie, wie Sie den Quick Suite-Zugriff auf eine Amazon RDS-DB-Instance in einer VPC aktivieren.

Tutorials und Beispielcode in GitHub

Die folgenden Tutorials und der Beispielcode GitHub zeigen Ihnen, wie Sie allgemeine Aufgaben mit Amazon Aurora ausführen:

- Erstellen einer Leihbibliothek von Aurora Serverless v2

  Erfahren Sie, wie Sie eine Leihbibliothekanwendung erstellen, in der Kunden Bücher ausleihen und zurückgeben können. Das Beispiel verwendet Aurora Serverless v2 und AWS SDK for Python (Boto3).
- Erstellen einer Item-Tracker-Anwendung von Amazon Aurora mit einer Spring-REST-API, die Aurora Serverless v2 mithilfe von SDK for Java 2.x abfragt

  Erfahren Sie, wie Sie eine Spring-REST-API erstellen, die Daten von Aurora Serverless v2 abfragt. Sie wird von einer React-Anwendung unter Verwendung von SDK für Java 2.x genutzt.

- Erstellen einer Amazon Aurora Aurora-Artikel-Tracker-Anwendung, die Aurora Serverless v2 Daten abfragt mit AWS SDK für PHP

  Erfahren Sie, wie Sie eine Anwendung erstellen, die den `RdsDataClient` der Daten-API und Aurora Serverless v2 verwendet, um Arbeitselemente zu verfolgen und Berichte darüber zu erstellen. Das Beispiel verwendet AWS SDK für PHP.

- Erstellen einer Item-Tracker-Anwendung von Amazon Aurora, die Aurora Serverless v2 mithilfe von AWS SDK for Python (Boto3) abfragt

  Erfahren Sie, wie Sie eine Anwendung erstellen, die den `RdsDataClient` der Daten-API und Aurora Serverless v2 verwendet, um Arbeitselemente zu verfolgen und Berichte darüber zu erstellen. Das Beispiel verwendet AWS SDK for Python (Boto3).

AWS Datenbank-Kochbuch

Das AWS DB Cookbook ist ein umfassender Datenbank-Leitfaden, in dem Sie lernen, wie Sie leistungsstarke, kostengünstige Datenbanklösungen erstellen, bereitstellen und verwalten. AWS Step-by-stepTutorials führen Sie durch die Erstellung produktionsreifer Anwendungen und die Bereitstellung der Apps mit Vorlagen. CloudFormation Sie lernen wichtige AWS Dienste kennen, wenn Sie eine Infrastruktur aufbauen, Netzwerke implementieren, serverlose Architekturen entwickeln, Datenbanken verwalten und generative KI integrieren. Lernen Sie AWS bewährte Methoden kennen, die Ihnen helfen, sichere, skalierbare Lösungen zu entwickeln und gleichzeitig die Kosten zu optimieren. Ganz gleich, ob Sie neu in der Branche sind AWS oder ein erfahrener Experte sind, das AWS DB Cookbook hilft Ihnen dabei, Fähigkeiten zu entwickeln, um häufig auftretende Datenbankprobleme zu lösen und unternehmenstaugliche Lösungen zu implementieren. Das Kochbuch enthält die folgenden Bereiche:

- Erste Schritte mit AWS für DB-Anwendungen — Lernen Sie die AWS Grundlagen kennen, z. B. wie Sie Ihr Konto und die Jupyter Notebook-Umgebung einrichten.
- Datenbankgrundlagen — Lernen Sie grundlegende Datenbankkonzepte kennen und vergleichen Sie AWS Datenbankdienste, um die richtige Lösung für Ihre Workloads auszuwählen.
- Serverlose Web-App mit Amazon Aurora — Erstellen Sie eine end-to-end Einzelhandelsanwendung mit Amazon Aurora PostgreSQL, die Inventar, Bestellungen und Kundendaten verwaltet.

- Überwachung und Beobachtbarkeit: Richten Sie Leistungsverfolgung ein und konfigurieren Sie Warnmeldungen, um potenzielle Datenbankprobleme zu erkennen, bevor sie sich auf Ihre Anwendungen auswirken.
- Skalieren mit Amazon Aurora: Erfahren Sie, wie Sie mit Aurora DSQL robuste Bereitstellungen in mehreren Regionen aufbauen und Ihre Datenbanken skalieren – für mehr Rechenleistung oder auf mehrere Instances für mehr Kapazität.
- Optimierungsleistung und Kosten: Optimieren Sie Ihre Datenbankleistung und senken Sie Kosten mit bewährten Optimierungsstrategien.
- Der Weg zu AWS speziell entwickelten Datenbanken — Bauen Sie eine sichere, zuverlässige Infrastruktur auf, die Ihre generativen KI-Lösungen und datengesteuerten Anwendungen vom Prototyp bis zur Bereitstellung im Unternehmen skaliert.
- GenAI-Anwendungen mit RAG: Bauen Sie ein intelligentes Suchsystem für Versicherungs- und Gesundheitsdokumente auf, das Retrieval Augmented Generation (RAG) verwendet, um genaue, kontextsensitive Ergebnisse zu liefern.

AWS Portal für Workshop- und Laborinhalte für Amazon Aurora PostgreSQL

Die folgende Sammlung von Workshops und anderen praktischen Inhalten hilft Ihnen, die Funktionen und Möglichkeiten von Amazon Aurora PostgreSQL zu verstehen:

- Manuelles Erstellen eines neuen Aurora-Clusters

  Erfahren Sie mehr dazu, wie Sie einen Cluster von Amazon Aurora PostgreSQL manuell erstellen.
- Konfigurieren von Cloud9 und Initialisieren der Datenbank

  Erfahren Sie, wie Sie Cloud9 konfigurieren und die PostgreSQL-Datenbank initialisieren.
- Schnelles Klonen

  Erfahren Sie mehr dazu, wie Sie einen schnellen Aurora-Klon erstellen.
- Abfrageplanverwaltung

  Erfahren Sie mehr dazu, wie Sie die Ausführungspläne für einen Satz von Anweisungen mithilfe der Abfrageplanverwaltung kontrollieren.
- Cluster-Cache-Verwaltung

Erfahren Sie mehr über die Cluster-Cache-Verwaltungsfunktion in Aurora PostgreSQL.

- Datenbankaktivitäts-Streaming

  Erfahren Sie mehr dazu, wie Sie mit dieser Funktion Ihre Datenbankaktivitäten überwachen und überprüfen.

- Verwenden von Performance-Insights

  Erfahren Sie, wie Sie Ihre DB-Instance mithilfe von Performance Insights überwachen und optimieren können.

- Leistungsüberwachung mit RDS-Tools

  Erfahren Sie, wie Sie Postgres-Tools (Cloudwatch, Enhanced Monitoring, Slow Query Logs, Performance Insights, PostgreSQL Catalog Views) verwenden AWS können, um Leistungsprobleme zu verstehen und Möglichkeiten zur Verbesserung der Leistung Ihrer Datenbank zu finden.

- Auto Scaling von Lesereplikaten

  Erfahren Sie, wie Auto Scaling von Aurora-Lesereplikaten mithilfe eines Lastgeneratorskripts in der Praxis funktioniert.

- Testen der Fehlertoleranz

  Erfahren Sie mehr über die Fehlertoleranz eines DB-Clusters.

- Globale Aurora-Datenbank

  Erfahren Sie mehr über globale Aurora-Datenbanken.

- Verwendung von Machine Learning

  Erfahren Sie mehr über Aurora Machine Learning.

- Aurora Serverless v2

  Erfahren Sie mehr über Aurora Serverless v2.

- Trusted Language Extensions für PostgreSQL

  Erfahren Sie, wie Sie Hochleistungserweiterungen erstellen, die sicher auf Aurora PostgreSQL laufen.

AWS Portal für Workshop- und Laborinhalte für Amazon Aurora MySQL

Die folgende Sammlung von Workshops und anderen praktischen Inhalten hilft Ihnen, die Funktionen und Möglichkeiten von Amazon Aurora MySQL zu verstehen:

- Erstellen eines Aurora-Clusters

  Erfahren Sie mehr dazu, wie Sie einen Cluster von Amazon Aurora MySQL manuell erstellen.
- Erstellen einer cloudbasierten Cloud9-IDE-Umgebung zur Verbindung mit Ihrer Datenbank

  Erfahren Sie, wie Sie Cloud9 konfigurieren und die MySQL-Datenbank initialisieren.
- Schnelles Klonen

  Erfahren Sie mehr dazu, wie Sie einen schnellen Aurora-Klon erstellen.
- Rückverfolgen eines Clusters

  Erfahren Sie mehr dazu, wie Sie einen DB-Cluster rückverfolgen.
- Verwenden von Performance-Insights

  Erfahren Sie, wie Sie Ihre DB-Instance mithilfe von Performance Insights überwachen und optimieren können.
- Leistungsüberwachung mit RDS-Tools

  Erfahren Sie, wie Sie SQL-Tools verwenden AWS , um Leistungsprobleme zu verstehen und Möglichkeiten zur Verbesserung der Leistung Ihrer Datenbank zu finden.
- Analysieren der Abfrageleistung

  Erfahren Sie, wie Sie Probleme mit der SQL-Leistung mithilfe verschiedener Tools beheben können.
- Auto Scaling von Lesereplikaten

  Erfahren Sie, wie Auto Scaling von Lesereplikaten funktioniert.
- Testen der Fehlertoleranz

  Erfahren Sie mehr über Hochverfügbarkeits- und Fehlertoleranzfunktionen in Aurora MySQL.
- Globale Aurora-Datenbank

Erfahren Sie mehr über globale Aurora-Datenbanken.

- Aurora Serverless v2

  Erfahren Sie mehr über Aurora Serverless v2.

- Verwendung von Machine Learning

  Erfahren Sie mehr über Aurora Machine Learning.

Verwenden Sie diesen Dienst mit einem AWS SDK

AWS Software Development Kits (SDKs) sind für viele beliebte Programmiersprachen verfügbar. Jedes SDK bietet eine API, Codebeispiele und Dokumentation, die es Entwicklern erleichtern, Anwendungen in ihrer bevorzugten Sprache zu erstellen.

SDK-Dokumentation	Codebeispiele
AWS SDK für C++	AWS SDK für C++ Codebeispiele
AWS CLI	AWS CLI Code-Beispiele
AWS SDK für Go	AWS SDK für Go Code-Beispiele
AWS SDK für Java	AWS SDK für Java Code-Beispiele
AWS SDK für JavaScript	AWS SDK für JavaScript Code-Beispiele
AWS SDK für Kotlin	AWS SDK für Kotlin Code-Beispiele
AWS SDK für .NET	AWS SDK für .NET Code-Beispiele
AWS SDK für PHP	AWS SDK für PHP Code-Beispiele
AWS -Tools für PowerShell	AWS -Tools für PowerShell Code-Beispiele
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) Code-Beispiele
AWS SDK für Ruby	AWS SDK für Ruby Code-Beispiele
AWS SDK für Rust	AWS SDK für Rust Code-Beispiele

SDK-Dokumentation	Codebeispiele
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP Code-Beispiele
AWS SDK für Swift	AWS SDK für Swift Code-Beispiele

Weitere Beispiele speziell für diesen Service finden Sie unter Codebeispiele für Aurora mit AWS SDKs.

> **Beispiel für die Verfügbarkeit**
>
> Sie können nicht finden, was Sie brauchen? Fordern Sie ein Codebeispiel an, indem Sie unten den Link Provide feedback (Feedback geben) auswählen.

Programmatischer Zugriff auf Amazon Aurora

Amazon RDS bietet Ihnen die folgenden Tools, um Ihre Amazon Aurora-Ressourcen programmgesteuert zu verwalten.

AWS Command Line Interface (AWS CLI)

Sie können Ihre Amazon Aurora-Ressourcen über die AWS CLI in einer Befehlszeilen-Shell erstellen und verwalten. Die AWS CLI bietet direkten Zugriff auf die APIs für AWS-Services wie Amazon RDS. Weitere Informationen zur Syntax und zu Beispielen für die Befehle für Amazon RDS finden Sie unter rds in der AWS CLI-Befehlsreferenz.

AWS CloudFormation

Mit diesem AWS Infrastructure as Code (IaC)-Tool von AWS können Sie Vorlagen erstellen, die alle gewünschten Amazon-Aurora-Ressourcen beschreiben, und AWS CloudFormation stellt diese Ressourcen für Sie bereit und konfiguriert sie für Sie. Weitere Informationen finden Sie unter the section called “Ressourcen erstellen mit AWS CloudFormation”.

AWS-Software Development Kits (SKDs)

AWS bietet SDKs für viele beliebte Technologien und Programmiersprachen. Sie erleichtern es Ihnen, AWS direkt über Ihre Services, und zwar in der jeweiligen Sprache oder Technologie aufzurufen. Weitere Informationen zu diesen SDKs finden Sie unter Tools für die Entwicklung und Verwaltung von Anwendungen auf AWS.

Amazon-RDS-API

Diese API ist die Schnittstelle auf Protokollebene für Amazon RDS. Wenn Sie diese API verwenden, müssen Sie jede HTTPS-Anfrage korrekt formatieren und jeder Anfrage eine gültige digitale Signatur hinzufügen. Weitere Informationen finden Sie unter Amazon RDS-API-Referenz.

Console-to-Code

Mit diesem Tool können Sie Code für Aktionen generieren, die Sie in der Amazon-RDS-Konsole ausführen, und diesen Code in anderen Tools wie AWS CloudFormation verwenden. Weitere Informationen finden Sie unter the section called “Console-to-Code”.

RDS-Daten-API

Die RDS-Daten-API (Daten-API) bietet programmatischen Zugriff auf die Daten in Ihren Amazon-Aurora-Serverless-Datenbanken. Mit der Daten-API können Sie einen sicheren HTTP-Endpunkt

verwenden, um Ihre SQL-Anweisungen auszuführen, ohne Verbindungen zu verwalten. Weitere Informationen finden Sie unter Verwenden der Daten-API von Amazon RDS.

Verwenden Sie Console-to-Code, um Code für Ihre Aktionen der Amazon Aurora-Konsole zu generieren.

Die Konsole bietet eine Anleitung zum Erstellen von Ressourcen und zum Testen von Prototypen. Wenn Sie dieselben Ressourcen in großem Umfang erstellen möchten, benötigen Sie Automatisierungscode. Console-to-Code ist ein Feature von Amazon Q Developer, die Ihnen den Einstieg in Automatisierungscode erleichtern kann. Console-to-Code zeichnet Ihre Konsolenaktionen auf, einschließlich der Standard- und Parameterwerte, die Sie bereitstellen. Anschließend wird mithilfe generativer KI Code in Ihrer bevorzugten Sprache und Ihrem bevorzugten Format für die ausgewählten Aktionen vorgeschlagen. Da der Konsolen-Workflow sicherstellt, dass die von Ihnen angegebenen Parameterwerte zusammen gültig sind, hat der mithilfe von Console-to-Code generierte Code kompatible Parameterwerte. Sie können den Code als Ausgangspunkt verwenden und ihn dann so anpassen, dass er für Ihren speziellen Anwendungsfall produktionsbereit ist.

Mit Console-to-Code können Sie beispielsweise die Erstellung einer eines Aurora-DB-Clusters aufzeichnen und festlegen, dass Code im JSON-Format AWS CloudFormation generiert wird. Anschließend können Sie diesen Code kopieren und für die Verwendung in Ihrer AWS CloudFormation-Vorlage anpassen.

Console-to-Code kann derzeit Infrastructure as Code (IaC) in den folgenden Sprachen und Formaten generieren:

- CDK Java
- CDK Python
- CDK TypeScript
- CloudFormation JSON
- CloudFormation YAML

Weitere Informationen und Anweisungen zur Verwendung von Console-to-Code finden Sie unter Automatisieren von AWS-Services mit Console-to-Code von Amazon Q Developer im Amazon-Q-Developer-Benutzerhandbuch.

Einrichten eines DB-Clusters von Amazon Aurora

In diesem Abschnitt wird erläutert, wie Sie Ihren Aurora-DB-Cluster einrichten. Legen Sie vor dem Erstellen eines Aurora-DB-Clusters fest, welche DB-Instance-Klasse den DB-Cluster ausführen soll. Legen Sie auch fest, wo der DB-Cluster ausgeführt werden soll. Wählen Sie dazu eine AWS-Region aus. Erstellen Sie nun den DB-Cluster. Wenn Sie Daten außerhalb von Aurora haben, können Sie diese in einen Aurora-DB-Cluster migrieren.

Themen

- Erstellen eines Amazon Aurora-DB Clusters
- Amazon-Aurora-Ressourcen erstellen mit AWS CloudFormation
- Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster
- Parametergruppen für Amazon Aurora
- Migrieren von Daten zu einem Amazon Aurora-DB-Cluster
- Erstellen eines Amazon-ElastiCache-Caches mithilfe der Einstellungen des Aurora-DB-Clusters
- Automatische Migration von EC2-Datenbanken zu Amazon Aurora mithilfe von AWS Database Migration Service
- Tutorial: Erstellen einer eines DB-Clusters für MySQL mit einem benutzerdefinierten Parameter

Erstellen eines Amazon Aurora-DB Clusters

Ein Amazon-Aurora-DB-Cluster besteht aus einer DB-Instance, die entweder mit MySQL oder PostgreSQL kompatibel ist, und einem Cluster-Volume, das die Daten für den DB-Cluster enthält und in drei Availability Zones als ein einziges virtuelles Volume kopiert wird. Standardmäßig enthält ein Aurora-DB-Cluster eine primäre DB-Instance, die Lese- und Schreibvorgänge ausführt, und optional bis zu 15 Aurora-Replikate (Reader-DB-Instances). Weitere Informationen zu Aurora-DB-Clustern finden Sie unter Amazon-Aurora-DB-Cluster.

Aurora bietet zwei Haupttypen von DB-Clustern:

- Von Aurora bereitgestellt – Sie wählen die DB-Instance-Klasse für die Writer- und Reader-Instances basierend auf Ihrer erwarteten Workload aus. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora. Der Typ „von Aurora bereitgestellt“ verfügt über mehrere Optionen, einschließlich globalen Aurora-Datenbanken. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.
- Aurora Serverless – Aurora Serverless v2 ist eine bedarfsabhängige Auto-Scaling-Konfiguration für Aurora. Die Kapazität wird automatisch basierend auf dem Anwendungsbedarf angepasst. Berechnet werden Ihnen nur die Ressourcen, die Ihr DB-Cluster verbraucht. Diese Automatisierung ist besonders nützlich für Umgebungen mit sehr variablen und unvorhersehbaren Workloads. Weitere Informationen finden Sie unter Verwenden von Aurora Serverless v2.

Im Folgenden erfahren Sie, wie Sie einen Aurora-DB-Cluster erstellen. Lesen Sie zum Einstieg zuerst Voraussetzungen für DB-Cluster.

Anweisungen zum Herstellen von Verbindungen mit Ihrem Aurora-DB-Cluster finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

Inhalt

- Voraussetzungen für DB-Cluster
  - Konfigurieren des Netzwerks für den DB-Cluster
    - Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren
    - Manuelles Konfigurieren des Netzwerks
  - Zusätzliche Voraussetzungen
- Erstellen eines DB-Clusters
  - Erstellen einer primären (Writer-)DB-Instance

- Einstellungen für Aurora-DB-Cluster
- Einstellungen, die nicht für Amazon-Aurora-DB-Cluster gelten
- Einstellungen, die nicht für Amazon-Aurora-DB-Instances gelten

Voraussetzungen für DB-Cluster

> **Important**
>
> Sie müssen die Schritte im Abschnitt Einrichten Ihrer Umgebung für Amazon Aurora abschließen, bevor Sie einen Aurora-DB-Cluster erstellen können.

Im Folgenden werden die Voraussetzungen vor dem Erstellen eines DB-Clusters aufgeführt.

Themen

- Konfigurieren des Netzwerks für den DB-Cluster
- Zusätzliche Voraussetzungen

Konfigurieren des Netzwerks für den DB-Cluster

Sie können einen Amazon Aurora-DB-Cluster nur in einer Virtual Private Cloud (VPC) in einer AWS-Region erstellen, die auf dem Amazon VPC-Service basiert und mindestens zwei Availability Zones besitzt. Die DB-Subnetzgruppe, die Sie für den DB-Cluster wählen, muss mindestens zwei Availability Zones abdecken. Diese Konfiguration stellt sicher, dass Ihr DB-Cluster stets über mindestens eine DB-Instance verfügt, um in dem unwahrscheinlichen Fall, dass eine Availability Zone ausfällt, einen Failover ausführen zu können.

Wenn Sie die Konnektivität zwischen Ihrem neuen DB-Cluster und einer EC2-Instance in derselben VPC einrichten möchten, können Sie dies während der Erstellung des DB-Clusters tun. Wenn Sie von anderen Ressourcen als EC2-Instances in derselben VPC aus eine Verbindung zu Ihrem DB-Cluster herstellen möchten, können Sie die Netzwerkverbindungen manuell konfigurieren.

Themen

- Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren
- Manuelles Konfigurieren des Netzwerks

Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren

Wenn Sie einen Aurora-DB-Cluster erstellen, können Sie die AWS-Managementkonsole verwenden, um die Verbindung zwischen einer Amazon-EC2-Instance und dem neuen DB-Cluster einzurichten. In diesem Fall konfiguriert RDS Ihre VPC- und Netzwerkeinstellungen automatisch. Der DB-Cluster wird in derselben VPC wie die EC2-Instance erstellt, sodass die EC2-Instance auf den DB-Cluster zugreifen kann.

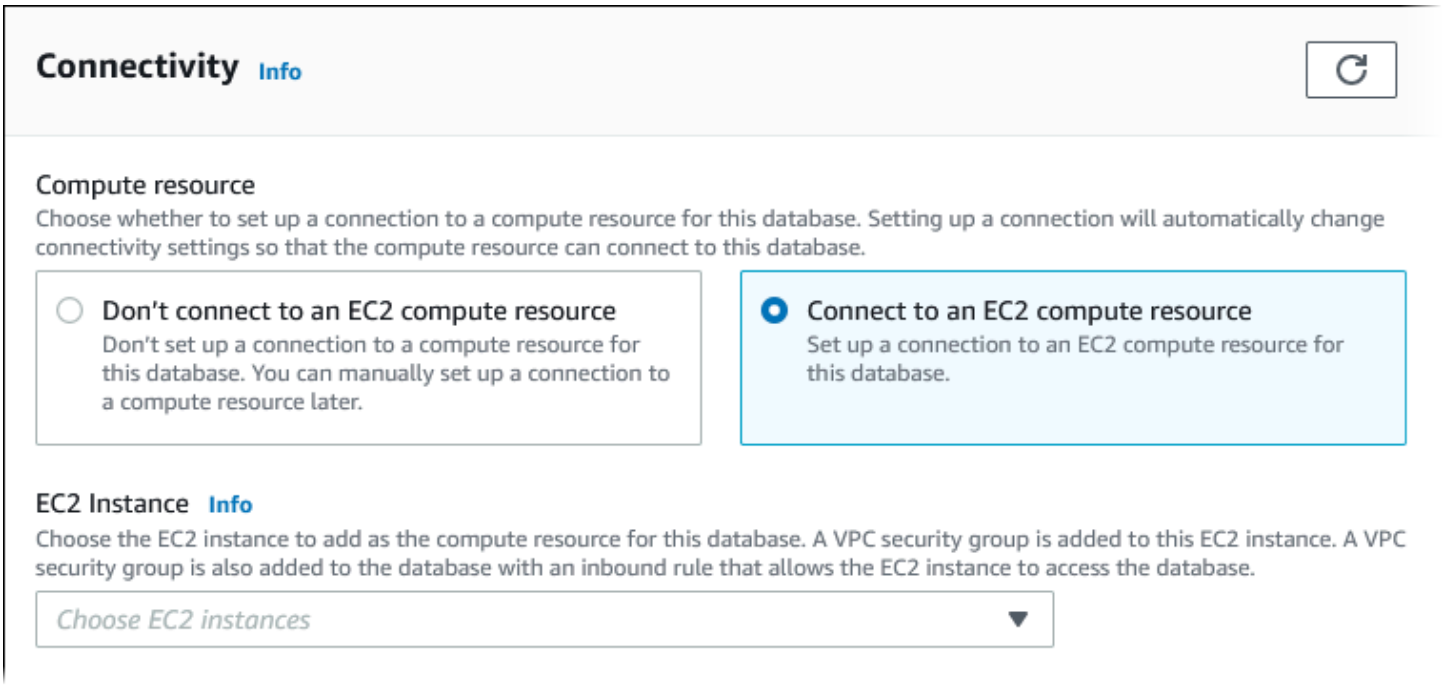
Im Folgenden sind die Anforderungen für die Verbindung einer EC2-Instance mit dem DB-Cluster aufgeführt:

- Die EC2-Instance muss in der AWS-Region vorhanden sein, bevor Sie den DB-Cluster erstellen.

  Wenn keine EC2-Instances in der AWS-Region vorhanden sind, bietet die Konsole einen Link zum Erstellen einer solchen.
- Derzeit kann der DB-Cluster kein DB-Cluster von Aurora Serverless oder Teil einer globalen Aurora-Datenbank sein.
- Der Benutzer, der die DB-Instance erstellt, muss über Berechtigungen zum Ausführen der folgenden Vorgänge verfügen:
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Mit dieser Option wird ein privater DB-Cluster erstellt. Der DB-Cluster verwendet eine DB-Subnetzgruppe mit ausschließlich privaten Subnetzen, um den Zugriff auf Ressourcen innerhalb der VPC einzuschränken.

Um eine EC2-Instance mit dem DB-Cluster zu verbinden, wählen Sie Connect to an EC2 compute resource (Verbindung mit einer EC2-Compute-Ressource herstellen) im Abschnitt Connectivity (Konnektivität) auf der Seite Create database (Datenbank erstellen) aus.



Wenn Sie Verbindung zu einer EC2-Rechenressource herstellen wählen, legt RDS die folgenden Optionen automatisch fest. Sie können diese Einstellungen nur ändern, wenn Sie sich dafür entscheiden, keine Konnektivität mit einer EC2-Instance einzurichten, indem Sie Keine Verbindung zu einer EC2-Rechenressource herstellen wählen.

Konsolenoption	Automatische Einstellung
Netzwerktyp	RDS legt den Netzwerktyp auf IPv4 fest. Derzeit wird der Dual-Stack-Modus nicht unterstützt, wenn Sie eine Verbindung zwischen einer EC2-Instance und dem DB-Cluster einrichten.
Virtual Private Cloud (VPC)	RDS legt die VPC auf die VPC fest, die der EC2-Instance zugeordnet ist.

Konsolenoption	Automatische Einstellung
DB-Subnetzgruppe	RDS erfordert eine DB-Subnetzgruppe mit einem privaten Subnetz in derselben Availability Zone wie die EC2-Insta nce. Wenn eine DB-Subnetzgruppe vorhanden ist, die diese Anforderung erfüllt, dann verwendet RDS die vorhandene DB-Subnetzgruppe. Standardmäßig ist diese Option auf Automatic setup (Automatische Einrichtung) eingestellt. Wenn Sie Automatic setup (Automatische Einrichtung) auswählen und es keine DB-Subnetzgruppe gibt, die diese Anforderung erfüllt, wird die folgende Aktion ausgeführt. RDS verwendet drei verfügbare private Subnetze in drei Availability Zones, wobei eine der Availability Zones mit der AZ der EC2-Instance identisch ist. Wenn kein privates Subnetz in einer Availability Zone verfügbar ist, erstellt RDS ein privates Subnetz in der Availability Zone. Anschließend erstellt RDS die DB-Subnetzgruppe. Wenn ein privates Subnetz verfügbar ist, verwendet RDS die zugehörige Routing-Tabelle und fügt alle Subnetze, die es erstellt, dieser Routing-Tabelle hinzu. Wenn kein privates Subnetz verfügbar ist, erstellt RDS eine Routing-Tabelle ohne Internet-Gateway-Zugriff und fügt die erstellten Subnetze der Routing-Tabelle hinzu. Mit RDS können Sie auch vorhandene DB-Subnetzgruppen verwenden. Wählen Sie Choose existing (Vorhandene wählen) aus, wenn Sie eine vorhandene DB-Subnetzgruppe Ihrer Wahl verwenden möchten.
Öffentlicher Zugriff	RDS wählt Nein, so dass der DB-Cluster nicht öffentlich zugänglich ist. Aus Sicherheitsgründen ist es eine bewährte Methode, die Datenbank privat zu halten und sicherzustellen, dass sie nicht über das Internet zugänglich ist.

Konsolenoption	Automatische Einstellung
VPC-Sicherheitsgruppe (Firewall)	RDS erstellt eine neue Sicherheitsgruppe, die mit dem DB-Cluster verknüpft ist. Die Sicherheitsgruppe heißt `rds-ec2-`*n*, wobei *n* eine Zahl ist. Diese Sicherheitsgruppe enthält eine Regel für eingehenden Datenverkehr mit der EC2 VPC-Sicherheitsgruppe (Firewall) als Quelle. Diese Sicherheitsgruppe, die dem DB-Cluster zugeordnet ist, ermöglicht der EC2-Instance den Zugriff auf den DB-Cluster. RDS erstellt außerdem eine neue Sicherheitsgruppe, die der EC2-Instance zugeordnet ist. Die Sicherheitsgruppe heißt `ec2-rds-`*n*, wobei *n* eine Zahl ist. Diese Sicherheitsgruppe enthält eine ausgehende Regel mit der VPC-Sicherheitsgruppe des DB-Clusters als Quelle. Diese Sicherheitsgruppe ermöglicht es der EC2-Instance, Datenverkehr an den DB-Cluster zu senden. Sie können eine weitere neue Sicherheitsgruppe hinzufügen, indem Sie Neu erstellen wählen und den Namen der neuen Sicherheitsgruppe eingeben. Sie können vorhandene Sicherheitsgruppen hinzufügen, indem Sie Bestehende auswählen und Sicherheitsgruppen auswählen, die hinzugefügt werden sollen.

Konsolenoption	Automatische Einstellung
Availability Zone	Wenn Sie keine Aurora-Replica in Availability & durability (Verfügbarkeit und Haltbarkeit) bei der Erstellung eines DB-Clusters (Single-AZ-Bereitstellung) erstellen, wählt RDS die Availability Zone der EC2-Instance aus. Wenn Sie während der Erstellung eines DB-Clusters ein Aurora Replica erstellen (Multi-AZ-Bereitstellung), wählt RDS die Availability Zone der EC2-Instance für eine DB-Instance im DB-Cluster aus. RDS wählt zufällig eine andere Availability Zone für die andere DB-Instance im DB-Cluster aus. Entweder die primäre DB-Instance oder Aurora-Replica wird in derselben Availability Zone wie die EC2-Instance erstellt. Es besteht die Möglichkeit von Kosten über Availability Zones hinweg, wenn sich die primäre DB-Instance und die EC2-Instance in unterschi edlichen Availability Zones befinden.

Weitere Informationen zu diesen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.

Wenn Sie nach dem Erstellen des DB-Clusters Änderungen an diesen Einstellungen vornehmen, können sich die Änderungen auf die Verbindung zwischen der EC2-Instance und dem DB-Cluster auswirken.

Manuelles Konfigurieren des Netzwerks

Wenn Sie von anderen Ressourcen als EC2-Instances in derselben VPC aus eine Verbindung zu Ihrem DB-Cluster herstellen möchten, können Sie die Netzwerkverbindungen manuell konfigurieren. Wenn Sie die AWS-Managementkonsole zum Erstellen Ihres DB-Clusters verwenden, kann Amazon RDS automatisch eine VPC für Sie erstellen. Sie können auch eine bestehende VPC verwenden oder eine neue VPC für Ihren Aurora-DB-Cluster erstellen. Unabhängig davon, welchen Ansatz Sie wählen, muss Ihre VPC mindestens ein Subnetz in jeder von mindestens zwei Availability Zones haben, damit Sie sie mit einem Amazon-Aurora-DB-Cluster verwenden können.

Standardmäßig erstellt Amazon RDS die primäre DB-Instance und das Aurora Replica in den Availability Zones automatisch für Sie. Um eine bestimmte Availability Zone auszuwählen, müssen Sie die Einstellung der Multi-AZ-Bereitstellung Availability & durability (Verfügbarkeit und Haltbarkeit) auf Kein Aurora-Replikat erstellen ändern. Dadurch wird eine Availability Zone-Einstellung,

mit der Sie zwischen den Availability Zones in Ihrer VPC auswählen können, angezeigt. Wir empfehlen jedoch dringend, die Standardeinstellung beizubehalten und Amazon RDS eine Multi-AZ-Bereitstellung erstellen zu lassen und Availability Zones für Sie auszuwählen. Auf diese Weise wird Ihr Aurora-DB-Cluster mit den schnellen Failover- und Hochverfügbarkeitsfunktionen erstellt, die zwei der Hauptvorteile von Aurora sind.

Wenn Sie keine Standard-VPC besitzen oder keine VPC erstellt haben, kann Amazon RDS automatisch eine VPC für Sie erstellen, wenn Sie ein DB-Cluster über die Konsole erstellen. Andernfalls müssen Sie wie folgt vorgehen:

- Erstellen Sie eine VPC mit mindestens einem Subnetz in jeder von mindestens zwei Availability Zones in der AWS-Region, in der Sie Ihr DB-Cluster bereitstellen möchten. Weitere Informationen erhalten Sie unter Arbeiten mit einer DB-Cluster in einer VPC und Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).
- Legen Sie eine VPC-Sicherheitsgruppe fest, die Verbindungen mit Ihrem DB-Cluster autorisiert. Weitere Informationen erhalten Sie unter Ermöglichen des Zugriffs auf den DB-Cluster in der VPC durch Erstellen einer Sicherheitsgruppe und Zugriffskontrolle mit Sicherheitsgruppen.
- Legen Sie eine RDS-DB-Subnetzgruppe fest, die mindestens zwei Subnetze in der VPC definiert, die vom DB-Cluster verwendet werden können. Weitere Informationen finden Sie unter Arbeiten mit DB-Subnetzgruppen.

Weitere Informationen zu VPCs finden Sie unter Amazon VPC und Amazon Aurora. Ein Tutorial, das das Netzwerk für einen privaten DB-Cluster konfiguriert, finden Sie unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).

Wenn Sie eine Verbindung mit einer Ressource herstellen möchten, die sich nicht in derselben VPC wie der Aurora-DB-Cluster befindet, sehen Sie sich die entsprechenden Szenarien unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC an.

Zusätzliche Voraussetzungen

Bevor Sie Ihren DB-Cluster erstellen, sollten Sie die folgenden zusätzlichen Voraussetzungen beachten:

- Wenn Sie sich mit AWS mithilfe von AWS Identity and Access Management (IAM)-Anmeldeinformationen anmelden, muss Ihr AWS-Konto über die IAM-Zugriffsrichtlinien mit den erforderlichen Berechtigungen für das Durchführen von Amazon-RDS-Aktionen verfügen. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora.

Wenn Sie IAM verwenden, um auf die Amazon-RDS-Konsole zuzugreifen, müssen Sie sich zuerst mit Ihren Anmeldeinformationen in der AWS-Managementkonsole anmelden. Öffnen Sie dann die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

- Wenn Sie die Konfigurationsparameter für Ihren DB-Cluster anpassen möchten, müssen Sie eine DB-Cluster-Parametergruppe und eine DB-Parametergruppe mit den erforderlichen Parametereinstellungen festlegen. Weitere Informationen über das Erstellen oder Ändern einer DB-Cluster-Parametergruppe oder DB-Parametergruppe finden Sie unter Parametergruppen für Amazon Aurora.
- Bestimmen Sie die TCP/IP-Portnummer, die Sie für Ihr DB-Cluster festlegen werden. Die Firewalls einiger Unternehmen blockieren Verbindungen zu den Standard-Ports (3306 für MySQL, 5432 für PostgreSQL) für Aurora. Wenn die Firewall Ihres Unternehmens den Standard-Port blockiert, wählen Sie einen anderen Port für Ihr DB-Cluster aus. Alle Instances in einem DB-Cluster verwenden denselben Port.
- Wenn die Haupt-Engine-Version für Ihre Datenbank das Ende des Standard-Supports für RDS erreicht hat, müssen Sie die CLI-Option für Extended Support oder den RDS-API-Parameter verwenden. Weitere Informationen finden Sie im Abschnitt zu RDS Extended Support unter Einstellungen für Aurora-DB-Cluster.

Erstellen eines DB-Clusters

Sie können einen Aurora-DB-Cluster über die AWS-Managementkonsole, die AWS CLI oder die RDS-API erstellen.

Konsole

Sie können einen DB-Cluster erstellen, indem Sie die AWS-Managementkonsole mit aktivierter oder deaktivierter Option Einfache Erstellung verwenden. Wenn die Option Einfache Erstellung aktiviert ist, geben Sie nur den Engine-Typ, die Größe der Instance und die Kennung der Instance für die DB an. Mit der Option Einfache Erstellung werden für die anderen Konfigurationsoptionen die Standardeinstellungen verwendet. Ist die Option Einfache Erstellung nicht aktiviert, geben Sie bei der Erstellung einer Datenbank weitere Konfigurationsoptionen an, über die Verfügbarkeit, Sicherheit, Backups und Wartung der Datenbank konfiguriert werden.

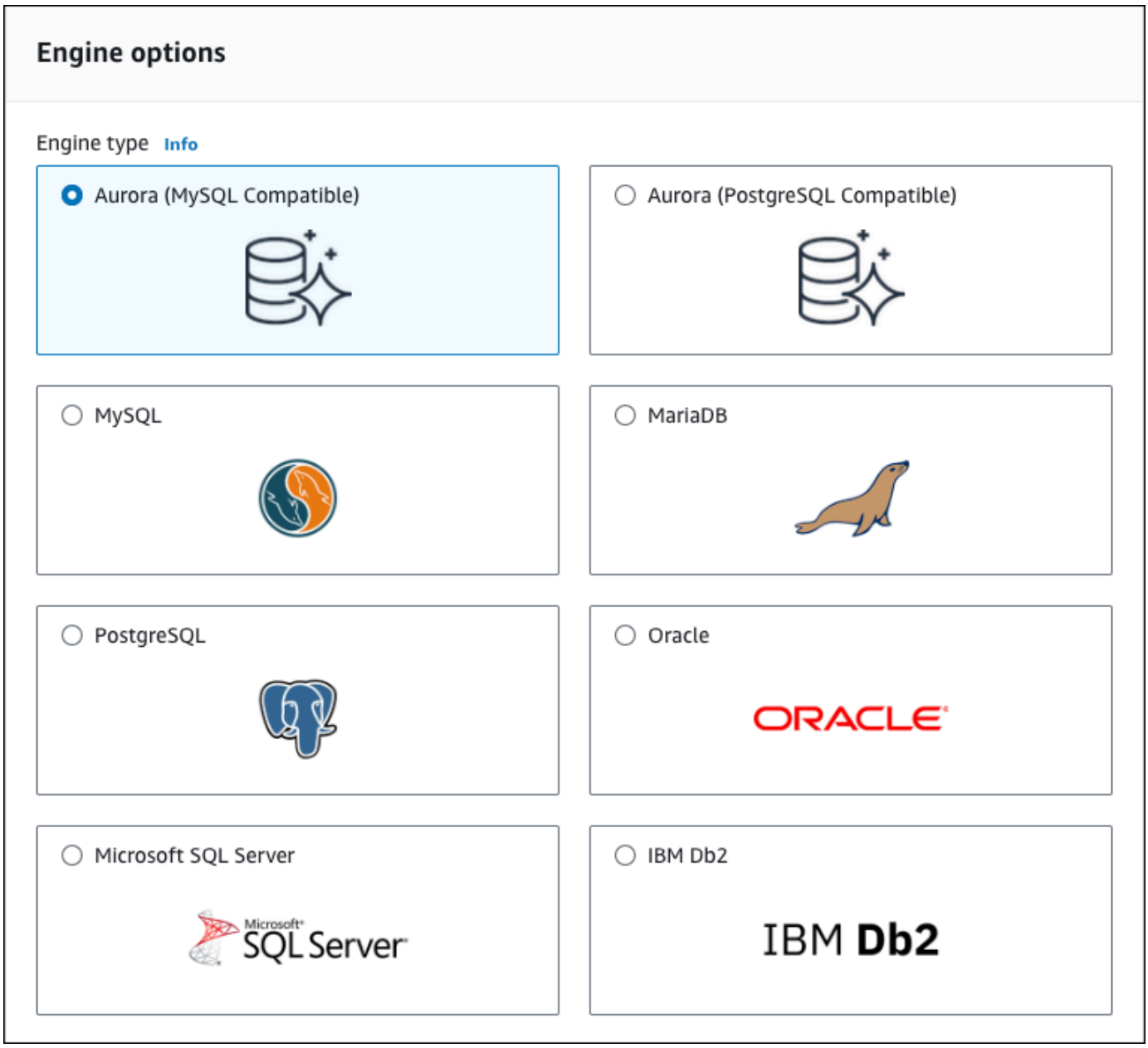
> **Note**
>
> In diesem Beispiel ist Standarderstellung aktiviert und Einfache Erstellung ist nicht aktiviert. Informationen zum Erstellen eines DB-Clusters unter Aktivierung der Option Einfache Erstellung finden Sie unter Erste Schritte mit Amazon Aurora.

Erstellen Sie einen Aurora-DB-Cluster mithilfe der Konsole wie folgt:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der oberen rechten Ecke von AWS-Managementkonsole die AWS-Region aus, in der Sie den DB-Cluster erstellen möchten.

   Aurora ist nicht in allen AWS-Regionen verfügbar. Die Liste der AWS-Regionen, in denen Aurora verfügbar ist, finden Sie unter Verfügbarkeit in Regionen.
3. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
4. Wählen Sie Datenbank erstellen aus.
5. Wählen Sie unter Auswahl einer Datenbank-Erstellungsmethode Standarderstellung aus.
6. Wählen Sie für Cache-Typ eine der folgenden Optionen aus:

   - Aurora (MySQL-kompatibel)
   - Aurora (PostgreSQL-kompatibel



7. Wählen Sie die Engine-Version.

   Weitere Informationen finden Sie unter Amazon-Aurora-Versionen. Sie können die Filter verwenden, um Versionen auszuwählen, die mit den gewünschten Funktionen kompatibel sind, z. B. Aurora Serverless v2. Weitere Informationen finden Sie unter Verwenden von Aurora Serverless v2.

8. Wählen Sie unter Vorlagen die Vorlage für Ihr Anwendungsszenario aus.

9. Gehen Sie wie folgt vor, um Ihr Masterpasswort einzugeben:

a. Öffnen Sie im Abschnitt Einstellungen die Option Einstellungen zu Anmeldeinformationen.

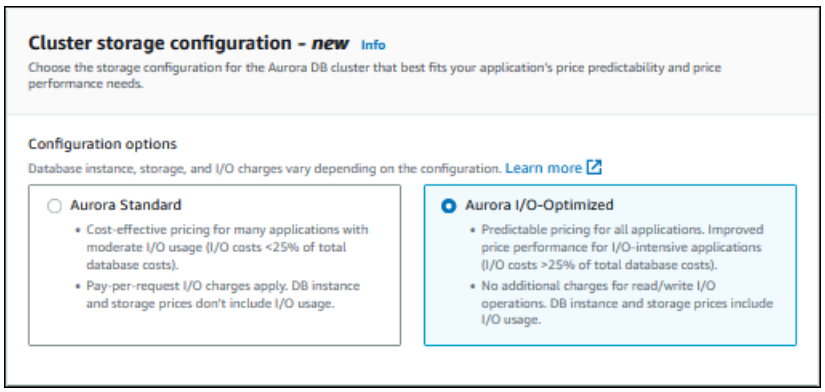
b. Deaktivieren Sie das Kontrollkästchen Ein Passwort automatisch erstellen.

c. (Optional) Ändern Sie den Wert für Hauptbenutzername und geben Sie in Hauptpasswort und Passwort bestätigen dasselbe Passwort ein.

Neu erstellte DB-Instances verwenden standardmäßig automatisch generierte Passwörter für den Hauptbenutzer.

10. Wenn Sie im Abschnitt Konnektivität unter VPC-Sicherheitsgruppe (Firewall) die Option Neu erstellen auswählen, wird eine VPC-Sicherheitsgruppe mit einer Regel für eingehenden Datenverkehr erstellt, die es der IP-Adresse Ihres lokalen Computers ermöglicht, auf die Datenbank zuzugreifen.

11. Wählen Sie für Cluster-Speicherkonfiguration entweder Aurora I/O-Optimized oder Aurora Standard aus. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.



12. (Optional) Richten Sie eine Verbindung zu einer Rechenressource für diesen DB-Cluster ein.

Sie können die Konnektivität zwischen einer Amazon-EC2-Instance und dem neuen DB-Cluster während der Erstellung eines DB-Clusters konfigurieren. Weitere Informationen finden Sie unter Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren.

13. In den übrigen Abschnitten geben Sie die Einstellungen für Ihren DB-Cluster an. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.

14. Wählen Sie Create database (Datenbank erstellen) aus.

Wenn Sie ein automatisch generiertes Passwort verwenden, wird auf der Seite Databases (Datenbanken) die Schaltfläche View credential details (Details zu Anmeldeinformationen anzeigen) angezeigt.

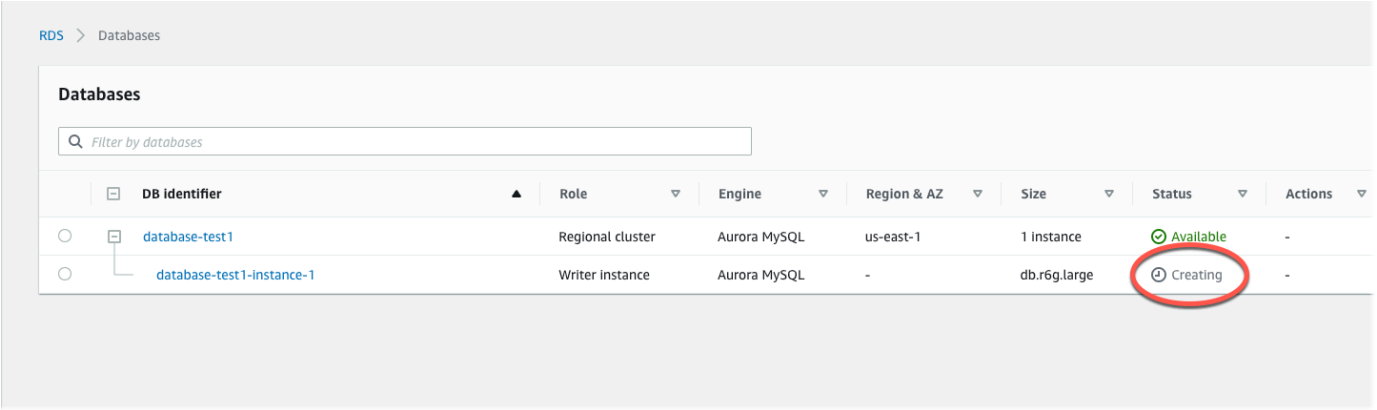
Um den Hauptbenutzernamen und das Passwort für den DB-Cluster anzuzeigen, wählen Sie Anmeldeinformationen anzeigen.

Verwenden Sie den angezeigten Benutzernamen und das angezeigte Passwort, um eine Verbindung zu DB-Instance als Hauptbenutzer herzustellen.

> **Important**
>
> Sie können dieses Passwort für den Hauptbenutzer nicht erneut anzeigen. Wenn Sie es nicht notieren, müssen Sie es möglicherweise ändern. Wenn Sie das Passwort für den Hauptbenutzer ändern müssen, nachdem die DB-Instance verfügbar wurde, können Sie die DB-Instance entsprechend ändern. Weitere Informationen über das Ändern einer DB-Instance finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

15. Wählen Sie unter Datenbanken den Namen des neuen Aurora-DB-Clusters aus.

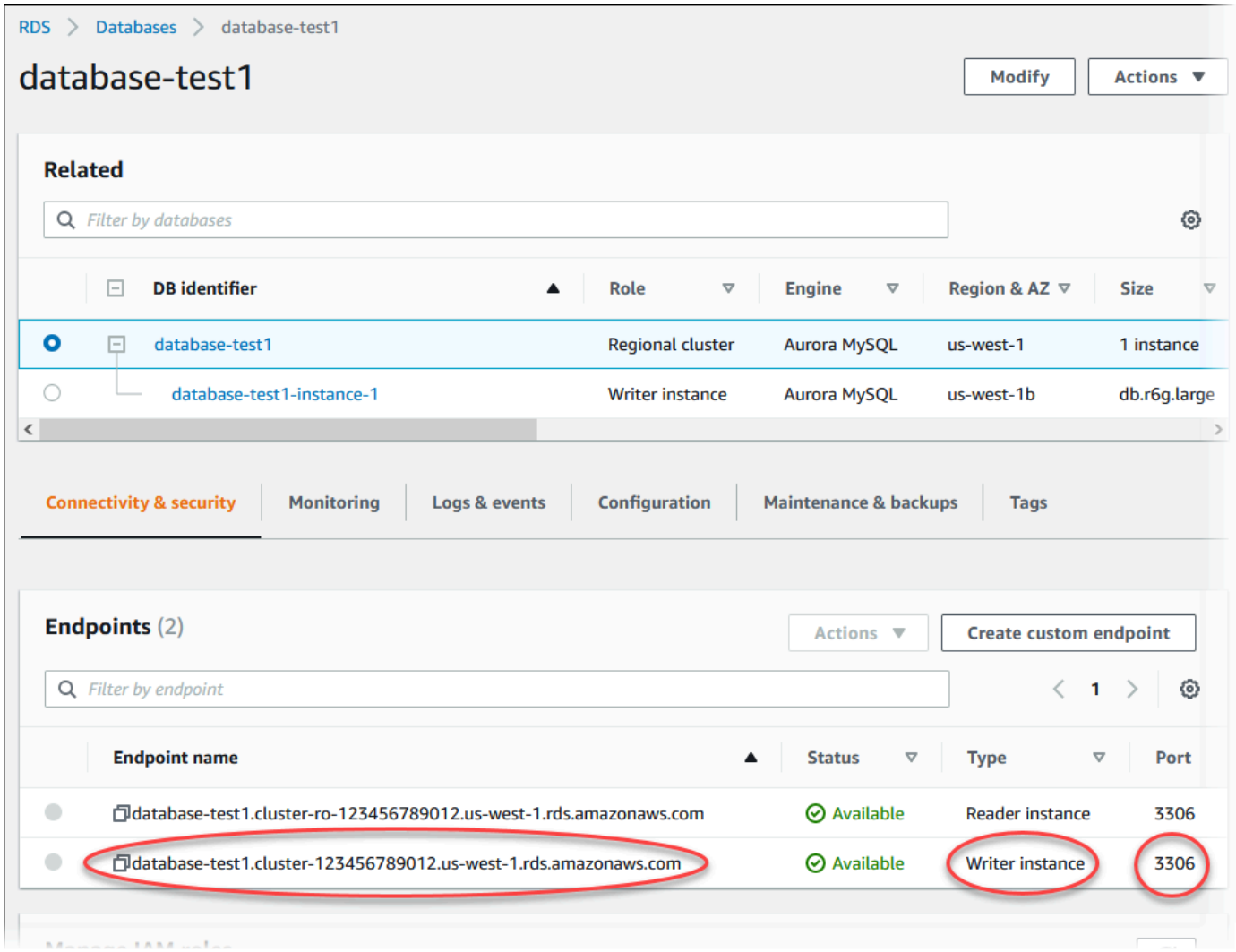
In der RDS-Konsole werden die Details des neuen DB-Clusters angezeigt. Bis der DB-Cluster verwendet werden kann, haben er und seine DB-Instance den Status Wird erstellt.



Wenn der Status zu Verfügbar geändert wird, können Sie eine Verbindung zum DB-Cluster herstellen. Je nach Klasse und Speicherort der DB-Instance kann es bis zu 20 Minuten dauern, bis der neue DB-Cluster verfügbar ist.

Um den neu erstellten Cluster anzuzeigen, wählen Sie Datenbanken im Navigationsbereich der Amazon-RDS-Konsole aus. Wählen Sie dann den DB-Cluster aus, um die DB-Cluster-Details

anzuzeigen. Weitere Informationen finden Sie unter Anzeigen eines DB-Clusters von Amazon Aurora.



Notieren Sie sich auf der Registerkarte Konnektivität und Sicherheit den Port und den Endpunkt der Writer-DB-Instance. Verwenden Sie diesen Endpunkt und Port in den JDBC- und ODBC-Verbindungszeichenfolgen aller Anwendungen, die Lese- oder Schreibvorgänge im Cluster ausführen.

AWS CLI

> **Note**
>
> Bevor Sie ein Aurora-DB-Cluster über die AWS CLI erstellen können, müssen Sie die Voraussetzungen erfüllen, wie das Erstellen einer VPC und einer RDS-DB-Subnetzgruppe. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.

Sie können die AWS CLI verwenden, um ein Aurora-MySQL-DB-Cluster oder ein Aurora-PostgreSQL-DB-Cluster zu erstellen.

So erstellen Sie ein Aurora-MySQL-DB-Cluster über die AWS CLI:

Wenn Sie einen mit Aurora MySQL 8.0 oder 5.7 kompatiblen DB-Cluster oder eine DB-Instance erstellen, geben Sie `aurora-mysql` für die Option `--engine` an.

Führen Sie folgende Schritte aus:

1. Geben Sie die DB-Subnetzgruppen- und VPC-Sicherheitsgruppen-ID für Ihr neues DB-Cluster an und rufen Sie dann den AWS CLI-Befehl create-db-cluster auf, um den Aurora-MySQL-DB-Cluster zu erstellen.

   Z. B. erstellt der folgende Befehl ein neues MySQL 8.0-kompatibles DB-Cluster mit dem Namen `sample-cluster`. Der Cluster verwendet die Standard-Engine-Version und den Speichertyp Aurora I/O-Optimized.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-mysql --engine-version 8.0 \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Für Windows:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-mysql --engine-version 8.0 ^
       --storage-type aurora-iopt1 ^
   ```

```
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

Der folgende Befehl erstellt ein neues MySQL 5.7-kompatibles DB-Cluster mit dem Namen `sample-cluster`. Der Cluster verwendet die Standard-Engine-Version und den Speichertyp Aurora Standard.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster  \
    --engine aurora-mysql --engine-version 5.7 \
    --storage-type aurora \
    --master-username user-name --manage-master-user-password \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

Für Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster sample-cluster  ^
    --engine aurora-mysql --engine-version 5.7 ^
    --storage-type aurora ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. Wenn Sie die Konsole für das Erstellen eines DB-Clusters verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihr DB-Cluster. Wenn Sie AWS CLI für das Erstellen eines DB-Clusters verwenden, müssen Sie die primäre Instance für Ihr DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance. Die DB-Cluster-Endpunkte bleiben im Status `Creating`, bis Sie die primäre DB-Instance erstellen.

   Rufen Sie den AWS CLI-Befehl create-db-instance auf, um die primäre Instance für Ihr DB-Cluster zu erstellen. Fügen Sie den Namen des DB-Clusters als `--db-cluster-identifier`-Optionswert ein.

   > **Note**
   >
   > Sie können die Option `--storage-type` für DB-Instances nicht festlegen. Sie können sie nur für DB-Cluster festlegen.

Der folgende Befehl erstellt beispielsweise eine neue MySQL 5.7-kompatible oder MySQL 8.0-kompatible DB-Instance namens `sample-instance`.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
     --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

Für Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
     --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

So erstellen Sie ein Aurora PostgreSQL-DB-Cluster über die AWS CLI

1. Geben Sie die DB-Subnetzgruppen- und VPC-Sicherheitsgruppen-ID für Ihr neues DB-Cluster an und rufen Sie dann den AWS CLI-Befehl create-db-cluster auf, um das Aurora-PostgreSQL-DB-Cluster zu erstellen.

   Beispielsweise erstellt der folgende Befehl ein neues DB-Cluster mit dem Namen `sample-cluster`. Der Cluster verwendet die Standard-Engine-Version und den Speichertyp Aurora I/O-Optimized.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-postgresql \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Für Windows:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-postgresql ^
       --storage-type aurora-iopt1 ^
   ```

```
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. Wenn Sie die Konsole für das Erstellen eines DB-Clusters verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihr DB-Cluster. Wenn Sie AWS CLI für das Erstellen eines DB-Clusters verwenden, müssen Sie die primäre Instance für Ihr DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance. Die DB-Cluster-Endpunkte bleiben im Status `Creating`, bis Sie die primäre DB-Instance erstellen.

   Rufen Sie den AWS CLI-Befehl create-db-instance auf, um die primäre Instance für Ihr DB-Cluster zu erstellen. Fügen Sie den Namen des DB-Clusters als `--db-cluster-identifier`-Optionswert ein.

   Für Linux, macOS oder Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
     --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

   Für Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
     --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

In diesen Beispielen wird die Option `--manage-master-user-password` zum Generieren des Hauptbenutzerpassworts und zum Verwalten dieses Passworts in Secrets Manager angegeben. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager. Alternativ können Sie die Option `--master-password` verwenden, um das Passwort selbst festzulegen und zu verwalten.

RDS-API

> **Note**
>
> Bevor Sie ein Aurora-DB-Cluster über die AWS CLI erstellen können, müssen Sie die Voraussetzungen erfüllen, wie das Erstellen einer VPC und einer RDS-DB-Subnetzgruppe. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.

Geben Sie die DB-Subnetzgruppe und die VPC-Sicherheitsgruppen-ID für Ihren neuen DB-Cluster an und rufen Sie anschließend die Aktion CreateDBCluster auf, um den DB-Cluster zu erstellen.

Wenn Sie einen DB-Cluster oder eine DB-Instance von Aurora MySQL Version 2 oder 3 geben Sie `aurora-mysql` für den Parameter `Engine` an.

Wenn Sie einen Aurora-PostgreSQL-DB-Cluster oder DB-Instances erstellen, müssen Sie `aurora-postgresql` für den Parameter `Engine` angeben.

Wenn Sie die Konsole für das Erstellen eines DB-Clusters verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihr DB-Cluster. Wenn Sie die RDS-API für das Erstellen eines DB-Clusters verwenden, müssen Sie die primäre Instance für Ihren DB-Cluster mit CreateDBInstance explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance. Die DB-Cluster-Endpunkte bleiben im Status `Creating`, bis Sie die primäre DB-Instance erstellen.

Erstellen einer primären (Writer-)DB-Instance

Wenn Sie die AWS-Managementkonsole für das Erstellen eines DB-Clusters verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihren DB-Cluster. Wenn Sie AWS CLI oder die RDS-API für das Erstellen eines DB-Clusters verwenden, müssen Sie die primäre Instance für den DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance. Die DB-Cluster-Endpunkte bleiben im Status `Creating`, bis Sie die primäre DB-Instance erstellen.

Weitere Informationen finden Sie unter Erstellen eines DB-Clusters.

> **Note**
>
> Wenn Sie einen DB-Cluster ohne Writer-DB-Instance haben, der auch als Headless-Cluster bezeichnet wird, können Sie die Konsole nicht verwenden, um eine Writer-Instance zu erstellen. Sie müssen die AWS CLI oder die RDS-API verwenden.

Im folgenden Beispiel wird der AWS CLI-Befehl create-db-instance verwendet, um eine Writer-Instance für einen Aurora PostgreSQL-DB-Cluster mit dem Namen `headless-test` zu erstellen.

```
aws rds create-db-instance \
    --db-instance-identifier no-longer-headless \
```

```
    --db-cluster-identifier headless-test \
    --engine aurora-postgresql \
    --db-instance-class db.t4g.medium
```

Einstellungen für Aurora-DB-Cluster

Die folgende Tabelle enthält Einzelheiten zu Einstellungen, die Sie beim Erstellen eines Aurora-DB-Clusters wählen können.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Automatisches Unterversion-Upgrade	Wählen Sie Automatische Aktualisierung von Unterversionen aktivieren aus, wenn Ihr Aurora-DB-Cluster automatisch bevorzugte Upgrades auf Unterversionen erhalten soll, sobald diese verfügbar werden. Die Einstellung Automatisches Unterversion-Upgrade gilt nur für Aurora-PostgreSQL- und Aurora-MySQL-DB-Cluster. Weitere Informationen über Engine-Updates für Aurora PostgreSQL finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL. Weitere Informationen über Engine-Updates für Aurora MySQL finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.	Legen Sie diesen Wert für jede DB-Instance in Ihrem Aurora-Cluster fest. Wenn für eine DB-Instance in Ihrem Cluster diese Einstellung deaktiviert ist, wird der Cluster nicht automatisch aktualisiert. Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` aus und legen Sie die Option `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie den Parameter `AutoMinorVersionUpgrade` fest.
AWS KMS key	Nur verfügbar, wenn Verschlüsselung auf Verschlüsselung	Bei Verwendung der AWS CLI: Führen Sie `create-db-`

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
	aktivieren festgelegt ist. Wählen Sie die AWS KMS key, die für die Verschlüsselung dieses DB-Clusters verwendet werden soll. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.	`cluster` aus und legen Sie die Option `--kms-key-id` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `KmsKeyId` fest.
Backtrack	Gilt nur für Aurora MySQL. Wählen Sie Enable Backtrack (Rückverfolgung aktivieren) aus, um die Rückverfolgung zu aktivieren, oder Disable Backtrack (Rückverfolgung deaktivieren), um die Rückverfolgung zu deaktivieren. Mit der Rückverfolgung können Sie einen DB-Cluster auf einen bestimmten Zeitpunkt zurückspulen, ohne einen neuen DB-Cluster zu erstellen. Standardmäßig ist die Rückverfolgung deaktiviert. Wenn Sie die Rückverfolgung aktivieren, geben Sie zusätzlich den Zeitraum an, für den die Rückverfolgung des DB-Clusters möglich sein soll (Zielfenster für die Rückverfolgung). Weitere Informationen finden Sie unter Rückverfolgen eines Aurora-DB-Clusters.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--backtrack-window` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `BacktrackWindow` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Zertifizierungsstelle	Die Zertifizierungsstelle (CA) für das Serverzertifikat, das von den DB-Instances im DB-Cluster verwendet wird. Weitere Informationen finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.	Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` aus und legen Sie die Option `--ca-certificate-identifier` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie den Parameter `CACertificateIdentifier` fest.
Cluster-Speicherkonfiguration	Der Speichertyp für den DB-Cluster: Aurora I/O-Optimized oder Aurora Standard. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--storage-type` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `StorageType` fest.
Tags zu Snapshots kopieren	Wählen Sie diese Option, wenn beim Erstellen eines Snapshots DB-Instance-Tags in den DB-Snapshot kopiert werden sollen. Weitere Informationen finden Sie unter Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--copy-tags-to-snapshot | --no-copy-tags-to-snapshot` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `CopyTagsToSnapshot` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Datenbank-Authentifizierung	Die Datenbank-Authentifizierung, die Sie verwenden möchten. Für MySQL: • Wählen Sie Passwortauthentifizierung aus, um Datenbankbenutzer ausschließlich mit Datenbankpasswörtern zu authentifizieren. • Wählen Sie Passwort- und IAM-Datenbank-Authentifizierung aus, um Datenbankbenutzer mit Datenbankpasswörtern und Benutzeranmeldeinformationen über IAM-Benutzer und -Rollen zu authentifizieren. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung. Für PostgreSQL: • Wählen Sie IAM database authentication (IAM-Datenbank-Authentifizierung) aus, um Datenbankbenutzer mit Datenbankpasswörtern und Benutzeranmeldeinformationen über Benutzer und Rollen zu authentifizieren. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung. • Wählen Sie Kerberos-Authentifizierung zur Authentifizierung	Um die IAM-Datenbank-Authentifizierung mit der AWS CLI durchzuführen, führen Sie `create-db-cluster` aus und legen Sie die `--enable-iam-database-authentication | --no-enable-iam-database-authentication` -Option fest. Um die IAM-Datenbank-Authentifizierung mit der RDS API zu verwenden, rufen Sie `CreateDBCluster` auf und legen Sie die `EnableIAMDatabaseAuthentication` -Parameter fest. Um die Kerberos-Authentifizierung mit der AWS CLI durchzuführen, führen Sie `create-db-cluster` aus und legen Sie die `--domain-` und `--domain-iam-role-name` -Optionen fest. Um die Kerberos-Authentifizierung mit der RDS API zu verwenden, rufen Sie `CreateDBCluster` auf und legen Sie die `Domain`- und `DomainIAMRoleName` -Parameter fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
	von Datenbankkennwörtern und Benutzeranmeldeinformationen mit Kerberos-Authentifizierung. Weitere Informationen finden Sie unter Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL.	
Datenbankport	Geben Sie den Port an, über den Anwendungen und Dienstprogramme auf die Datenbank zugreifen können. Aurora MySQL-DB-Cluster verwenden standardmäßig den MySQL-Standardport 3306. Aurora-PostgreSQL-DB-Cluster verwenden standardmäßig den PostgreSQL-Standardport 5432. Die Firewalls einiger Unternehmen blockieren Verbindungen zu diesen Standard-Ports. Wenn die Firewall Ihres Unternehmens den Standardport blockiert, wählen Sie einen anderen Port für den neuen DB-Cluster aus.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--port` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `Port` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
DB-Cluster-Kennung	Geben Sie einen Namen für Ihren DB-Cluster ein, der für Ihr Konto in der von Ihnen gewählten AWS-Region eindeutig ist. Dieser Bezeichner wird in der Cluster-Endpunktadresse für den DB-Cluster verwendet. Weitere Informationen zum Cluster-Endpunkt finden Sie unter Amazon-Aurora-Endpunktverbindungen. Für den DB-Cluster-Bezeichner gelten folgende Einschränkungen: • Sie darf 1 bis 63 alphanumerische Zeichen oder Bindestriche enthalten. • Das erste Zeichen muss ein Buchstabe sein. • Er darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten. • Er muss für alle DB-Cluster pro AWS-Konto und AWS-Region eindeutig sein.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--db-cluster-identifier` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `DBClusterIdentifier` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
DB-Cluster-Parametergruppe	Wählen Sie eine DB-Cluster-Parametergruppe. Aurora verfügt über eine DB-Cluster-Standardparametergruppe, die Sie verwenden können, oder Sie können Ihre eigene DB-Cluster-Parametergruppe erstellen. Weitere Informationen zu DB-Cluster-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--db-cluster-parameter-group-name` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `DBClusterParameterGroupName` fest.
DB-Instance-Klasse	Gilt nur für den bereitgestellten Kapazitätstyp. Wählen Sie eine DB-Instance-Klasse aus, die die Verarbeitungs- und Speicheranforderungen für jede Instance im DB-Cluster definiert. Weitere Informationen zu DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.	Legen Sie diesen Wert für jede DB-Instance in Ihrem Aurora-Cluster fest. Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` aus und legen Sie die Option `--db-instance-class` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie den Parameter `DBInstanceClass` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
DB-Parametergruppe	Wählen Sie eine Parametergruppe. Aurora verfügt über eine Standardparametergruppe, die Sie verwenden können, oder Sie können Ihre eigene Parametergruppe erstellen. Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.	Legen Sie diesen Wert für jede DB-Instance in Ihrem Aurora-Cluster fest. Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` aus und legen Sie die Option `--db-parameter-group-name` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie den Parameter `DBParameterGroupName` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
DB-Subnetzgruppe	Die DB-Subnetzgruppe, die Sie für den DB-Cluster verwenden möchten. Wählen Sie Choose existing (Vorhandene wählen) aus, um eine vorhandene DB-Subnetzgruppe zu verwenden. Wählen Sie dann die erforderliche Subnetzgruppe aus der Dropdown-Liste Existing DB subnet groups (Vorhandene DB-Subnetzgruppen) aus. Wählen Sie Automatic setup (Automatische Einrichtung) aus, damit RDS eine kompatible DB-Subnetzgruppe auswählen kann. Wenn keine vorhanden ist, erstellt RDS eine neue Subnetzgruppe für Ihren Cluster. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--db-subnet-group-name` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `DBSubnetGroupName` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Löschschutz aktivieren	Wählen Sie Löschschutz aktivieren aus, um zu verhindern, dass Ihr DB-Cluster gelöscht wird. Wenn Sie einen Produktions-DB-Cluster über die Konsole erstellen, ist der Löschschutz standardmäßig aktiviert.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--deletion-protection | --no-deletion-protection` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `DeletionProtection` fest.
Verschlüsselung aktivieren	Wählen Sie `Enable encryption` , um die Verschlüsselung von Daten bei der Speicherung für diesen DB-Cluster zu aktivieren. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--storage-encrypted | --no-storage-encrypted` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `StorageEncrypted` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Erweiterte Überwachung aktivieren	Wählen Sie Enable enhanced monitoring (Erweiterte Überwachung aktivieren) aus, um die Erfassung von Metriken in Echtzeit für das Betriebssystem zu aktivieren, in dem Ihr DB-Cluster ausgeführt wird. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).	Legen Sie diese Werte für jede DB-Instance in Ihrem Aurora-Cluster fest. Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` und legen Sie die Optionen `--monitoring-interval` und `--monitoring-role-arn` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie die Parameter `MonitoringInterval` und `MonitoringRoleArn` fest.
RDS-Daten-API aktivieren	Wählen Sie RDS-Daten-API aktivieren aus, um die RDS-Daten-API (Daten-API) zu aktivieren. Die Daten-API bietet einen sicheren HTTP-Endpunkt für die Ausführung von SQL-Anweisungen, ohne Verbindungen zu verwalten. Weitere Informationen finden Sie unter Verwenden der Daten-API von Amazon RDS.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--enable-http-endpoint | --no-enable-http-endpoint` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `EnableHttpEndpoint` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Engine-Typ	Wählen Sie den Namen der Datenbank-Engine aus, die für diesen DB-Cluster verwendet werden soll.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--engine` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `Engine` fest.
Engine-Version	Gilt nur für den bereitgestellten Kapazitätstyp. Wählen Sie die Versionsnummer der Datenbank-Engine.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--engine-version` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `EngineVersion` fest.
Failover-Priorität	Wählen Sie eine Failover-Priorität für die Instance aus. Wenn Sie keinen Wert auswählen, wird als Standard tier-1 (Tier-1) eingestellt. Diese Priorität bestimmt die Reihenfolge, in der Aurora-Replikate bei der Wiederherstellung nach einem Ausfall der primären Instance hochgestuft werden. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.	Legen Sie diesen Wert für jede DB-Instance in Ihrem Aurora-Cluster fest. Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` aus und legen Sie die Option `--promotion-tier` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie den Parameter `PromotionTier` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Anfänglicher Datenbankname	Geben Sie einen Namen für Ihre Standarddatenbank ein. Wenn Sie keinen Namen für einen Aurora MySQL-DB-Cluster angeben, erstellt Amazon RDS nicht automatisch eine Datenbank im neuen DB-Cluster. Wenn Sie keinen Namen für einen Aurora PostgreSQL-DB-Cluster angeben, erstellt Amazon RDS eine Datenbank mit dem Namen `postgres`. Für Aurora MySQL hat der Standarddatenbankname diese Einschränkungen: • Er muss 1–64 alphanumerische Zeichen enthalten. • Es darf kein von der Datenbank-Engine reserviertes Wort sein. Für Aurora PostgreSQL hat der Standarddatenbankname diese Einschränkungen: • Er muss 1–63 alphanumerische Zeichen enthalten. • Er muss mit einem Buchstaben beginnen. Nachfolgende Zeichen können Buchstaben, Unterstriche oder Ziffern sein (0–9).	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--database-name` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `DatabaseName` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
	• Es darf kein von der Datenbank-Engine reserviertes Wort sein. Verbinden Sie sich mit dem DB-Cluster und verwenden Sie den SQL-Befehl CREATE DATABASE, um zusätzliche Datenbanken zu erstellen. Weitere Informationen zum Herstellen einer Verbindung mit dem DB-Cluster finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.	
Protokollexporte	Wählen Sie im Bereich Protokollexporte die Protokolle aus, die Sie in Amazon CloudWatch Logs veröffentlichen möchten. Weitere Informationen über das Veröffentlichen von Aurora MySQL-Protokollen für Amazon CloudWatch Logs finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs. Weitere Informationen über das Veröffentlichen von Aurora PostgreSQL-Protokollen für Amazon CloudWatch Logs finden Sie unter Veröffentlichen von Aurora-PostgreSQL-Protokollen in Amazon CloudWatch Logs.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--enable-cloudwatch-logs-exports` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `EnableCloudwatchLogsExports` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Wartungsfenster	Wählen Sie Fenster auswählen aus und geben Sie den wöchentlichen Zeitraum an, in dem Systemwartungen durchgeführt werden können. Oder wählen Sie für Amazon RDS Keine Präferenz aus, um zufällig einen Zeitraum zuzuordnen.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--preferred-maintenance-window` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `PreferredMaintenanceWindow` fest.
Master-Anmeldeinformationen verwalten in AWS Secrets Manager	Wählen Sie Master-Anmeldeinformationen verwalten in AWS Secrets Manager aus, um das Hauptbenutzerpasswort in Secrets Manager geheim zu verwalten. Wählen Sie optional einen KMS-Schlüssel zum Schutz des Secrets aus. Wählen Sie aus den KMS-Schlüsseln in Ihrem Konto oder geben Sie den Schlüssel eines anderen Kontos ein. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` und legen Sie die Optionen `--manage-master-user-password | --no-manage-master-user-password` und `--master-user-secret-kms-key-id` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie die Parameter `ManageMasterUserPassword` und `MasterUserSecretKmsKeyId` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Hauptpasswort	Geben Sie ein Passwort ein, um sich an Ihrem DB-Cluster anzumelden: • Für Aurora MySQL muss das Passwort 8–41 druckbare ASCII-Zeichen enthalten. • Für Aurora PostgreSQL muss es 8–99 druckbare ASCII-Zeichen enthalten. • Es darf weder `/`, `"`, `@` noch ein Leerzeichen enthalten.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--master-user-password` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `MasterUserPassword` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Master-Benutzername	Geben Sie einen Namen ein, der als Master-Benutzername für die Anmeldung an Ihrem DB-Cluster verwendet werden soll: • Für Aurora MySQL muss der Name 1–16 alphanumerische Zeichen enthalten. • Für Aurora PostgreSQL muss er 1–63 alphanumerische Zeichen enthalten. • Das erste Zeichen muss ein Buchstabe sein. • Der Name darf kein von der Datenbank-Engine reserviertes Wort sein. Sie können den Master-Benutzernamen nicht ändern, nachdem der DB-Cluster erstellt wurde.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--master-username` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `MasterUsername` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Multi-AZ-Bereitstellung	Gilt nur für den bereitgestellten Kapazitätstyp. Legen Sie fest, ob Aurora-Replikate in anderen Availability Zones zur Failover-Unterstützung erstellt werden sollen. Wenn Sie Create Replica in Different Zone (Replica in anderer Zone erstellen) auswählen, erstellt Amazon RDS in Ihrem DB-Cluster eine Aurora-Replica in einer anderen Availability Zone als der, in dem sich die primäre Instance Ihres DB-Clusters befindet. Weitere Informationen über die Multi-AZ-Bereitstellung finden Sie unter Regionen und Availability Zones.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--availability-zones` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `AvailabilityZones` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Netzwerktyp	Die vom DB-Cluster unterstützten IP-Adressierungsprotokolle. IPv4, um anzugeben, dass Ressourcen mit dem DB-Cluster nur über das IPv4-Adressierungsprotokoll kommunizieren können. Dual-Stack-Modus, um anzugeben, dass Ressourcen mit dem DB-Cluster über IPv4, IPv6 oder beiden kommunizieren können. Verwenden Sie den Dual-Stack-Modus, wenn Sie über Ressourcen verfügen, die über das IPv6-Adressierungsprotokoll mit Ihrem DB-Cluster kommunizieren müssen. Wenn Sie den Dual-Stack-Modus verwenden möchten, stellen Sie sicher, dass mindestens zwei Subnetze über zwei Availability Zones verteilt sind die sowohl das IPv4- als auch das IPv6-Netzwerkprotokoll unterstützen. Stellen Sie außerdem sicher, dass Sie einen IPv6-CIDR-Block mit allen Subnetzen in der von Ihnen angegebenen DB-Subnetzgruppe verknüpfen. Weitere Informationen finden Sie unter Amazon-Aurora-IP-Adressierung.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `-network-type` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `NetworkType` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Öffentlicher Zugriff	Wählen Sie Publicly accessible (Öffentlich zugänglich), um dem DB-Cluster eine öffentliche IP-Adresse zu geben, oder wählen Sie Not publicly accessible (Nicht öffentlich zugänglich). Die Instances in Ihrem DB-Cluster können eine Mischung aus öffentlichen und privaten DB-Instances sein. Weitere Informationen darüber, wie Sie den öffentlichen Zugriff für Instances deaktivieren, finden Sie unter Ausblenden einer DB-Clusters in einer VPC vor dem Internet. Um eine Verbindung zu einer DB-Instance von außerhalb ihrer Amazon VPC herzustellen, muss die DB-Instance öffentlich zugänglich sein und der Zugriff muss unter Anwendung der Regeln für den eingehenden Datenverkehr der Sicherheitsgruppe der DB-Instance gewährt werden. Darüber hinaus müssen weitere Anforderungen erfüllt werden. Weitere Informationen finden Sie unter Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden. Wenn Ihre DB-Instance nicht öffentlich zugänglich ist, können	Legen Sie diesen Wert für jede DB-Instance in Ihrem Aurora-Cluster fest. Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` aus und legen Sie die Option `--publicly-accessible | --no-publicly-accessible` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie den Parameter `PubliclyAccessible` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
	Sie auch eine AWS-Site-to-Site-VPN-Verbindung oder eine Direct Connect-Verbindung verwenden, um über ein privates Netzwerk darauf zuzugreifen. Weitere Informationen finden Sie unter Richtlinie für den Datenverkehr zwischen Netzwerken.	
RDS Extended Support	Wählen Sie Enable RDS Extended Support aus, damit unterstützte Engine-Hauptversionen auch nach dem Ende des Standard-Supports von Aurora weiter ausgeführt werden können. Wenn Sie einen DB-Cluster erstellen, verwendet Amazon Aurora standardmäßig RDS Extended Support. Um zu verhindern, dass nach dem Ende des Standard-Supports von Aurora ein neuer DB-Cluster erstellt wird, und um Gebühren für RDS Extended Support zu vermeiden, deaktivieren Sie diese Einstellung. Für Ihre vorhandenen DB-Cluster fallen bis zum Startdatum der Preise für die erweiterte RDS-Unterstützung keine Gebühren an. Weitere Informationen finden Sie unter Amazon RDS Extended Support mit Amazon Aurora.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--engine-lifecycle-support` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `EngineLifecycleSupport` fest.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
RDS-Proxy	Wählen Sie Create an RDS-Proxy (RDS-Proxy erstellen) aus, um einen Proxy für Ihren DB-Cluster zu erstellen. Amazon RDS erstellt automatisch eine IAM-Rolle und ein Secrets-Manager-Secret für den Proxy. Weitere Informationen finden Sie unter Amazon-RDS-Proxy für Aurora.	Nicht verfügbar beim Erstellen eines DB-Clusters
Aufbewahrungszeitraum	Wählen Sie den Aufbewahrungszeitraum (1 bis 35 Tage) für Sicherungskopien der Datenbank in Aurora aus. Sicherungskopien können für sekundengenaue Point-in-Time-Wiederherstellungen (PITR) der Datenbank verwendet werden.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--backup-retention-period` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `BackupRetentionPeriod` fest.
DevOps Guru aktivieren	Klicken Sie auf DevOps Guru aktivieren, um Amazon DevOps Guru für Ihre Aurora-Datenbank zu aktivieren. Damit DevOps Guru for RDS eine detaillierte Analyse von Leistungsanomalien bereitstellen kann, muss Performance Insights aktiviert sein. Weitere Informationen finden Sie unter DevOpsGuru für RDS einrichten.	Sie können DevOps Guru for RDS von der RDS-Konsole aus, jedoch nicht mithilfe der RDS-API oder CLI aktivieren. Weitere Informationen zum Aktivieren von DevOps Guru finden Sie im Amazon-DevOps-Guru-Benutzerhandbuch.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
Performance Insights aktivieren	Klicken Sie auf Performance Insights aktivieren, um Erkenntnisse zur Amazon-RDS-Leistung zu aktivieren. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .	Legen Sie diese Werte für jede DB-Instance in Ihrem Aurora-Cluster fest. Bei Verwendung der AWS CLI: Führen Sie `create-db-instance` aus und legen Sie die Optionen `--enable-performance-insights | --no-enable-performance-insights` , `--performance-insights-kms-key-id` und `--performance-insights-retention-period` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBInstance` auf und legen Sie die Parameter `EnablePerformanceInsights` , `PerformanceInsightsKMSKeyId` und `PerformanceInsightsRetentionPeriod` fest.
Virtual Private Cloud (VPC)	Wählen Sie die VPC zum Hosten des DB-Clusters aus. Wählen Sie Create a new VPC (Neue VPC erstellen) aus, damit Amazon RDS eine VPC für Sie erstellt. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.	Für die AWS CLI und API geben Sie die VPC-Sicherheitsgruppen-IDs an.

Konsoleneinstellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
VPC-Sicherheitsgruppe (Firewall)	Wählen Sie Neu erstellen aus, damit Amazon RDS eine VPC-Sicherheitsgruppe für Sie erstellt. Oder wählen Sie Vorhandene auswählen aus und geben Sie eine oder mehrere VPC-Sicherheitsgruppen an, um den Netzwerkzugriff auf den DB-Cluster zu sichern. Wenn Sie in der RDS-Konsole die Option Neu erstellen auswählen, wird eine Sicherheitsgruppe mit einer Regel für eingehenden Datenverkehr erstellt, die den Zugriff auf die DB-Instance von der im Browser erkannten IP-Adresse aus gestattet. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.	Bei Verwendung der AWS CLI: Führen Sie `create-db-cluster` aus und legen Sie die Option `--vpc-security-group-ids` fest. Bei Verwendung der RDS API: Rufen Sie `CreateDBCluster` auf und legen Sie den Parameter `VpcSecurityGroupIds` fest.

Einstellungen, die nicht für Amazon-Aurora-DB-Cluster gelten

Die folgenden Einstellungen im AWS CLI-Befehl `create-db-cluster` und in der RDS-API-Operation `CreateDBCluster` werden nicht auf Amazon-Aurora-DB-Cluster angewendet.

> **Note**
>
> Die AWS-Managementkonsole zeigt diese Einstellungen für Aurora-DB-Cluster nicht an.

AWS CLI-Einstellung	RDS-API-Einstellung
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`
`--publicly-accessible | --no-publicly-accessible`	`PubliclyAccessible`

Einstellungen, die nicht für Amazon-Aurora-DB-Instances gelten

Die folgenden Einstellungen im AWS CLI-Befehl `create-db-instance` und in der RDS-API-Operation `CreateDBInstance` werden nicht auf DB-Instances-Amazon-Aurora-DB-Cluster angewendet.

> **Note**
>
> Die AWS-Managementkonsole zeigt diese Einstellungen für Aurora-DB-Instances nicht an.

AWS CLI-Einstellung	RDS-API-Einstellung
`--allocated-storage`	`AllocatedStorage`
`--availability-zone`	`AvailabilityZone`
`--backup-retention-period`	`BackupRetentionPeriod`
`--backup-target`	`BackupTarget`
`--character-set-name`	`CharacterSetName`
`--character-set-name`	`CharacterSetName`
`--custom-iam-instance-profile`	`CustomIamInstanceProfile`
`--db-security-groups`	`DBSecurityGroups`
`--deletion-protection | --no-deletion-protection`	`DeletionProtection`
`--domain`	`Domain`
`--domain-iam-role-name`	`DomainIAMRoleName`
`--enable-cloudwatch-logs-exports`	`EnableCloudwatchLogsExports`
`--enable-customer-owned-ip | --no-enable-customer-owned-ip`	`EnableCustomerOwnedIp`
`--enable-iam-database-authentication | --no-enable-iam-database-authentication`	`EnableIAMDatabaseAuthentication`
`--engine-version`	`EngineVersion`
`--iops`	`Iops`
`--kms-key-id`	`KmsKeyId`
`--master-username`	`MasterUsername`

AWS CLI-Einstellung	RDS-API-Einstellung
`--master-user-password`	`MasterUserPassword`
`--max-allocated-storage`	`MaxAllocatedStorage`
`--multi-az | --no-multi-az`	`MultiAZ`
`--nchar-character-set-name`	`NcharCharacterSetName`
`--network-type`	`NetworkType`
`--option-group-name`	`OptionGroupName`
`--preferred-backup-window`	`PreferredBackupWindow`
`--processor-features`	`ProcessorFeatures`
`--storage-encrypted | --no-storage-encrypted`	`StorageEncrypted`
`--storage-type`	`StorageType`
`--tde-credential-arn`	`TdeCredentialArn`
`--tde-credential-password`	`TdeCredentialPassword`
`--timezone`	`Timezone`
`--vpc-security-group-ids`	`VpcSecurityGroupIds`

Amazon-Aurora-Ressourcen erstellen mit AWS CloudFormation

Amazon Aurora ist in AWS CloudFormation integriert, welches ein Service ist, der Ihnen hilft, Ihre AWS-Ressourcen zu modellieren und einzurichten, damit Sie weniger Zeit mit der Erstellung und Verwaltung Ihrer Ressourcen und Infrastruktur verbringen können. Sie erstellen eine Vorlage, die alle gewünschten AWS Ressourcen beschreibt (wie DB-Cluster und DB-Cluster-Parametergruppen), CloudFormation und diese Ressourcen für Sie bereitstellt und konfiguriert.

Wenn Sie CloudFormation verwenden, können Sie Ihre Vorlage wiederverwenden, um Ihre Aurora-Ressourcen konsistent und wiederholt einzurichten. Sie beschreiben Ihre Ressourcen dann einmal und können die gleichen Ressourcen dann in mehreren AWS-Konten und -Regionen immer wieder bereitstellen.

Aurora und CloudFormation-Vorlagen

CloudFormation-Vorlagen sind formatierte Textdateien in JSON oder YAML. Diese Vorlagen beschreiben die Ressourcen, die Sie in Ihren CloudFormation-Stacks bereitstellen möchten. Wenn Sie noch keine Erfahrungen mit JSON oder YAML haben, können Sie CloudFormation Designer verwenden, der den Einstieg in die Arbeit mit CloudFormation-Vorlagen erleichtert. Weitere Informationen finden Sie unter Was ist CloudFormation-Designer? im AWS CloudFormation-Benutzerhandbuch.

Aurora unterstützt das Erstellen von Ressourcen in CloudFormation. Weitere Informationen, einschließlich Beispiele für JSON- und YAML-Vorlagen für diese Ressourcen, finden Sie in der Referenz zum RDS-Ressourcentyp im AWS CloudFormation-Benutzerhandbuch.

Weitere Informationen zu CloudFormation

Weitere Informationen zu CloudFormation finden Sie in den folgenden Ressourcen.

- AWS CloudFormation
- AWS CloudFormation-Benutzerhandbuch
- CloudFormation API Referenz
- AWS CloudFormation-Benutzerhandbuch für die Befehlszeilenschnittstelle

Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster

Sie können mit denselben Tools eine Verbindung zu einer DB-Instance in einem Aurora-DB-Cluster einrichten, die Sie UCH für die Verbindung mit einer MySQL- oder PostgreSQL-Datenbank verwenden. Sie geben eine Verbindungszeichenfolge mit einem beliebigen Skript, einem Hilfsprogramm oder einer Anwendung an, das/die sich mit einer MySQL- oder PostgreSQL-DB-Instance verbindet. Hierbei können Sie den gleichen öffentlichen Schlüssel für Secure Sockets Layer (SSL)-Verbindungen nutzen.

In der Verbindungszeichenfolge verwenden Sie typischerweise die Host- und Port-Informationen von besonderen mit dem DB-Cluster verbundenen Endpunkten. Mit diesen Endpunkten können Sie dieselben Verbindungsparameter verwenden, unabhängig davon, wie viele DB-Instances sich in dem Cluster befinden. Sie können die Host- und Port-Informationen aus einer spezifischen DB-Instance in Ihrem Aurora-DB_Cluster für spezielle Aufgaben wie die Fehlerbehebung verwenden.

> **Note**
>
> Bei Aurora Serverless-DB-Clustern stellen Sie eine Verbindung zum Datenbank-Endpunkt statt mit der DB-Instance her. Sie finden den Datenbank-Endpunkt für einen Aurora Serverless-DB-Cluster auf der Registerkarte Connectivity & security (Konnektivität und Sicherheit) von AWS-Managementkonsole. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Serverless v1.

Unabhängig von der Aurora-DB-Engine und spezifischen Tools, die Sie für die Arbeit mit dem DB-Cluster oder der Instance verwenden, muss der Endpunkt zugänglich sein. Ein Aurora-DB-Cluster kann in einer Virtual Private Cloud (VPC) nur basierend auf dem Amazon-VPC-Service erstellt werden. Das bedeutet, dass Sie mit einem der folgenden Ansätze entweder innerhalb der VPC oder außerhalb der VPC auf den Endpunkt zugreifen.

- Greifen Sie auf den Aurora-DB-Cluster innerhalb der VPC zu – Aktivieren Sie den Zugriff auf den Aurora-DB-Cluster über die VPC. Sie tun dies, indem Sie die Inbound-Regeln der Sicherheitsgruppe für die VPC bearbeiten, um den Zugriff auf Ihren spezifischen Aurora-DB-Cluster zu ermöglichen. Weitere Informationen, einschließlich der Konfiguration Ihrer VPC für verschiedene Aurora-DB-Cluster-Szenarien, finden Sie unter Amazon Virtual Private Cloud VPCs und Amazon Aurora.

- Greifen Sie auf den Aurora-DB-Cluster außerhalb der VPC zu – Verwenden Sie die öffentliche Endpunktadresse des DB-Clusters, um von außerhalb der VPC auf einen Aurora-DB-Cluster zuzugreifen.

Weitere Informationen finden Sie unter Behebung von Aurora-Verbindungsfehlern.

Inhalt

- Herstellen einer Verbindung mit Aurora-DB-Clustern mithilfe der AWS-Treiber
- Herstellen einer Verbindung mit einem Amazon Aurora MySQL-DB-Cluster
  - Hilfsprogramme für die Herstellung von Verbindungen für Aurora MySQL
  - Herstellen einer Verbindung mit Aurora MySQL über das MySQL-Dienstprogramm
  - Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-JDBC-Treibers
  - Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-Python-Treibers
  - Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-ODBC-Treibers für MySQL
  - Herstellen einer Verbindung mit Amazon MySQL mithilfe des AWS-Advanced-NodeJS-Wrappers
  - Herstellen einer Verbindung mit Aurora MySQL mithilfe von SSL
- Herstellen einer Verbindung mit einem Amazon-Aurora-PostgreSQL-DB-Cluster
  - Hilfsprogramme für die Herstellung von Verbindungen für Aurora PostgreSQL
  - Herstellen einer Verbindung mit Aurora PostgreSQL mithilfe des AWS-JDBC-Treibers
  - Herstellen einer Verbindung mit Aurora PostgreSQL mithilfe des AWS-Python-Treibers
  - Herstellen einer Verbindung mit Amazon PostgreSQL mithilfe des AWS-Advanced-NodeJS-Wrappers
- Behebung von Aurora-Verbindungsfehlern

Herstellen einer Verbindung mit Aurora-DB-Clustern mithilfe der AWS-Treiber

Die AWS-Treibersuite wurde so konzipiert, dass sie schnellere Umstellungs- und Failover-Zeiten sowie die Authentifizierung mit AWS Secrets Manager, AWS Identity and Access Management (IAM) und Verbundidentität unterstützt. Die AWS-Treiber müssen den Status des DB-Clusters überwachen und die Clustertopologie kennen, um den neuen Writer zu ermitteln. Dieser Ansatz reduziert die Umstellungs- und Failover-Zeiten auf Werte im einstelligen Sekundenbereich, im Vergleich zu Werten im zweistelligen Bereich bei Open-Source-Treibern.

In der folgenden Tabelle sind die Funktionen aufgeführt, die für jeden Treiber unterstützt werden. Im Zuge der Einführung neuer Servicefunktionen besteht das Ziel der AWS-Treibersuite darin, eine integrierte Unterstützung für diese Servicefunktionen zu bieten.

Funktion	AWS-JDBC-Treiber	AWS-Python-Treiber	AWS-ODBC-Treiber für MySQL	AWS-Advanced-NodeJS-Wrapper
Failover-Unterstützung	Yes (Ja)	Ja	Ja	Yes (Ja)
Erweiterte Failover-Überwachung	Yes (Ja)	Ja	Ja	Yes (Ja)
Aufteilung von Lese-/Schreibvorgängen	Yes (Ja)	Ja	Nein	Yes (Ja)
Aurora-Verbindungs-Tracker	Yes (Ja)	Ja	Nein	Yes (Ja)
Verbindung mit Treiber-Metadaten	Yes (Ja)	–	–	N/A
Telemetrie	Yes (Ja)	Ja	Nein	Yes (Ja)
Secrets Manager	Yes (Ja)	Ja	Ja	Yes (Ja)
IAM-Authentifizierung	Yes (Ja)	Ja	Ja	Yes (Ja)
Verbundidentität (AD FS)	Yes (Ja)	Ja	Nein	Yes (Ja)
Verbundidentität (Okta)	Yes (Ja)	Ja	Ja	Yes (Ja)

Funktion	AWS-JDBC-Treiber	AWS-Python-Treiber	AWS-ODBC-Treiber für MySQL	AWS-Advanced-NodeJS-Wrapper
Aurora PostgreSQL Limitless Database	Ja (nur Aurora PostgreSQL)	Nein	Nein	Ja (nur Aurora PostgreSQL)

Weitere Informationen zu den AWS-Treibern finden Sie in der Dokumentation zum Treiber der entsprechenden Sprache für Ihren Aurora-MySQL- oder Aurora-PostgreSQL-DB-Cluster.

Herstellen einer Verbindung mit einem Amazon Aurora MySQL-DB-Cluster

Um das Aurora-MySQL-DB-Cluster zu authentifizieren, können Sie entweder den MySQL-Authentifizierung mit Benutzername und Passwort oder die AWS Identity and Access Management-(IAM)-Datenbankauthentifizierung verwenden. Weitere Informationen über die Verwendung der MySQL-Benutzernamen- und Passwort-Authentifizierung finden Sie unter Access Control and Account Management in der MySQL-Dokumentation. Weitere Informationen zur Verwendung der IAM-Datenbankauthentifizierung finden Sie unter IAM-Datenbankauthentifizierung.

Wenn Sie eine Verbindung mit Ihrem Amazon-Aurora-DB-Cluster mit MySQL 8.0-Kompatibilität hergestellt haben, können Sie SQL-Befehle ausführen, die mit MySQL 8.0 kompatibel sind. Die minimale kompatible Version ist MySQL 8.0.23. Weitere Informationen zur MySQL 8.0-SQL-Syntax finden Sie im MySQL 8.0-Referenzhandbuch. Informationen zu Einschränkungen für Aurora-MySQL-Version 3 finden Sie unter Vergleich von Aurora-MySQL-Version 3 und MySQL 8.0 Community Edition.

Wenn Sie eine Verbindung mit Ihrem Amazon-Aurora-DB-Cluster mit MySQL 5.7-Kompatibilität hergestellt haben, können Sie SQL-Befehle ausführen, die mit MySQL 5.7 kompatibel sind. Weitere Informationen zur MySQL 5.7-SQL-Syntax finden Sie im MySQL 5.7-Referenzhandbuch. Informationen zu Einschränkungen für Aurora MySQL 5.7 finden Sie unter Aurora-MySQL-Version 2, kompatibel mit MySQL 5.7.

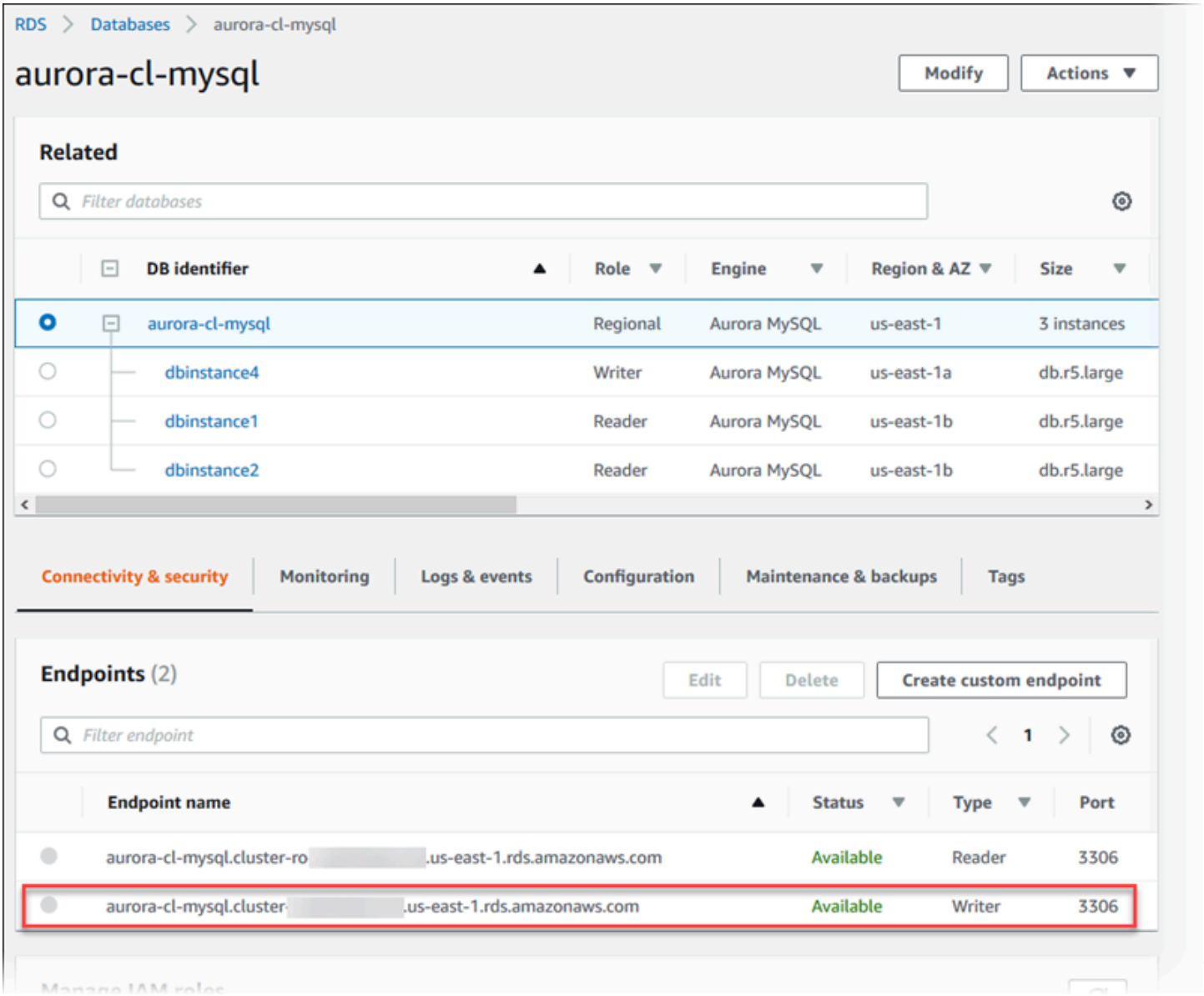
> **Note**
>
> Eine nützliche und detaillierte Anleitung für die Herstellung von Verbindungen mit einem Amazon Aurora MySQL-DB-Cluster finden Sie im Handbuch für das Aurora-Verbindungsmanagement.

In der Detailansicht für Ihren DB-Cluster finden Sie den Cluster-Endpunkt, den Sie in Ihrer MySQL-Verbindungszeichenfolge verwenden können. Der Endpunkt besteht aus dem Domänenamen und dem Port für Ihr DB-Cluster. Wenn beispielsweise ein Endpunktwert `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306` lautet, sollten Sie die folgenden Werte in einer MySQL-Verbindungszeichenfolge angeben:

- Geben Sie als Host oder Host-Namen, an `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- Geben Sie für Port `3306` oder den Port-Wert an, den Sie beim Erstellen des DB-Clusters verwendet haben

Der Cluster-Endpunkt verbindet Sie mit der primären Instance für das DB-Cluster. Sie können sowohl Lese- als auch Schreiboperationen mithilfe des Cluster-Endpunkts durchführen. Ihr DB-Cluster kann darüber hinaus bis zu 15 Aurora-Replicas besitzen, die einen schreibgeschützten Zugriff auf die Daten im DB-Cluster unterstützen. Die primäre Instance und jede Aurora-Replica haben jeweils einen eindeutigen Endpunkt, der unabhängig vom Cluster-Endpunkt ist, und Ihnen ermöglicht sich mit einer bestimmten DB-Instance im Cluster direkt zu verbinden. Der Cluster-Endpunkt verweist immer auf die primäre Instance. Falls die primäre Instance ausfällt und ersetzt wird, verweist der Cluster-Endpunkt auf die neue primäre Instance.

Um den Cluster-Endpunkt (Writer-Endpunkt) anzuzeigen, wählen Sie Databases (Datenbanken) in der Amazon-RDS-Konsole und anschließend den Namen des DB-Clusters aus, um die Details des DB-Clusters anzuzeigen.



Themen

- Hilfsprogramme für die Herstellung von Verbindungen für Aurora MySQL
- Herstellen einer Verbindung mit Aurora MySQL über das MySQL-Dienstprogramm
- Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-JDBC-Treibers
- Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-Python-Treibers
- Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-ODBC-Treibers für MySQL
- Herstellen einer Verbindung mit Amazon MySQL mithilfe des AWS-Advanced-NodeJS-Wrappers
- Herstellen einer Verbindung mit Aurora MySQL mithilfe von SSL

Hilfsprogramme für die Herstellung von Verbindungen für Aurora MySQL

Unter anderem können Sie dazu die folgenden Verbindungsdienstprogramme verwenden:

- Befehlszeile – Sie können mittels Tools wie dem MySQL-Befehlszeilen-Hilfsprogramm Verbindungen mit einem Amazon-Aurora-DB-Cluster herstellen: Weitere Informationen zur Verwendung des MySQL-Hilfsprogramms finden Sie unter mysql – The MySQL Command Line Tool in der MySQL-Dokumentation.
- GUI – Sie können das Hilfsprogramm MySQL Workbench verwenden, um eine Verbindung über eine Benutzeroberfläche herzustellen. Weitere Informationen finden Sie auf der Seite Download MySQL Workbench.
- AWS-Treiber:
  - Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-JDBC-Treibers
  - Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-Python-Treibers
  - Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-ODBC-Treibers für MySQL
  - Herstellen einer Verbindung mit Amazon MySQL mithilfe des AWS-Advanced-NodeJS-Wrappers

Herstellen einer Verbindung mit Aurora MySQL über das MySQL-Dienstprogramm

Gehen Sie wie folgt vor: Es wird davon ausgegangen, dass Sie Ihren DB-Cluster in einem privaten Subnetz der VPC konfiguriert haben. Sie stellen eine Verbindung mit einer Amazon-EC2-Instance her, die Sie gemäß den Tutorials in Tutorial: Erstellen eines Webservers und einer eines Amazon Aurora-DB-Clusters konfiguriert haben.

> **Note**
>
> Für dieses Verfahren ist die Installation des Webservers im Tutorial nicht nötig, erfordert jedoch die Installation von MariaDB 10.5.

So stellen Sie eine Verbindung mit einem DB-Cluster mit dem MySQL-Dienstprogramm her

1. Melden Sie sich bei der EC2-Instance an, die Sie verwenden, um eine Verbindung mit Ihrem DB-Cluster herzustellen.

   Die Ausgabe sollte in etwa wie folgt aussehen:

```
Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-xxx.xxx ~]$
```

2. Geben Sie den folgenden Befehl in die Befehlszeile ein, um eine Verbindung mit der primären DB-Instance Ihres DB-Clusters herzustellen.

   Ersetzen Sie für den `-h`-Parameter den Endpunkt-DNS-Namen für Ihre primäre Instance. Ersetzen Sie den `-u`-Parameter durch die Benutzer-ID eines Datenbankbenutzerkontos.

   ```
   mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306
    -u database_user -p
   ```

   Beispiel:

   ```
   mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
   central-1.rds.amazonaws.com -P 3306 -u admin -p
   ```

3. Geben Sie das Passwort für den Datenbankbenutzer ein.

   Die Ausgabe sollte in etwa wie folgt aussehen:

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 1770
   Server version: 8.0.23 Source distribution

   Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

   Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

   MySQL [(none)]>
   ```

4. Geben Sie Ihre SQL-Befehle ein.

Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-JDBC-Treibers

Der JDBC-Treiber für Amazon Web Services (AWS) ist als fortschrittlicher JDBC-Wrapper konzipiert. Dieser Wrapper ergänzt und erweitert die Funktionalität eines vorhandenen JDBC-Treibers, um Anwendungen dabei zu unterstützen, die Funktionen geclusterter Datenbanken wie Aurora MySQL zu nutzen. Der Treiber ist Drop-In-kompatibel mit dem Community MySQL-Connector/J-Treiber und dem Community-MariaDB-Connector/J-Treiber.

Um den AWS-JDBC-Treiber zu installieren, hängen Sie die JAR-Datei des AWS-JDBC-Treibers an (befindet sich im Anwendungs-`CLASSPATH`) und behalten Sie die Verweise auf den jeweiligen Community-Treiber bei. Aktualisieren Sie das jeweilige Verbindungs-URL-Präfix wie folgt:

- `jdbc:mysql://` auf `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` auf `jdbc:aws-wrapper:mariadb://`

Weitere Informationen zum AWS-JDBC-Treiber und vollständige Anweisungen zur Verwendung finden Sie im GitHub-Repository für den AWS-JDBC-Treiber.

> **Note**
>
> Version 3.0.3 des MariaDB-Connector/J-Dienstprogramms bietet keine Unterstützung mehr für Aurora-DB-Cluster, daher empfehlen wir dringend, zum AWS-JDBC-Treiber zu wechseln.

Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-Python-Treibers

Der Python-Treiber für Amazon Web Services (AWS) ist als fortschrittlicher Python-Wrapper konzipiert. Dieser Wrapper ergänzt den Open-Source-Treiber Psycopg und erweitert dessen Funktionen. Der AWS-Python-Treiber unterstützt Python-Version 3.8 und höher. Sie können das `aws-advanced-python-wrapper`-Paket zusammen mit den `psycopg`-Open-Source-Paketen mit dem `pip`-Befehl installieren.

Weitere Informationen zum AWS-Python-Treiber und vollständige Anweisungen zur Verwendung finden Sie im GitHub-Repository für den AWS-Python-Treiber.

Herstellen einer Verbindung mit Aurora MySQL mithilfe des AWS-ODBC-Treibers für MySQL

Der AWS-ODBC-Treiber für MySQL ist ein Client-Treiber, der für die hohe Verfügbarkeit von Aurora MySQL konzipiert wurde. Der Treiber kann zusammen mit dem MySQL-Connector/ODBC-Treiber existieren und ist mit denselben Workflows kompatibel.

Weitere Informationen zum AWS-ODBC-Treiber für MySQL und vollständige Anweisungen zur Verwendung finden Sie im GitHub-Repository für den AWS-ODBC-Treiber.

Herstellen einer Verbindung mit Amazon MySQL mithilfe des AWS-Advanced-NodeJS-Wrappers

Der AWS-Advanced-NodeJS-Wrapper ergänzt und erweitert die Funktionalität eines vorhandenen NodeJS-Treibers. Er unterstützt Anwendungen dabei, die Funktionen geclusterter Datenbanken wie Aurora MySQL zu nutzen.

Weitere Informationen zum AWS-Advanced-NodeJS-Wrapper und vollständige Anweisungen zur Verwendung finden Sie im GitHub-Repository für den AWS-Advanced-NodeJS-Wrapper.

Herstellen einer Verbindung mit Aurora MySQL mithilfe von SSL

Sie können SSL-Verschlüsselung für Verbindungen mit einer Aurora MySQL DB-Instance verwenden. Weitere Informationen finden Sie unter TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

Verwenden Sie das MySQL-Hilfsprogramm wie im folgenden Vorgang beschrieben, um eine Verbindung per SSL herzustellen. Wenn Sie die IAM-Datenbank-Authentifizierung nutzen, müssen Sie eine SSL-Verbindung verwenden. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung.

> **Note**
>
> Damit Sie sich mit dem Cluster-Endpunkt unter Verwendung von SSL verbinden können, muss Ihr Client-Verbindungshilfsprogramm Subject Alternative Names (SAN) unterstützen. Wenn Ihr Client-Verbindungshilfsprogramm kein SAN unterstützt, können Sie sich nicht direkt mit den Instances in Ihrem Aurora-DB-Cluster verbinden. Weitere Informationen zu Aurora-Endpunkten finden Sie unter Amazon-Aurora-Endpunktverbindungen.

So stellen Sie eine SSL-Verbindung mit einem DB-Cluster mittels des MySQL-Hilfsprogramms her:

1. Laden Sie den öffentlichen Schlüssel für das Amazon-RDS-Signaturzertifikat herunter.

   Informationen zum Herunterladen von Zertifikaten finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.
2. Geben Sie den folgenden Befehl in die Befehlszeile ein, um sich mit der primären Instance eines DB-Clusters per SSL mithilfe des MySQL-Hilfsprogramms zu verbinden. Ersetzen Sie für den `-h`-Parameter den Endpunkt-DNS-Namen für Ihre primäre Instance. Ersetzen Sie den `-u`-Parameter durch die Benutzer-ID eines Datenbankbenutzerkontos. Ersetzen Sie entsprechend den SSL-Zertifikatsnamen für den Parameter `--ssl-ca`. Geben Sie das Hauptbenutzerpasswort ein, falls Sie aufgefordert werden.

   ```
   mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u admin_user -p --ssl-ca=[full path]global-bundle.pem --ssl-verify-server-cert
   ```

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 8.0.26-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Allgemeine Anweisungen zum Konstruieren von RDS für MySQL-Verbindungszeichenfolgen und zum Auffinden des öffentlichen Schlüssels für SSL-Verbindungen finden Sie unter Verbinden mit einer DB-Instance auf einer MySQL-Datenbank-Engine.

Herstellen einer Verbindung mit einem Amazon-Aurora-PostgreSQL-DB-Cluster

Sie können sich mit einer DB-Instance in Ihrem Amazon Aurora PostgreSQL-DB-Cluster mit denselben Tools verbinden, die Sie für die Verbindung mit einer PostgreSQL-Datenbank verwendet haben. Hierbei können Sie den gleichen öffentlichen Schlüssel für Secure Sockets Layer (SSL)-Verbindungen nutzen. Sie können die Endpunkt- und Portinformationen der primären Instance oder der Aurora-Replicas in Ihrem Aurora-PostgreSQL-DB-Cluster in der Verbindungszeichenfolge aller Skripts, Hilfsprogramme oder Anwendungen verwenden, die Verbindungen mit einer PostgreSQL-

DB-Instance herstellen. Geben Sie in der Verbindungszeichenfolge die DNS-Adresse aus der primären Instance oder dem Endpunkt der Aurora-Replica als Host-Parameter an. Geben Sie die Portnummer aus dem Endpunkt als Portparameter an.

Sobald Sie eine Verbindung zu einer DB-Instance in Ihrem Amazon-Aurora-PostgreSQL-DB-Cluster hergestellt haben, können Sie einen beliebigen SQL-Befehl ausführen, der mit PostgreSQL kompatibel ist.

In der Detailansicht Ihres Aurora-PostgreSQL-DB-Clusters finden Sie den Cluster-Endpunktnamen, den Status, den Typ und die Portnummer. Sie verwenden den Endpunkt und die Portnummer in Ihrer PostgreSQL-Verbindungszeichenfolge. Wenn beispielsweise der Endpunktwert `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com` lautet, geben Sie die folgenden Werte in der PostgreSQL-Verbindungszeichenfolge an:

- Geben Sie als Host oder Host-Namen, an `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- Geben Sie für Port `5432` oder den Port-Wert an, den Sie beim Erstellen des DB-Clusters verwendet haben

Der Cluster-Endpunkt verbindet Sie mit der primären Instance für das DB-Cluster. Sie können sowohl Lese- als auch Schreiboperationen mithilfe des Cluster-Endpunkts durchführen. Ihr DB-Cluster kann darüber hinaus bis zu 15 Aurora-Replicas besitzen, die einen schreibgeschützten Zugriff auf die Daten im DB-Cluster unterstützen. Jede DB-Instance im Aurora-Cluster (d. h. die primäre Instance und alle Aurora-Replicas) besitzt einen eindeutigen Endpunkt, der vom Cluster-Endpunkt unabhängig ist. Über diesen eindeutigen Endpunkt können Sie eine direkte Verbindung zu einer bestimmten DB-Instance im Cluster herstellen. Der Cluster-Endpunkt verweist immer auf die primäre Instance. Falls die primäre Instance ausfällt und ersetzt wird, verweist der Cluster-Endpunkt auf die neue primäre Instance.

Um den Cluster-Endpunkt (Writer-Endpunkt) anzuzeigen, wählen Sie Databases (Datenbanken) in der Amazon-RDS-Konsole und anschließend den Namen des DB-Clusters aus, um die Details des DB-Clusters anzuzeigen.
Aurora-PostgreSQL-Writer-Endpunkt.

Hilfsprogramme für die Herstellung von Verbindungen für Aurora PostgreSQL

Unter anderen können Sie dazu die folgenden Verbindungsdienstprogramme verwenden:

- Befehlszeile: Sie können mittels Tools wie dem interaktiven PostgreSQL-Terminal psql Verbindungen mit DB-Clustern von Aurora PostgreSQL herstellen. Weitere Informationen über die Verwendung des interaktiven PostgreSQL-Terminals finden Sie unter psql in der PostgreSQL-Dokumentation.
- GUI: Sie können das Dienstprogramm pgAdmin verwenden, um über eine Benutzeroberfläche Verbindungen mit DB-Clustern von Aurora PostgreSQL herzustellen. Weitere Informationen finden Sie auf der Seite Download der pgAdmin-Website.
- AWS-Treiber:
  - Herstellen einer Verbindung mit Aurora PostgreSQL mithilfe des AWS-JDBC-Treibers
  - Herstellen einer Verbindung mit Aurora PostgreSQL mithilfe des AWS-Python-Treibers
  - Herstellen einer Verbindung mit Amazon PostgreSQL mithilfe des AWS-Advanced-NodeJS-Wrappers

Herstellen einer Verbindung mit Aurora PostgreSQL mithilfe des AWS-JDBC-Treibers

Der JDBC-Treiber für Amazon Web Services (AWS) ist als fortschrittlicher JDBC-Wrapper konzipiert. Dieser Wrapper ergänzt und erweitert die Funktionalität eines vorhandenen JDBC-Treibers, um Anwendungen dabei zu unterstützen, die Funktionen geclusterter Datenbanken wie Aurora PostgreSQL zu nutzen. Der Treiber ist Drop-In-kompatibel mit dem Community-pgJDBC-Treiber.

Um den AWS-JDBC-Treiber zu installieren, hängen Sie die JAR-Datei des AWS-JDBC-Treibers an (befindet sich im Anwendungs-`CLASSPATH`) und behalten Sie die Verweise auf den pgJDBC-Community-Treiber bei. Aktualisieren Sie das Präfix der Verbindungs-URL von `jdbc:postgresql://` auf `jdbc:aws-wrapper:postgresql://`.

Weitere Informationen zum AWS-JDBC-Treiber und vollständige Anweisungen zur Verwendung finden Sie im GitHub-Repository für den AWS-JDBC-Treiber.

Herstellen einer Verbindung mit Aurora PostgreSQL mithilfe des AWS-Python-Treibers

Der Python-Treiber für Amazon Web Services (AWS) ist als fortschrittlicher Python-Wrapper konzipiert. Dieser Wrapper ergänzt den Open-Source-Treiber Psycopg und erweitert dessen Funktionen. Der AWS-Python-Treiber unterstützt Python-Version 3.8 und höher. Sie können das `aws-advanced-python-wrapper`-Paket zusammen mit den `psycopg`-Open-Source-Paketen mit dem `pip`-Befehl installieren.

Weitere Informationen zum AWS-Python-Treiber und vollständige Anweisungen zur Verwendung finden Sie im GitHub-Repository für den AWS-Python-Treiber.

Herstellen einer Verbindung mit Amazon PostgreSQL mithilfe des AWS-Advanced-NodeJS-Wrappers

Der AWS-Advanced-NodeJS-Wrapper ergänzt und erweitert die Funktionalität eines vorhandenen NodeJS-Treibers. Er unterstützt Anwendungen dabei, die Funktionen geclusterter Datenbanken wie Aurora PostgreSQL zu nutzen.

Weitere Informationen zum AWS-Advanced-NodeJS-Wrapper und vollständige Anweisungen zur Verwendung finden Sie im GitHub-Repository für den AWS-Advanced-NodeJS-Wrapper.

Behebung von Aurora-Verbindungsfehlern

Häufige Ursachen für Verbindungsfehler mit einem neuen Aurora-DB-Cluster umfassen Folgende:

- Die Sicherheitsgruppe in der VPC erlaubt keinen Zugriff – Ihre VPC muss Verbindungen von Ihrem Gerät oder von einer Amazon-EC2-Instance durch ordnungsgemäße Konfiguration der Sicherheitsgruppe in der VPC zulassen. Um dies zu beheben, ändern Sie die Inbound-Regeln der Sicherheitsgruppe Ihrer VPC, um Verbindungen zuzulassen. Ein Beispiel finden Sie unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).
- Port durch Firewallregeln blockiert – Überprüfen Sie den Wert des für Ihren Aurora-DB-Cluster konfigurierten Ports. Wenn eine Firewallregel diesen Port blockiert, können Sie die Instance mithilfe eines anderen Ports neu erstellen.
- Unvollständige oder falsche IAM Konfiguration – Wenn Sie Ihre Aurora-DB-Instance für die Verwendung von IAM–basierter Authentifizierung erstellt haben, stellen Sie sicher, dass sie ordnungsgemäß konfiguriert ist. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung.

Weitere Informationen zur Fehlerbehebung von Aurora DB-Verbindungsproblemen finden Sie unter Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden.

Parametergruppen für Amazon Aurora

Database parameters (Datenbankparameter) – geben Sie an, wie die Datenbank konfiguriert wird. Datenbankparameter können z. B. die Menge der Ressourcen angeben, die einer Datenbank zugewiesen werden sollen, wie etwa den Speicher.

Sie verwalten Ihre Datenbankkonfiguration, indem Sie Ihre DB-Instances und Aurora-DB-Cluster mit Parametergruppen zuordnen. Aurora definiert Parametergruppen mit Standardeinstellungen. Sie können auch eigene Parametergruppen mit angepassten Einstellungen definieren.

Themen

- Übersicht über Parametergruppen
- DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster
- DB-Parametergruppen für DB-Instances von Amazon Aurora
- Vergleichen von DB-Parametergruppen
- Festlegen von DB-Parametern

Übersicht über Parametergruppen

Eine DB-Cluster-Parametergruppe dient als Container für Engine-Konfigurationswerte, die für jede DB-Instance in einem Aurora-DB-Cluster gelten. Das freigegebene Aurora-Speichermodell erfordert, dass jede DB-Instance in einem Aurora-Cluster dieselbe Einstellung für Parameter wie verwende `innodb_file_per_table`. Daher sind Parameter, die sich auf das Layout des physischen Speichers auswirken, Teil der Cluster-Parametergruppe. Die DB-Cluster-Parametergruppe enthält auch Standardwerte für alle Parameter auf Instance-Ebene.

Eine DB-Parametergruppe dient als Container für Engine-Konfigurationswerte, die auf eine oder mehrere DB-Instances angewendet werden. DB-Parametergruppen gelten für DB-Instances in Amazon RDS und Aurora. Diese Konfigurationseinstellungen gelten für Eigenschaften, die je nach DB-Instances in einem Aurora-Cluster unterschiedlich sein können, z. B. Größe der Speicherpuffer.

Themen

- Standard- und benutzerdefinierte Parametergruppen
- Statische und dynamische DB-Cluster-Parameter
- Statische und dynamische DB-Instance-Parameter

- Zeichensatzparameter
- Unterstützte Parameter und Parameterwerte

Standard- und benutzerdefinierte Parametergruppen

Wenn Sie eine DB-Instance ohne Angabe einer DB-Parametergruppe erstellen, verwendet die DB-Instance eine Standard-DB-Parametergruppe. Beim Erstellen eines Aurora-DB-Clusters ohne Angabe einer DB-Cluster-Parametergruppe verwendet der DB-Cluster ebenso eine Standard-DB-Cluster-Parametergruppe. Jede Standard-Parametergruppe enthält Standardeinstellungen für die Datenbank-Engine und das Amazon-RDS-System, die auf der Engine, der Datenverarbeitungsklasse und dem zugeordneten Speicher der Instance basieren.

Sie können die Parametereinstellungen für eine Standard-Parametergruppe nicht ändern. Stattdessen können Sie Folgendes tun:

1. Neue Parametergruppe erstellen.
2. Ändern Sie die Einstellungen Ihrer gewünschten Parameter. In einer Parametergruppe können nicht alle DB-Engine-Parameter geändert werden.
3. Ändern Sie dann die DB-Instance oder den DB-Cluster so, sodass die neue Parametergruppe zugewiesen wird.

   Informationen zum Ändern eines DB-Clusters oder einer DB-Instance finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

   > **Note**
   >
   > Wenn Sie Ihre DB-Instance so geändert haben, dass sie eine benutzerdefinierte Parametergruppe verwendet, und Sie die DB-Instance starten, startet RDS die DB-Instance im Rahmen des Startvorgangs automatisch neu. Für Multi-AZ-Instances von RDS für SQL Server mit aktivierter Always-On- oder Spiegelungsoption wird ein Failover erwartet, wenn die Instance nach dem Startvorgang neu gestartet wird.

RDS wendet die geänderten statischen und dynamischen Parameter in einer neu zugeordneten Parametergruppe erst an, nachdem die DB-Instance neu gestartet wurde. Wenn Sie jedoch dynamische Parameter in der DB-Parametergruppe ändern, nachdem Sie sie der DB-Instance zugeordnet haben, werden diese Änderungen sofort ohne Neustart angewendet. Weitere

Informationen zum Ändern der DB-Parametergruppe finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Wenn Sie Parameter innerhalb einer DB-Parametergruppe aktualisieren, gelten die Änderungen für alle DB-Instances, die dieser Parametergruppe zugeordnet sind. Wenn Sie Parameter innerhalb einer Parametergruppe eines Aurora-DB-Clusters aktualisieren, gelten die Änderungen ebenso für alle Aurora-DB-Cluster, die dieser DB-Cluster-Parametergruppe zugeordnet sind.

Wenn Sie keine Parametergruppe von Grund auf neu erstellen möchten, können Sie eine vorhandene Parametergruppe mit dem AWS CLI-Befehl copy-db-parameter-group oder dem Befehl copy-db-cluster-parameter-group kopieren. Das Kopieren einer Parametergruppe kann sich in einigen Fällen als nützlich erweisen. Wenn Sie beispielsweise die am häufigsten verwendeten benutzerdefinierten Parameter und Werte einer vorhandenen -Parametergruppe in eine neue -Parametergruppe aufnehmen möchten.

Statische und dynamische DB-Cluster-Parameter

Die DB-Cluster-Parameter sind entweder statisch oder dynamisch. Sie unterscheiden sich in den folgenden Punkten:

- Wenn Sie einen statischen Parameter ändern und die DB-Cluster-Parametergruppe speichern, wird die Änderung des Parameters nach einem manuellen Neustart der DB-Instances in jedem zugeordneten DB-Cluster wirksam. Wenn Sie die AWS-Managementkonsole verwenden, um DB-Cluster-Parameterwerte zu ändern, wird immer `pending-reboot` als `ApplyMethod` genutzt.
- Wenn Sie einen dynamischen Parameter ändern, wird die Parameteränderung standardmäßig sofort wirksam, ohne dass ein Neustart erforderlich ist. Wenn Sie die Konsole verwenden, wird immer `immediate` als `ApplyMethod` verwendet. Wenn Sie die Parameteränderung aufschieben möchten, bis die DB-Instances in einem zugeordneten DB-Cluster neu gestartet wurden, verwenden Sie die AWS CLI oder RDS-API. Legen Sie die `ApplyMethod` für die Parameteränderung auf `pending-reboot` fest.

Weitere Informationen zur Verwendung der AWS CLI, um einen Parameterwert zu ändern, finden Sie unter modify-db-cluster-parameter-group. Weitere Informationen zur Verwendung der RDS-API, um einen Parameterwert zu ändern, finden Sie unter ModifyDBClusterParameterGroup.

Wenn Sie die DB-Cluster-Parametergruppe geändert haben, die einem DB-Cluster zugeordnet ist, starten Sie die DB-Instances im DB-Cluster neu. Durch den Neustart werden die Änderungen auf alle DB-Instances im DB-Cluster angewendet. Um zu ermitteln, ob die DB-Instances eines DB-Clusters

zum Übernehmen von Änderungen neu gestartet werden muss, führen Sie den folgenden AWS CLI-Befehl aus.

```
aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier
```

Überprüfen Sie den Wert `DBClusterParameterGroupStatus` für die primäre DB-Instance in der Ausgabe. Wenn der Wert `pending-reboot` lautet, starten Sie die DB-Instances des DB-Clusters neu.

Statische und dynamische DB-Instance-Parameter

Die DB-Instance-Parameter sind entweder statisch oder dynamisch. Sie weisen folgende Unterschiede auf:

- Wenn Sie einen statischen Parameter ändern und eine DB-Parametergruppe speichern, wird die Änderung des Parameters nach einem manuellen Neustart der zugeordneten DB-Instances angewendet. Bei statischen Parametern verwendet die Konsole immer `pending-reboot` als `ApplyMethod`.
- Wenn Sie einen dynamischen Parameter ändern, wird die Parameteränderung standardmäßig sofort wirksam, ohne dass ein Neustart erforderlich ist. Wenn Sie die AWS-Managementkonsole verwenden, um DB-Instance-Parameterwerte zu ändern, wird immer `immediate` für die `ApplyMethod` für dynamische Parameter genutzt. Wenn Sie die Parameteränderung aufschieben möchten, bis eine zugeordnete DB-Instance neu gestartet wird, verwenden Sie die AWS CLI oder RDS-API. Legen Sie die `ApplyMethod` für die Parameteränderung auf `pending-reboot` fest.

Weitere Informationen zur Verwendung der AWS CLI, um einen Parameterwert zu ändern, finden Sie unter modify-db-parameter-group. Weitere Informationen zur Verwendung der RDS-API, um einen Parameterwert zu ändern, finden Sie unter ModifyDBParameterGroup.

Wenn auf der DB-Instance noch nicht die neuesten Änderungen der zugeordneten DB-Parametergruppe übernommen wurden, gibt die Konsole für die DB-Parametergruppe den Status pending-reboot an. Dieser Status führt während des nächsten Wartungsfensters nicht zu einem automatischen Neustart. Damit die neuesten Parameteränderungen für diese DB-Instance übernommen werden, starten Sie die DB-Instance manuell neu.

Zeichensatzparameter

Bevor Sie einen Cluster erstellen, legen Sie alle Parameter für den Zeichensatz oder die Datenbanksortierung in Ihrer Parametergruppe fest. Führen Sie diesen Schritt auch aus, bevor Sie

darin eine Datenbank erstellen. Dadurch stellen Sie sicher, dass die Standard-Datenbank und neue Datenbanken den Zeichensatz und die Sortierungswerte verwenden, die Sie angeben. Wenn Sie einen Zeichensatz oder eine Sammlung von Parametern ändern, werden die Parameteränderungen nicht in Ihren bestehenden Datenbanken angewandt.

Bei einigen DB-Engines können Sie den Zeichensatz oder die Sortierreihenfolge für eine bestehende Datenbank ändern, indem Sie z. B. den Befehl `ALTER DATABASE` verwenden:

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

Weitere Informationen zum Ändern des Zeichensatzes oder der Sortierreihenfolge für eine Datenbank finden Sie in der Dokumentation zu Ihrer DB-Engine.

Unterstützte Parameter und Parameterwerte

Wenn Sie die unterstützten Parameter für Ihre DB-Engine ermitteln möchten, zeigen Sie die Parameter in der DB-Parametergruppe und in der DB-Cluster-Parametergruppe an, die vom DB-Cluster oder von der DB-Instance verwendet werden. Weitere Informationen erhalten Sie unter Anzeigen von Parameterwerten für eine DB-Parametergruppe in Amazon Aurora und Anzeigen von Parameterwerten für eine DB-Cluster Parametergruppe in Amazon Aurora.

In vielen Fällen können Sie Ganzzahl- und Boolesche Parameter mithilfe von Ausdrücken, Formeln und Funktionen angeben. Funktionen können einen mathematischen "log"-Ausdruck enthalten. Nicht alle Parameter unterstützen jedoch Ausdrücke, Formeln und Funktionen für Parameterwerte. Weitere Informationen finden Sie unter Festlegen von DB-Parametern.

Für eine globale Aurora-Datenbank können Sie verschiedene Konfigurationseinstellungen für die einzelnen Aurora-Cluster festlegen. Wenn Sie einen sekundären Cluster zum primären Cluster hochstufen, müssen die Einstellungen in ausreichendem Umfang ähnlich sein, um ein einheitliches Verhalten zu erzeugen. Verwenden Sie beispielsweise die gleichen Einstellungen für Zeitzonen und Zeichensätze für alle Cluster einer globalen Aurora-Datenbank.

Werden die Parameter in einer Parametergruppe unpassend eingestellt, kann dies unbeabsichtigte unerwünschte Auswirkungen haben, einschließlich verminderter Leistung und Systeminstabilität. Gehen Sie immer mit Bedacht vor, wenn Sie Datenbankparameter ändern, und sichern Sie Ihre Daten, bevor Sie eine Parametergruppe ändern. Führen Sie Änderungen an einer Parametergruppe immer zuerst auf einer Test-DB-Instance oder einem DB-Cluster aus, bevor Sie diese Änderungen für eine Produktions-DB-Instance oder einen -DB-Cluster übernehmen.

DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster

Amazon-Aurora-DB-Cluster verwenden DB-Cluster-Parametergruppen. Die folgenden Abschnitte beschreiben das Konfigurieren und Verwalten von DB-Cluster-Parametergruppen.

Themen

- Amazon Aurora-DB-Cluster und DB-Instance-Parameter
- Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora
- Zuordnen einer DB-Cluster-Parametergruppe zu einem DB-Cluster
- Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora
- Zurücksetzen von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora
- Kopieren einer DB-Cluster-Parametergruppe in Amazon Aurora
- Auflisten von DB-Cluster-Parametergruppen in Amazon Aurora
- Anzeigen von Parameterwerten für eine DB-Cluster Parametergruppe in Amazon Aurora
- Löschen einer DB-Cluster-Parametergruppe in Amazon Aurora

Amazon Aurora-DB-Cluster und DB-Instance-Parameter

Aurora verwendet ein Zwei-Level-System von Konfigurationseinstellungen:

- Parameter in einer DB-Cluster-Parametergruppe werden auf jede DB-Instance in einem DB-Cluster angewandt. Ihre Daten werden im freigegebenen Aurora-Speicher-Subsystem gespeichert. Daher müssen alle Parameter, die sich auf das physische Layout von Tabellendaten beziehen, bei allen DB-Instances in einem Aurora-Cluster gleich sein. Da Aurora-DB-Instances durch Replikation verbunden sind, müssen alle Parameter für Replikationseinstellungen innerhalb eines Aurora-Clusters ebenfalls identisch sein.
- Parameter in einer DB-Parametergruppe werden auf eine einzelne DB-Instance in einem Aurora-DB-Cluster angewandt. Diese Parameter beziehen sich auf Aspekte wie die Speichernutzung, die Sie über die DB-Instances in demselben Aurora-Cluster hinweg unterschiedlich einstellen können. Ein Cluster enthält beispielsweise oft DB-Instances mit verschiedenen AWS-Instance-Klassen.

Jeder Aurora-Cluster ist einer DB-Cluster-Parametergruppe zugeordnet. Diese Parametergruppe weist jedem Konfigurationswert Standardwerte für die entsprechende DB-Engine zu. Die Cluster-Parametergruppe enthält auch Standardwerte für Parameter sowohl auf Cluster- als auch auf

Instance-Ebene. Jede DB-Instance innerhalb eines bereitgestellten oder eines Aurora Serverless v2-Clusters erbt die Einstellungen von dieser DB-Cluster-Parametergruppe.

Jede DB-Instance wird außerdem einer DB-Parametergruppe zugeordnet. Die Werte in der DB-Parametergruppe können Standardwerte aus der Cluster-Parametergruppe außer Kraft setzen. Wenn beispielsweise bei einer Instance in einem Cluster Probleme auftraten, können Sie dieser Instance eine benutzerdefinierte DB-Parametergruppe zuweisen. Die benutzerdefinierte Parametergruppe hat möglicherweise spezifische Einstellungen für Parameter im Hinblick auf Debuggen oder Leistungsoptimierung.

Wenn Sie einen Cluster oder eine neue DB-Instance erstellen, weist Aurora Parametergruppen basierend auf der angegebenen Datenbank-Engine und -Version zu. Sie können stattdessen eine benutzerdefinierte Parametergruppe angeben. Sie erstellen diese Parametergruppen selbst und können die Parameterwerte bearbeiten. Diese benutzerdefinierten Parametergruppen geben Sie bei der Erstellung an. Sie können auch einen DB-Cluster oder eine Instance später ändern, um eine benutzerdefinierte Parametergruppe zu verwenden.

Bei bereitgestellten und Aurora Serverless v2-Instances überschreiben alle Konfigurationswerte, die Sie in der DB-Cluster-Parametergruppe ändern, die Standardwerte in der DB-Parametergruppe. Wenn Sie die entsprechenden Werte in der DB-Parametergruppe bearbeiten, überschreiben diese Werte die Einstellungen aus der DB-Cluster-Parametergruppe.

Alle von Ihnen geänderten DB-Parameter-Einstellungen haben Vorrang vor den Werten der DB-Cluster-Parametergruppe. Dies gilt selbst dann, wenn Sie die Konfigurationsparameter wieder auf deren Standardwerte zurücksetzen. Mithilfe des AWS CLI-Befehls describe-db-parameters oder der RDS-API-Operation DescribeDBParameters können Sie sehen, welche Parameter überschrieben werden. Das Feld `Source` enthält den Wert `user`, wenn Sie diesen Parameter geändert haben. Wenn Sie einen oder mehrere Parameter zurückzusetzen möchten, sodass dem Wert der DB-Cluster-Parametergruppe Vorrang gegeben wird, verwenden Sie den AWS CLI-Befehl reset-db-parameter-group oder die RDS-API-Operation ResetDBParameterGroup.

Die Parameter des DB-Clusters und der DB-Instance, die Ihnen in Aurora zur Verfügung stehen, variieren je nach Kompatibilität der Datenbank-Engine.

Datenbank-Engine	Parameter
Aurora MySQL	Siehe Aurora MySQL Konfigurationsparameter.

Datenbank-Engine	Parameter
	Für Aurora Serverless-Cluster finden Sie weitere Einzelheiten unter Arbeiten mit Parametergruppen für Aurora Serverless v2 und Parametergruppen für Aurora Serverless v1.
Aurora PostgreSQL	Siehe Amazon-Aurora-PostgreSQL-Parameter. Für Aurora Serverless-Cluster finden Sie weitere Einzelheiten unter Arbeiten mit Parametergruppen für Aurora Serverless v2 und Parametergruppen für Aurora Serverless v1.

> **Note**
>
> Aurora Serverless v1-Cluster verfügen nur über zugehörige DB-Cluster-Parametergruppen und nicht über DB-Parametergruppen. Für Aurora Serverless v2-Cluster nehmen Sie alle Ihre Änderungen an benutzerdefinierten Parametern in der DB-Cluster-Parametergruppe vor. Aurora Serverless v2 verwendet sowohl DB-Cluster-Parametergruppen als auch DB-Parametergruppen. Mit Aurora Serverless v2 können Sie fast alle Konfigurationsparameter ändern. Aurora Serverless v2 überschreibt die Einstellungen einiger kapazitätsbezogener Konfigurationsparameter, sodass Ihre Workload nicht unterbrochen wird, wenn Aurora Serverless v2-Instances herunterskalieren.
> Weitere Informationen zu Aurora Serverless-Konfigurationseinstellungen und dazu, welche Einstellungen Sie ändern können, finden Sie unter Arbeiten mit Parametergruppen für Aurora Serverless v2 und Parametergruppen für Aurora Serverless v1.

Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora

Sie können eine neue DB-Cluster-Parametergruppe mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API erstellen.

Nachdem Sie eine DB-Cluster-Parametergruppe erstellt haben, sollten Sie mindestens fünf Minuten warten, bevor Sie einen DB-Cluster erstellen, der diese DB-Cluster-Parametergruppe verwendet. Auf diese Weise kann Amazon RDS die Parametergruppe vollständig erstellen, bevor sie vom neuen DB-Cluster verwendet wird. Mithilfe der Option Parameter Groups (Parametergruppen) in der Amazon-RDS-Konsole oder mithilfe des Befehls describe-db-cluster-parameters können Sie überprüfen, ob Ihre DB-Cluster-Parametergruppe erstellt wurde.

Die folgenden Einschränkungen gelten für den Namen der DB-Cluster-Parametergruppe:

- Der Name muss zwischen 1 und 255 Buchstaben, Zahlen oder Bindestriche enthalten.

  Standardnamen für Parametergruppen können einen Punkt enthalten, z. B. `default.aurora-mysql5.7`. Namen von benutzerdefinierten Parametergruppen dürfen jedoch keinen Punkt enthalten.
- Das erste Zeichen muss ein Buchstabe sein.
- Der Name darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.

Konsole

So erstellen Sie eine DB-Cluster-Parametergruppe

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie Create parameter group (Parametergruppe erstellen).
4. Geben Sie im Feld Name der Parametergruppe den Namen der neuen DB-Cluster-Parametergruppe ein.
5. Geben Sie im Feld Beschreibung eine Beschreibung für die neue DB-Cluster-Parametergruppe ein.
6. Wählen Sie als Engine-Typ Ihre Datenbank-Engine aus.
7. Wählen Sie als Parametergruppenfamilie eine DB-Parametergruppenfamilie aus.
8. Wählen Sie Erstellen aus.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `create-db-cluster-parameter-group` zum Erstellen einer DB-Cluster-Parametergruppe.

Im folgenden Beispiel wird eine DB-Cluster-Parametergruppe mit dem Namen mydbclusterparametergroup für Aurora MySQL Version 5.7 und der Beschreibung „My new cluster parameter group“ erstellt.

Nutzen Sie die folgenden erforderlichen Parameter:

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

Um alle verfügbaren Parametergruppenfamilien aufzulisten, führen Sie den folgenden Befehl aus:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> Die Ausgabe enthält Duplikate.

Example

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new cluster parameter group"
```

Für Windows:

```
aws rds create-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

Die Ausgabe dieses Befehls sieht etwa so aus:

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
    }
```

```
}
```

RDS-API

Um eine DB-Cluster-Parametergruppe zu erstellen, verwenden Sie die RDS-API-Aktion `CreateDBClusterParameterGroup`.

Nutzen Sie die folgenden erforderlichen Parameter:

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Zuordnen einer DB-Cluster-Parametergruppe zu einem DB-Cluster

Sie können Ihre eigenen DB-Cluster-Parametergruppen mit benutzerdefinierten Einstellungen erstellen. Sie können eine DB-Cluster-Parametergruppe mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API zu einem DB-Cluster zuordnen. Das können Sie tun, wenn Sie einen DB-Cluster erstellen oder ändern.

Informationen über das Erstellen einer DB-Cluster-Parametergruppe finden Sie unter Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora. Informationen zum Erstellen eines DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters. Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

> **Note**
>
> Wenn Sie für Aurora PostgreSQL 15.2, 14.7, 13.10, 12.14 und alle 11-Versionen die DB-Cluster-Parametergruppe ändern, die einem DB-Cluster zugeordnet ist, starten Sie jede Replikat-Instance neu, um die Änderungen anzuwenden.
> Um zu ermitteln, ob die primäre DB-Instance eines DB-Clusters zum Übernehmen von Änderungen neu gestartet werden muss, führen Sie den folgenden AWS CLI-Befehl aus: `aws rds describe-db-clusters --db-cluster-identifier` *`db_cluster_identifier`*
> Überprüfen Sie den Wert `DBClusterParameterGroupStatus` für die primäre DB-Instance in der Ausgabe. Wenn der Wert `pending-reboot` lautet, starten Sie die primäre DB-Instance des DB-Clusters neu.

Konsole

So ordnen Sie eine DB-Cluster-Parametergruppe zu einem DB-Cluster zu

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den zu ändernden DB-Cluster aus.
3. Wählen Sie Ändern aus. Die Seite Modify DB cluster (DB-Cluster ändern) wird angezeigt.
4. Ändern Sie die Einstellung für DB-Cluster-Parametergruppen.
5. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.

   Die Änderung wird unabhängig von der Einstellung Einplanung von Änderungen sofort angewendet.
6. Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen. Wenn sie korrekt sind, wählen Sie Modify cluster (Cluster ändern) aus, um Ihre Änderungen zu speichern.

   Klicken Sie anderenfalls auf Zurück, um Ihre Änderungen zu bearbeiten, oder klicken Sie auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `modify-db-cluster` mit den folgenden Optionen, um eine DB-Cluster-Parametergruppe zuzuordnen:

- `--db-cluster-name`
- `--db-cluster-parameter-group-name`

Im folgenden Beispiel wird die `mydbclpg`-DB-Parametergruppe zum `mydbcluster`-DB-Cluster zugeordnet.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --db-cluster-parameter-group-name mydbclpg
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-parameter-group-name mydbclpg
```

RDS-API

Um eine DB-Cluster-Parametergruppe einem DB-Cluster zuzuordnen, verwenden Sie die RDS-API-Operation `ModifyDBCluster` mit den folgenden Parametern:

- `DBClusterIdentifier`
- `DBClusterParameterGroupName`

Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora

Sie können Parameterwerte in einer vom Kunden erstellten DB-Clusterparametergruppe ändern. Sie können die Parameterwerte in einer Standard-DB-Clusterparametergruppe nicht ändern. Änderungen an Parametern in einer benutzerdefinierten DB-Cluster-Parametergruppe gelten für alle DB-Cluster, die dieser DB-Cluster-Parametergruppe zugeordnet sind.

Konsole

So ändern Sie eine DB-Cluster-Parametergruppe

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste die zu ändernde Parametergruppe.
4. Wählen Sie für Parameter group actions (Parametergruppenaktionen) die Option Bearbeiten.
5. Ändern Sie die Werte der Parameter, die Sie ändern möchten. Sie können durch die Parameter scrollen, in dem Sie die Pfeiltasten oben rechts im Dialogfeld verwenden.

   Die Werte in einer Standardparametergruppe können Sie nicht ändern.
6. Wählen Sie Änderungen speichern aus.
7. Starten Sie die primäre DB-Instance (Writer) im Cluster neu, um die Änderungen darauf anzuwenden.
8. Starten Sie anschließend die Reader-DB-Instances neu, um die Änderungen anzuwenden.

Wenn Sie die DB-Instances nicht neu starten, kann ein Failover-Vorgang länger als normal dauern.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `modify-db-cluster-parameter-group` mit den folgenden erforderlichen Parametern, um eine DB-Cluster-Parametergruppe zu ändern:

- `--db-cluster-parameter-group-name`
- `--parameters`

Im folgenden Beispiel werden die Werte `server_audit_logging` und `server_audit_logs_upload` in der DB-Cluster-Parametergruppe mydbclusterparametergroup geändert.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

RDS-API

Um eine DB-Cluster-Parametergruppe zu ändern, verwenden Sie den RDS-API-Befehl `ModifyDBClusterParameterGroup` mit den folgenden erforderlichen Parametern:

- `DBClusterParameterGroupName`
- `Parameters`

Zurücksetzen von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora

Sie können Parameter in einer vom Kunden erstellten DB-Clusterparametergruppe auf ihre Standardwerte zurücksetzen. Änderungen an Parametern in einer benutzerdefinierten DB-Cluster-Parametergruppe gelten für alle DB-Cluster, die dieser DB-Cluster-Parametergruppe zugeordnet sind.

> **Note**
>
> In einer Standardparametergruppe des DB-Clusters werden die Parameter immer auf ihre Standardwerte eingestellt.

Konsole

So setzen Sie Parameter in einer DB-Cluster-Parametergruppe auf ihre Standardwerte zurück

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste die Parametergruppe aus.
4. Wählen Sie für Parameter group actions (Parametergruppenaktionen) die Option Bearbeiten.
5. Wählen Sie die Parameter aus, die Sie auf ihre Standardwerte zurücksetzen möchten. Sie können durch die Parameter scrollen, in dem Sie die Pfeiltasten oben rechts im Dialogfeld verwenden.

   Sie können die Werte in einer Standardparametergruppe nicht zurücksetzen.
6. Wählen Sie Reset (Zurücksetzen) und bestätigen Sie dann mit Reset parameters (Parameter zurücksetzen).

7. Starten Sie die primäre DB-Instance im DB-Cluster neu, um die Änderungen auf alle DB-Instances in dem DB-Cluster anzuwenden.

AWS CLI

Um Parameter in einer DB-Cluster-Parametergruppe auf ihre Standardwerte zurückzusetzen, verwenden Sie den AWS CLI`reset-db-cluster-parameter-group`-Befehl mit der folgenden erforderlichen Option: `--db-cluster-parameter-group-name`.

Um alle Parameter in der Parametergruppe des DB-Clusters zurückzusetzen, wählen Sie die Option `--reset-all-parameters`. Um bestimmte Parameter zurückzusetzen, geben Sie die Option `--parameters` an.

Im folgenden Beispiel werden alle Parameter in der DB-Parametergruppe namens mydbparametergroup auf ihre Standardwerte zurückgesetzt.

Example

Für Linux, macOS oder Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Für Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

Im folgenden Beispiel werden die Werte `server_audit_logging` und `server_audit_logs_upload` in der DB-Cluster-Parametergruppe mydbclusterparametergroup auf ihre Standardwerte zurückgesetzt.

Example

Für Linux, macOS oder Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
```

```
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
                 "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

Für Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

RDS-API

Um Parameter in einer DB-Cluster-Parametergruppe auf ihre Standardwerte zurückzusetzen, verwenden Sie den `ResetDBClusterParameterGroup`-RDS-API-Befehl mit dem folgenden erforderlichen Parameter: `DBClusterParameterGroupName`.

Um alle Parameter in der Parametergruppe des DB-Clusters zurückzusetzen, legen Sie den Parameter `ResetAllParameters` auf `true` fest. Um bestimmte Parameter zurückzusetzen, geben Sie den Parameter `Parameters` an.

Kopieren einer DB-Cluster-Parametergruppe in Amazon Aurora

Sie können die von Ihnen erstellten benutzerdefinierten DB-Cluster-Parametergruppen kopieren. Das Kopieren einer Parametergruppe ist eine praktische Lösung, wenn Sie bereits eine DB-Cluster-Parametergruppe erstellt haben und die am häufigsten verwendeten Parameter und Werte aus dieser Gruppe in eine neuen DB-Cluster-Parametergruppe übernehmen möchten. Sie können eine DB-Cluster-Parametergruppe mit dem AWS CLI-Befehl copy-db-cluster-parameter-group oder der RDS-API-Operation CopyDBClusterParameterGroup kopieren.

Nachdem Sie eine DB-Cluster-Parametergruppe kopiert haben, warten Sie mindestens fünf Minuten, bevor Sie einen DB-Cluster erstellen, der diese DB-Cluster-Parametergruppe verwendet. Auf diese Weise kann Amazon RDS die Parametergruppe vollständig kopieren, bevor sie vom neuen DB-Cluster verwendet wird. Mithilfe der Option Parameter Groups (Parametergruppen) in der Amazon-

RDS-Konsole oder mithilfe des Befehls describe-db-cluster-parameters können Sie überprüfen, ob Ihre DB-Cluster-Parametergruppe erstellt wurde.

> **Note**
>
> Standardparametergruppen können nicht kopiert werden. Sie können jedoch eine neue Parametergruppe erstellen, die auf einer Standardparametergruppe basiert.
> Sie können eine DB-Cluster-Parametergruppe nicht in ein anderes AWS-Konto oder eine andere AWS-Region kopieren.

Konsole

So kopieren Sie eine DB-Cluster-Parametergruppe

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste die zu kopierende benutzerdefinierte Parametergruppe.
4. Wählen Sie für Parameter group actions (Parametergruppenaktionen) die Option Kopieren.
5. Geben Sie unter New DB parameter group identifier (Neue DB-Parametergruppenkennung) einen Namen für die neue Parametergruppe ein.
6. Geben Sie unter Beschreibung eine Beschreibung für die neue Parametergruppe ein.
7. Wählen Sie die Option Copy aus.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `copy-db-cluster-parameter-group` mit den folgenden erforderlichen Parametern, um eine DB-Cluster-Parametergruppe zu kopieren:

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

Im folgenden Beispiel wird eine neue DB-Cluster-Parametergruppe mit dem Namen `mygroup2` als Kopie der DB-Cluster-Parametergruppe `mygroup1` erstellt.

Example

Für Linux, macOS oder Unix:

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

Für Windows:

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS-API

Um eine DB-Cluster-Parametergruppe zu kopieren, verwenden Sie die RDS-API-Operation `CopyDBClusterParameterGroup` mit den folgenden erforderlichen Parametern:

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

Auflisten von DB-Cluster-Parametergruppen in Amazon Aurora

Sie können die DB-Cluster-Parametergruppen, die Sie für Ihr AWS-Konto erstellt haben, in einer Liste anzeigen.

> **Note**
>
> Standardparametergruppen werden automatisch aus einer Standardparametervorlage generiert, wenn Sie ein DB-Cluster für eine bestimmte DB-Engine und -Version erstellen. Diese Standardparametergruppen enthalten bevorzugte Parametereinstellungen und können nicht geändert werden. Wenn Sie eine benutzerdefinierte Parametergruppe erstellen, können Sie Parametereinstellungen ändern.

Konsole

So zeigen Sie alle DB-Cluster-Parametergruppen für ein AWS-Konto in einer Liste an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.

   Die DB-Cluster-Parametergruppen erscheinen in der Liste mit DB cluster parameter group (DB-Cluster-Parametergruppe) als Type (Typ).

AWS CLI

Verwenden Sie den AWS CLI-Befehl `describe-db-cluster-parameter-groups`, um alle DB-Cluster-Parametergruppen für ein AWS-Konto auflisten zu lassen.

Example

Im folgenden Beispiel werden alle verfügbaren DB-Cluster-Parametergruppen für ein AWS-Konto aufgelistet.

```
aws rds describe-db-cluster-parameter-groups
```

Im folgenden Beispiel wird die Parametergruppe mydbclusterparametergroup beschrieben.

Für Linux, macOS oder Unix:

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Für Windows:

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
{
```

```
    "DBClusterParameterGroups": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

RDS-API

Verwenden Sie die RDS-API-Aktion `DescribeDBClusterParameterGroups`, um alle DB-Cluster-Parametergruppen für ein AWS-Konto in einer Liste anzuzeigen.

Anzeigen von Parameterwerten für eine DB-Cluster Parametergruppe in Amazon Aurora

Sie können alle Parameter in einer DB-Cluster-Parametergruppe mit ihren Werten in einer Liste anzeigen.

Konsole

So zeigen Sie die Parameterwerte für eine DB-Cluster-Parametergruppe an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.

   Die DB-Cluster-Parametergruppen erscheinen in der Liste mit DB cluster parameter group (DB-Cluster-Parametergruppe) als Type (Typ).
3. Wählen Sie den Namen der DB-Cluster-Parametergruppe, um deren Parameterliste anzuzeigen.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `describe-db-cluster-parameters` mit dem folgenden erforderlichen Parameter, um die Parameterwerte für eine DB-Cluster-Parametergruppe anzuzeigen.

- `--db-cluster-parameter-group-name`

Example

Das folgende Beispiel listet die Parameter und Parameterwerte für eine DB-Cluster-Parametergruppe namens mydbclusterparametergroup im JSON-Format auf.

Die Ausgabe des Befehls ähnelt der Folgenden:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```

```
{
    "Parameters": [
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "aurora_binlog_read_buffer_size",
            "ParameterValue": "5242880",
            "Description": "Read buffer size used by master dump thread when the switch
 aurora_binlog_use_large_read_buffer is ON.",
            "Source": "engine-default",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "8192-536870912",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },

...
```

RDS-API

Um die Parameterwerte für eine DB-Cluster-Parametergruppe anzuzeigen, verwenden Sie den RDS-API-Befehl `DescribeDBClusterParameters` mit dem folgenden erforderlichen Parameter.

- `DBClusterParameterGroupName`

In einigen Fällen werden die zulässigen Werte für einen Parameter nicht angezeigt. Es handelt sich dabei immer um Parameter, bei denen die Quelle die Standardeinstellung der Datenbank-Engine ist.

Um die Werte dieser Parameter anzuzeigen, können Sie die folgenden SQL-Anweisungen ausführen:

- MySQL:

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
mysql$ SHOW VARIABLES;
```

- PostgreSQL:

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

Löschen einer DB-Cluster-Parametergruppe in Amazon Aurora

Sie können eine DB-Cluster-Parametergruppe mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API löschen. Eine DB-Cluster-Parametergruppe kann nur gelöscht werden, wenn sie mit keinem DB-Cluster verknüpft ist.

Konsole

So löschen Sie Parametergruppen:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.

   Die verfügbaren Parametergruppen werden in einer Liste angezeigt.
3. Wählen Sie den Namen der zu löschenden DB-Cluster-Parametergruppen aus.
4. Wählen Sie Actions (Aktionen) und anschließend Delete (Löschen) aus.
5. Überprüfen Sie die Namen der Parametergruppen und wählen Sie dann Löschen.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `delete-db-cluster-parameter-group` mit den folgenden erforderlichen Parametern, um eine DB-Cluster-Parametergruppe zu löschen:

- `--db-parameter-group-name`

Example

Im folgenden Beispiel wird die DB-Cluster-Parametergruppe mydbparametergroup gelöscht.

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS-API

Verwenden Sie den Befehl `DeleteDBClusterParameterGroup` mit den folgenden erforderlichen Parametern, um eine DB-Cluster-Parametergruppe zu löschen.

- `DBParameterGroupName`

DB-Parametergruppen für DB-Instances von Amazon Aurora

DB-Instances verwenden DB-Parametergruppen. Die folgenden Abschnitte beschreiben das Konfigurieren und Verwalten von DB-Instance-Parametergruppen.

Themen

- Erstellen einer DB-Parametergruppe in Amazon Aurora
- Verknüpfen einer DB-Parametergruppe mit einer DB-Instance in Amazon Aurora
- Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora

- Zurücksetzen von Parametern in einer DB-Parametergruppe auf ihre Standardwerte in Amazon Aurora
- Kopieren einer DB-Parametergruppe in Amazon Aurora
- Auflisten von DB-Parametergruppen in Amazon Aurora
- Anzeigen von Parameterwerten für eine DB-Parametergruppe in Amazon Aurora
- Löschen einer DB-Parametergruppe in Amazon Aurora

Erstellen einer DB-Parametergruppe in Amazon Aurora

Sie können eine neue DB-Parametergruppe mithilfe der AWS-Managementkonsole AWS CLI, der oder der RDS-API erstellen.

Die folgenden Einschränkungen gelten für den Namen der DB-Parametergruppe:

- Der Name muss zwischen 1 und 255 Buchstaben, Zahlen oder Bindestriche enthalten.

  Standardnamen für Parametergruppen können einen Punkt enthalten, z. B. `default.mysql8.0`. Namen von benutzerdefinierten Parametergruppen dürfen jedoch keinen Punkt enthalten.
- Das erste Zeichen muss ein Buchstabe sein.
- Der Name darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.

Konsole

So erstellen Sie eine DB-Parametergruppe:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie Create parameter group (Parametergruppe erstellen).
4. Geben Sie für Parametergruppenname den Namen Ihrer neuen DB-Parametergruppe ein.
5. Geben Sie für Beschreibung eine Beschreibung für Ihre neue DB-Parametergruppe ein.
6. Wählen Sie als Engine-Typ Ihre DB-Engine aus.
7. Wählen Sie als Parametergruppenfamilie eine DB-Parametergruppenfamilie aus.

8. Wählen Sie für Typ, falls zutreffend, DB-Parametergruppe aus.
9. Wählen Sie Erstellen aus.

AWS CLI

Verwenden Sie den AWS CLI `create-db-parameter-group`Befehl, um eine DB-Parametergruppe zu erstellen. Im folgenden Beispiel wird eine DB-Parametergruppe mit dem Namen mydbparametergroup für MySQL Version 8.0 und der Beschreibung „My new parameter group“ erstellt.

Nutzen Sie die folgenden erforderlichen Parameter:

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

Um alle verfügbaren Parametergruppenfamilien aufzulisten, führen Sie den folgenden Befehl aus:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> Die Ausgabe enthält Duplikate.

Example

Für Linux, macOS oder Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new parameter group"
```

Für Windows:

```
aws rds create-db-parameter-group ^
```

```
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new parameter group"
```

Die Ausgabe dieses Befehls sieht etwa so aus:

```
DBPARAMETERGROUP  mydbparametergroup  aurora-mysql5.7  My new parameter group
```

RDS-API

Um eine DB-Parametergruppe zu erstellen, verwenden Sie die RDS-API-Operation `CreateDBParameterGroup`.

Nutzen Sie die folgenden erforderlichen Parameter:

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Verknüpfen einer DB-Parametergruppe mit einer DB-Instance in Amazon Aurora

Sie können Ihre eigenen DB-Parametergruppen mit benutzerdefinierten Einstellungen erstellen. Sie können einer DB-Instance mithilfe der AWS-Managementkonsole, der oder der RDS-API eine DB-Parametergruppe zuordnen. AWS CLI Das können Sie tun, wenn Sie eine DB-Instance erstellen oder ändern.

Informationen über das Erstellen einer Parametergruppe finden Sie unter Erstellen einer DB-Parametergruppe in Amazon Aurora. Informationen zum Ändern einer DB-Instance finden Sie unter Ändern einer DB-Instance in einem DB-Cluster.

> **Note**
>
> Wenn Sie eine neue DB-Parametergruppe einer DB-Instance zuordnen, werden die geänderten statischen und dynamischen Parameter erst nach Neustart der DB-Instance angewendet. Wenn Sie jedoch dynamische Parameter in der DB-Parametergruppe ändern, nachdem Sie sie der DB-Instance zugeordnet haben, werden diese Änderungen sofort ohne Neustart angewendet.

Konsole

So verknüpfen Sie eine DB-Parametergruppe mit einer DB-Instance

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die DB-Instance, die Sie ändern möchten.
3. Wählen Sie Modify aus. Die Seite Modify DB instance (DB-Instance ändern) wird angezeigt.
4. Ändern Sie die Einstellung für DB-Parametergruppen .
5. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
6. (Optional) Klicken Sie auf Apply immediately (Sofort anwenden), um die Änderungen direkt zu übernehmen. Die Auswahl dieser Option kann in einigen Fällen einen Ausfall verursachen.
7. Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen. Wenn sie korrekt sind, wählen Sie Modify DB Instance (DB-Instance ändern) aus, um Ihre Änderungen zu speichern.

   Oder klicken Sie auf Zurück, um Ihre Änderungen zu bearbeiten, oder auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

Um eine DB-Parametergruppe mit einer DB-Instance zu verknüpfen, verwenden Sie den AWS CLI `modify-db-instance`Befehl mit den folgenden Optionen:

- `--db-instance-identifier`
- `--db-parameter-group-name`

Im folgenden Beispiel wird die `mydbpg`-DB-Parametergruppe mit der `database-1`-DB-Instance verknüpft. Die Änderungen werden mit sofort übernomme `--apply-immediately`. Verwenden Sie `--no-apply-immediately`, um Änderungen im nächsten Wartungszeitraum anzuwenden.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-instance \
```

```
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

Für Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

RDS-API

Um eine DB-Parametergruppe zu einer DB-Instance zuzuordnen, verwenden Sie die RDS-API-Operation `ModifyDBInstance` mit den folgenden Parametern:

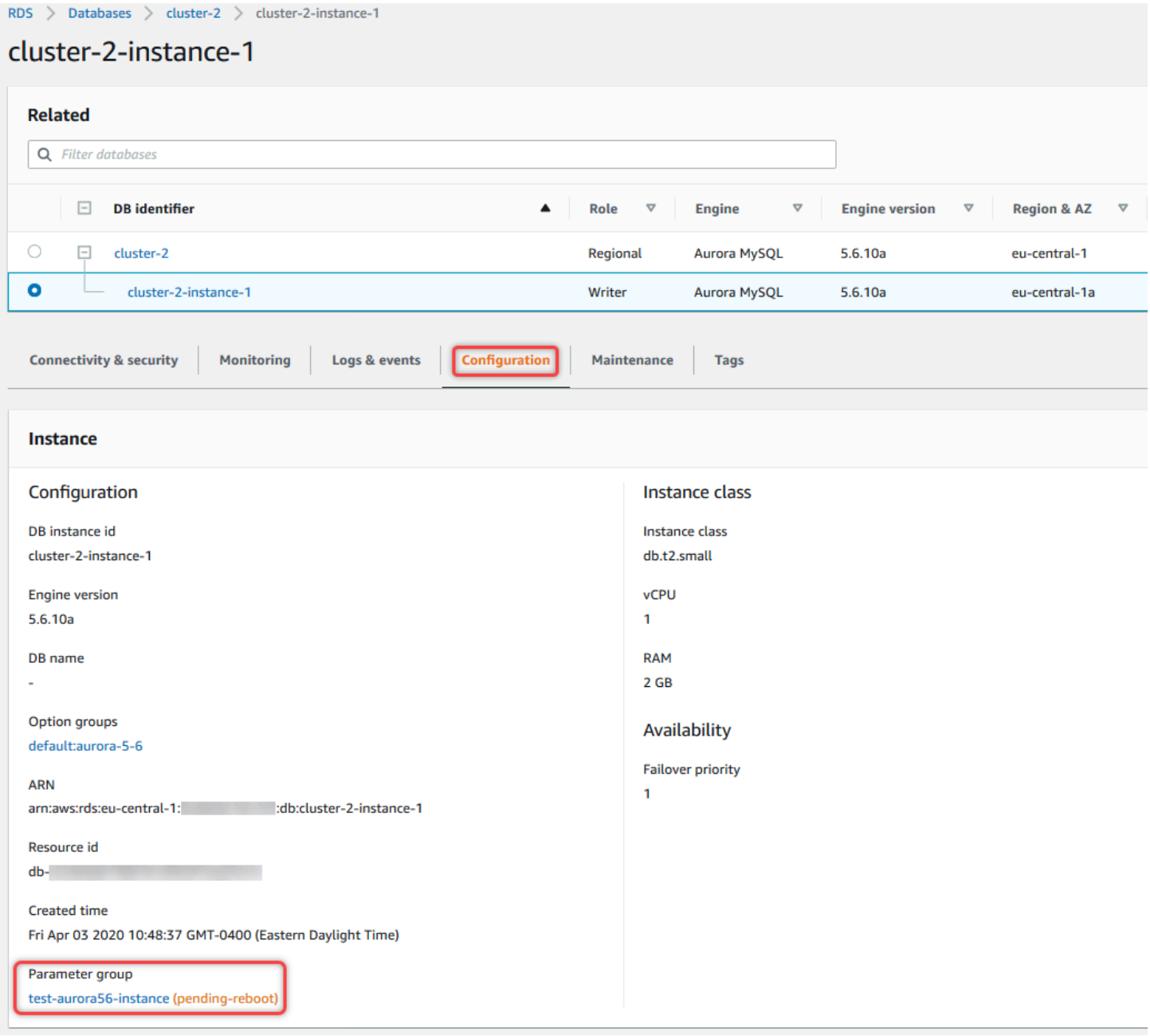
- `DBInstanceName`
- `DBParameterGroupName`

Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora

Sie können die Parameterwerte in einer benutzerdefinierten DB-Parametergruppe ändern. Die Parameterwerte in einer Standard-DB-Parametergruppe können nicht geändert werden. Änderungen bei Parametern in einer benutzerdefinierten DB-Parametergruppe werden auf alle DB-Instances angewandt, die dieser DB-Parametergruppe zugeteilt sind.

Es gibt zwei Arten von Parametern: dynamische Parameter und statische Parameter. Änderungen an dynamischen Parametern werden sofort und ohne Neustart auf die DB-Instance angewendet. Änderungen an statischen Parametern werden erst nach dem Neustart der DB-Instance übernommen.

In der RDS-Konsole wird der Status einer DB-Parametergruppe, die einer DB-Instance zugeordnet ist, auf der Registerkarte Konfiguration angezeigt. Wenn beispielsweise auf der DB-Instance die neuesten Änderungen an der zugeordneten DB-Parametergruppe noch nicht übernommen wurden, gibt die RDS-Konsole für diese DB-Parametergruppe den Status pending-reboot (Neustart ausstehend) an. Damit die neuesten Parameteränderungen für diese DB-Instance übernommen werden, starten Sie die DB-Instance manuell neu.



Konsole

So ändern Sie Parameter in einer DB-Parametergruppe

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste den Namen der zu ändernden Parametergruppe aus.
4. Wählen Sie für Parameter group actions (Parametergruppenaktionen) die Option Bearbeiten.

5. Ändern Sie wie gewünscht die Werte der Parameter. Sie können durch die Parameter scrollen, in dem Sie die Pfeiltasten oben rechts im Dialogfeld verwenden.

   Die Werte in einer Standardparametergruppe können Sie nicht ändern.

6. Wählen Sie Save Changes.

AWS CLI

Um eine DB-Parametergruppe zu ändern, verwenden Sie den AWS CLI `modify-db-parameter-group`Befehl mit den folgenden erforderlichen Optionen:

- `--db-parameter-group-name`
- `--parameters`

Im folgenden Beispiel werden die Werte `max_connections` und `max_allowed_packet` in der DB-Parametergruppe mydbparametergroup geändert.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
DBPARAMETERGROUP  mydbparametergroup
```

RDS-API

Zum Ändern einer DB-Parametergruppe verwenden Sie die RDS-API-Operation `ModifyDBParameterGroup` mit den folgenden erforderlichen Parametern:

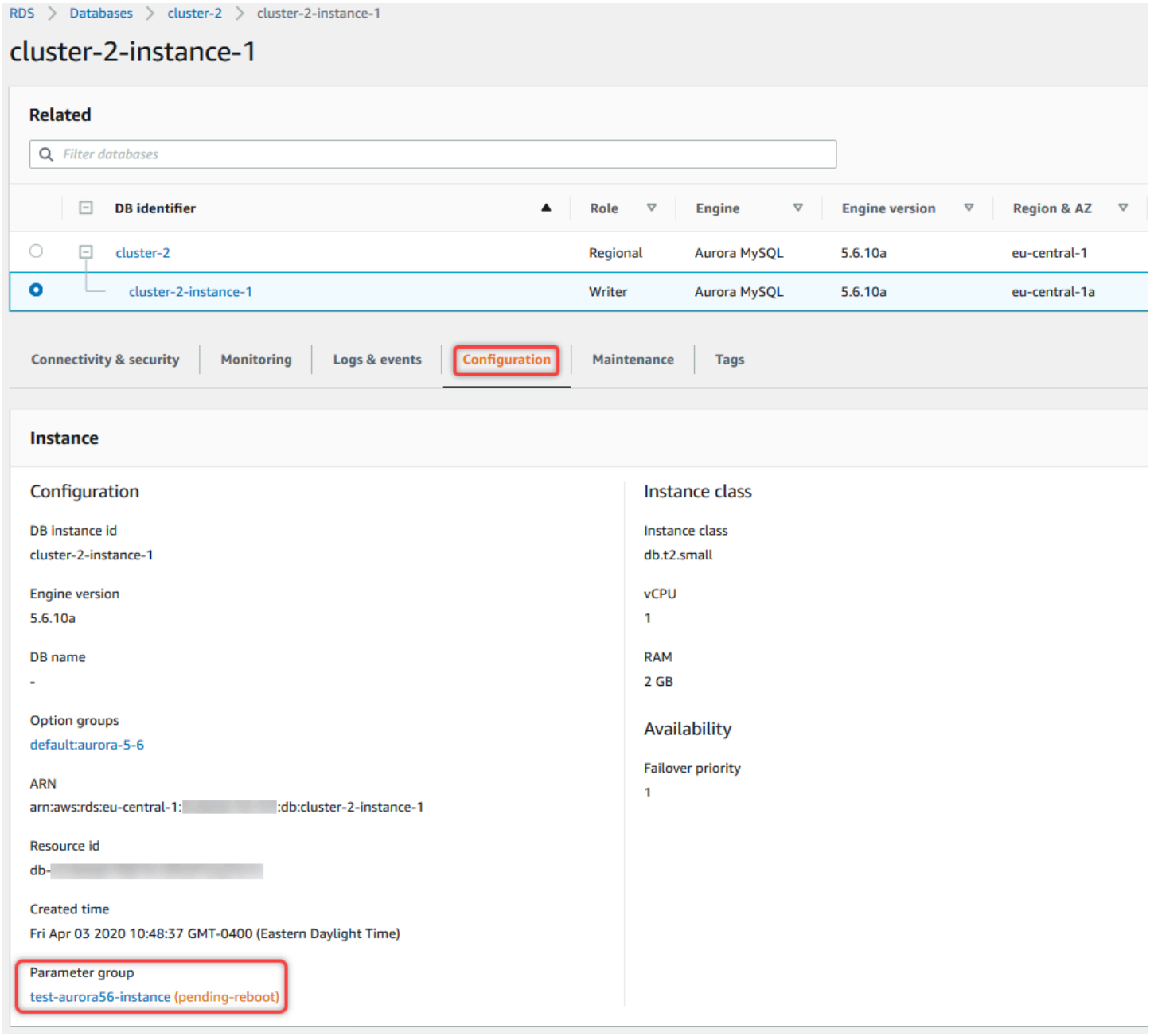
- `DBParameterGroupName`
- `Parameters`

Zurücksetzen von Parametern in einer DB-Parametergruppe auf ihre Standardwerte in Amazon Aurora

Sie können Parameterwerte in einer vom Kunden erstellten DB-Parametergruppe auf ihre Standardwerte zurücksetzen. Änderungen bei Parametern in einer benutzerdefinierten DB-Parametergruppe werden auf alle DB-Instances angewandt, die dieser DB-Parametergruppe zugeteilt sind.

Wenn Sie die Konsole verwenden, können Sie bestimmte Parameter auf ihre Standardwerte zurücksetzen. Sie können jedoch nicht einfach alle Parameter in der DB-Parametergruppe auf einmal zurücksetzen. Wenn Sie die AWS CLI oder die RDS-API verwenden, können Sie bestimmte Parameter auf ihre Standardwerte zurücksetzen. Sie können auch alle Parameter in der DB-Parametergruppe auf einmal zurücksetzen.

Änderungen an einigen Parametern werden sofort ohne Neustart auf die DB-Instance angewendet. Änderungen an anderen Parametern werden erst angewendet, nachdem die DB-Instance neu gestartet wurde. In der RDS-Konsole wird der Status einer DB-Parametergruppe, die einer DB-Instance zugeordnet ist, auf der Registerkarte Konfiguration angezeigt. Nehmen wir beispielsweise an, dass auf der DB-Instance die neuesten Änderungen an der zugeordneten DB-Parametergruppe noch nicht übernommen wurden. In diesem Fall gibt die RDS-Konsole für die DB-Parametergruppe den Status pending-reboot an. Damit die neuesten Parameteränderungen für diese DB-Instance übernommen werden, starten Sie die DB-Instance manuell neu.



> **Note**
>
> In einer Standard-DB-Parametergruppe werden Parameter immer auf ihre Standardwerte festgelegt.

Konsole

So setzen Sie Parameter in einer DB-Parametergruppe auf ihre Standardwerte zurück

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste die Parametergruppe aus.
4. Wählen Sie für Parameter group actions (Parametergruppenaktionen) die Option Bearbeiten.
5. Wählen Sie die Parameter aus, die Sie auf ihre Standardwerte zurücksetzen möchten. Sie können durch die Parameter scrollen, in dem Sie die Pfeiltasten oben rechts im Dialogfeld verwenden.

   Sie können die Werte in einer Standardparametergruppe nicht zurücksetzen.
6. Wählen Sie Reset (Zurücksetzen) und bestätigen Sie dann mit Reset parameters (Parameter zurücksetzen).

AWS CLI

Um einige oder alle Parameter in einer DB-Parametergruppe zurückzusetzen, verwenden Sie den AWS CLI `reset-db-parameter-group`Befehl mit der folgenden erforderlichen Option:`--db-parameter-group-name`.

Um alle Parameter in der DB-Parametergruppe zurückzusetzen, geben Sie die Option `--reset-all-parameters` an. Um bestimmte Parameter zurückzusetzen, geben Sie die Option `--parameters` an.

Im folgenden Beispiel werden alle Parameter in der DB-Parametergruppe namens mydbparametergroup auf ihre Standardwerte zurückgesetzt.

Example

Für Linux, macOS oder Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Für Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

Im folgenden Beispiel werden die Optionen `max_connections` und `max_allowed_packet` in der DB-Parametergruppe namens mydbparametergroup auf ihre Standardwerte zurückgesetzt.

Example

Für Linux, macOS oder Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Für Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
DBParameterGroupName  mydbparametergroup
```

RDS-API

Um Parameter in einer DB-Parametergruppe auf ihre Standardwerte zurückzusetzen, verwenden Sie den RDS-API-Befehl `ResetDBParameterGroup` mit dem folgenden erforderlichen Parameter: `DBParameterGroupName`.

Um alle Parameter in der DB-Parametergruppe zurückzusetzen, setzen Sie den Parameter `ResetAllParameters` auf `true`. Um bestimmte Parameter zurückzusetzen, geben Sie den Parameter `Parameters` an.

Kopieren einer DB-Parametergruppe in Amazon Aurora

Sie können benutzerdefinierte DB-Parametergruppen, die Sie erstellt haben, kopieren. Das Kopieren einer Parametergruppe kann eine bequeme Lösung sein. Ein Beispiel ist, wenn Sie eine DB-

Parametergruppe erstellt haben und die am häufigsten verwendeten Parameter und Werte in einer neuen DB-Parametergruppe aufnehmen möchten. Sie können eine DB-Parametergruppe kopieren, indem Sie den verwenden AWS-Managementkonsole. Sie können auch den AWS CLI copy-db-parameter-groupBefehl oder den Vorgang RDS API Copy DBParameter Group verwenden.

Nachdem Sie eine DB-Parametergruppe kopiert haben, warten Sie mindestens fünf Minuten, bevor Sie die erste DB-Instance erstellen, die diese DB-Parametergruppe als Standardparametergruppe verwendet. So kann die Kopieraktion in Amazon RDS abgeschlossen werden, bevor die Parametergruppe verwendet wird. Dies ist insbesondere für Parameter wichtig, die beim Erstellen der Standarddatenbank für eine DB-Instance wichtig sind. Ein Beispiel ist der Zeichensatz für die mit dem Parameter `character_set_database` definierte Standarddatenbank. Verwenden Sie die Option Parameter Groups der Amazon RDS-Konsole oder den describe-db-parametersBefehl, um zu überprüfen, ob Ihre DB-Parametergruppe erstellt wurde.

> **Note**
>
> Standardparametergruppen können nicht kopiert werden. Sie können jedoch eine neue Parametergruppe erstellen, die auf einer Standardparametergruppe basiert.
> Sie können eine DB-Parametergruppe nicht in eine andere AWS-Konto oder kopieren AWS-Region.

Konsole

So kopieren Sie eine DB-Parametergruppe

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste die zu kopierende benutzerdefinierte Parametergruppe.
4. Wählen Sie für Parameter group actions (Parametergruppenaktionen) die Option Kopieren.
5. Geben Sie unter New DB parameter group identifier (Neue DB-Parametergruppenkennung) einen Namen für die neue Parametergruppe ein.
6. Geben Sie unter Beschreibung eine Beschreibung für die neue Parametergruppe ein.
7. Wählen Sie die Option Copy aus.

AWS CLI

Um eine DB-Parametergruppe zu kopieren, verwenden Sie den AWS CLI `copy-db-parameter-group`Befehl mit den folgenden erforderlichen Optionen:

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

Im folgenden Beispiel wird eine neue DB-Parametergruppe mit dem Namen `mygroup2 that is a copy of the DB parameter group mygroup1` erstellt.

Example

Für Linux, macOS oder Unix:

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

Für Windows:

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

RDS-API

Zum Kopieren einer DB-Parametergruppe verwenden Sie die RDS-API-Aktion `CopyDBParameterGroup` mit den folgenden erforderlichen Parametern:

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`
- `TargetDBParameterGroupDescription`

Auflisten von DB-Parametergruppen in Amazon Aurora

Sie können die DB-Parametergruppen auflisten, die Sie für Ihr AWS Konto erstellt haben.

> **Note**
>
> Standard-Parametergruppen werden automatisch aus einer Vorlage für Standard-Parameter erstellt, wenn Sie eine DB-Instance für eine bestimmte DB-Engine und -Version erstellen. Diese Standardparametergruppen enthalten bevorzugte Parametereinstellungen und können nicht geändert werden. Wenn Sie eine benutzerdefinierte Parametergruppe erstellen, können Sie Parametereinstellungen ändern.

Konsole

Um alle DB-Parametergruppen für ein AWS Konto aufzulisten

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.

   Die verfügbaren DB-Parametergruppen erscheinen in einer Liste.

AWS CLI

Verwenden Sie den AWS CLI `describe-db-parameter-groups`Befehl, um alle DB-Parametergruppen für ein AWS Konto aufzulisten.

Example

Im folgenden Beispiel werden alle verfügbaren DB-Parametergruppen für ein AWS -Konto aufgelistet.

```
aws rds describe-db-parameter-groups
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

Im folgenden Beispiel wird die Parametergruppe mydbparamgroup1 beschrieben.

Für Linux, macOS oder Unix:

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

Für Windows:

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

RDS-API

Verwenden Sie den DescribeDBParameterGroupsRDS-API-Vorgang, um alle DB-Parametergruppen für ein AWS Konto aufzulisten.

Anzeigen von Parameterwerten für eine DB-Parametergruppe in Amazon Aurora

Sie können eine Liste aller Parameter in einer DB-Parametergruppe und ihren Werten erhalten.

Konsole

So können Sie die Parameterwerte für eine DB-Parametergruppe ansehen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.

   Die verfügbaren DB-Parametergruppen erscheinen in einer Liste.
3. Wählen Sie den Namen der Parametergruppe, um deren Parameterliste anzuzeigen.

AWS CLI

Um die Parameterwerte für eine DB-Parametergruppe anzuzeigen, verwenden Sie den AWS CLI describe-db-parametersBefehl mit dem folgenden erforderlichen Parameter.

- `--db-parameter-group-name`

Example

Im folgenden Beispiel werden die Parameter und Parameterwerte für eine DB-Parametergruppe mit dem Namen mydbparametergroup. aufgelistet.

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

Die Ausgabe des Befehls ähnelt der Folgenden:

```
DBPARAMETER  Parameter Name            Parameter Value  Source           Data Type
 Apply Type  Is Modifiable
DBPARAMETER  allow-suspicious-udfs                      engine-default   boolean
 static      false
DBPARAMETER  auto_increment_increment                   engine-default   integer
 dynamic     true
DBPARAMETER  auto_increment_offset                      engine-default   integer
 dynamic     true
DBPARAMETER  binlog_cache_size         32768            system           integer
 dynamic     true
DBPARAMETER  socket                    /tmp/mysql.sock  system           string
 static      false
```

RDS-API

Um die Parameterwerte für eine DB-Parametergruppe anzuzeigen, verwenden Sie den RDS-API-Befehl `DescribeDBParameters` mit dem folgenden erforderlichen Parameter.

- `DBParameterGroupName`

Löschen einer DB-Parametergruppe in Amazon Aurora

Sie können eine DB-Parametergruppe mithilfe der AWS-Managementkonsole AWS CLI, oder RDS-API löschen. Eine Parametergruppe kann nur gelöscht werden, wenn sie mit keiner DB-Instance verknüpft ist.

Konsole

So löschen Sie eine DB-Parametergruppe:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.

   Die verfügbaren DB-Parametergruppen erscheinen in einer Liste.
3. Wählen Sie den Namen der zu löschenden Parametergruppen aus.
4. Wählen Sie Aktionen und anschließend Löschen aus.
5. Überprüfen Sie die Namen der Parametergruppen und wählen Sie dann Löschen.

AWS CLI

Um eine DB-Parametergruppe zu löschen, verwenden Sie den AWS CLI `delete-db-parameter-group`Befehl mit dem folgenden erforderlichen Parameter.

- `--db-parameter-group-name`

Example

Im folgenden Beispiel wird die DB-Parametergruppe namens mydbparametergroup gelöscht.

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS-API

Verwenden Sie den RDS-API-Befehl `DeleteDBParameterGroup` mit de folgenden erforderlichen Parameter, um eine DB-Parametergruppe zu löschen.

- `DBParameterGroupName`

Vergleichen von DB-Parametergruppen

Mit der AWS-Managementkonsole können Sie die Unterschiede zwischen zwei DB-Parametergruppen anzeigen.

Die angegebenen Parametergruppen müssen beide DB-Parametergruppen oder DB-Cluster-Parametergruppen sein. Dies gilt, auch wenn die DB-Engine und die Version identisch sind. Beispielsweise können Sie eine DB-Parametergruppe von `aurora-mysql8.0` (Aurora-MySQL-Version 3) nicht mit einer DB-Cluster-Parametergruppe von `aurora-mysql8.0` vergleichen.

Sie können DB-Parametergruppen von Aurora MySQL und RDS für MySQL vergleichen, auch für verschiedene Versionen. Im Gegensatz dazu können Sie DB-Parametergruppen von Aurora PostgreSQL und RDS für PostgreSQL jedoch nicht vergleichen.

So vergleichen Sie zwei DB-Parametergruppen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste die beiden zu vergleichenden Parametergruppen.

   > **Note**
   >
   > Um eine Standardparametergruppe mit einer benutzerdefinierten Parametergruppe zu vergleichen, wählen Sie zuerst die Standardparametergruppe auf der Registerkarte Standard und dann die benutzerdefinierte Parametergruppe auf der Registerkarte Benutzerdefiniert aus.

4. Wählen Sie unter Aktionen die Option Vergleichen aus.

Festlegen von DB-Parametern

Zu den DB-Parametertypen gehören die folgenden:

- Ganzzahl
- Boolesch
- String
- Long
- Double
- Zeitstempel
- Objekt anderer definierter Datentypen
- Array von Werten vom Typ Ganzzahlwert, Boolesch, String, Long, Double, Zeitstempel oder Objekt

Sie können auch Ganzzahl- und Boolesche Parameter mit Ausdrücken, Formeln und Funktionen angeben.

Inhalt

- DB-Parameter-Formeln
  - DB-Parameter-Formel-Variablen
  - DB-Parameter-Formel-Operatoren
- DB-Parameter-Funktionen
- Protokollausdrücke von DB-Parametern
- Beispiele für DB-Parameterwerte

DB-Parameter-Formeln

Eine DB-Parameterformel ist ein Ausdruck, der in einen Ganzzahlwert oder einen booleschen Wert aufgelöst wird. Sie schließen den Ausdruck in Klammern ein: {}. Sie können eine Formel entweder für einen DB-Parameterwert oder als Argument für eine DB-Parameterfunktion verwenden.

Syntax

```
{FormulaVariable}
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

DB-Parameter-Formel-Variablen

Jede Formelvariable gibt einen Ganzzahlwert oder einen booleschen Wert zurück. Bei den Namen aller Variablen muss die Groß- und Kleinschreibung beachtet werden.

AllocatedStorage

Gibt einen Ganzzahlwert zurück, der die Größe des Datenvolumens in Byte darstellt.

DBInstanceClassMemory

Gibt eine Ganzzahl für die Anzahl der Speicherbytes zurück, die für den Datenbankprozess verfügbar sind. Diese Zahl wird intern berechnet, indem mit der Gesamtspeichermenge für die DB-Instance-Klasse begonnen wird. Hiervon wird bei der Berechnung der Speicher subtrahiert, der für das Betriebssystem und die RDS-Prozesse reserviert ist, die die Instance verwalten. Daher ist die Zahl immer etwas niedriger als die Speicherzahlen, die in den Instance-Klassentabellen unter DB-Instance-Klassen von Amazon Aurora genannt werden. Der genaue Wert hängt von

einer Kombination von Faktoren ab. Dazu gehören Instance-Klasse, DB-Engine und die Frage, ob der Wert für eine RDS-Instance oder eine Instance gilt, die Teil eines Aurora-Clusters ist.

DBInstanceVCPU

Gibt eine Ganzzahl zurück, die die Anzahl der virtuellen zentralen Verarbeitungseinheiten (vCPUs) darstellt, die von Amazon RDS verwendet werden, um die Instance zu verwalten.

Endpunkt-Port

Gibt eine Ganzzahl zurück, die den beim Herstellen einer Verbindung mit der DB-Instance verwendeten Port darstellt.

TrueIfReplica

Gibt 1 zurück, wenn es sich bei der DB-Instance um ein Lesereplikat handelt, und `0`, wenn dies nicht der Fall ist. Dies ist der Standardwert für den Parameter `read_only` in Aurora MySQL.

DB-Parameter-Formel-Operatoren

DB-Parameter-Formeln unterstützen zwei Operatoren: Division und Multiplikation.

Divisions-Operator: /

Dividiert den Dividend durch den Divisor, gibt einen Quotienten als Ganzzahl zurück. Dezimalzahlen in einem Quotienten werden gekürzt und nicht gerundet.

Syntax

```
dividend / divisor
```

Die Dividend- und Divisor-Argumente müssen Ausdrücke mit Ganzzahlen sein.

Multiplikations-Operator: *

Multipliziert die Ausdrücke und gibt das Produkt der Ausdrücke zurück. Dezimalstellen in Ausdrücken werden gekürzt (und nicht gerundet).

Syntax

```
expression * expression
```

Beide Ausdrücke müssen Ganzzahlen sein.

DB-Parameter-Funktionen

Sie geben die Argumente von DB-Parameter-Funktionen entweder als Ganzzahlen oder Formeln an. Jede Funktion muss mindestens ein Argument haben. Geben Sie mehrere Argumente als kommagetrennte Liste an. Die Liste darf keine leeren Elemente aufweisen, z. B. argument1,,argument3. Bei Funktionsnamen wird zwischen Groß- und Kleinschreibung unterschieden.

IF

Gibt ein Argument zurück.

Syntax

```
IF(argument1, argument2, argument3)
```

Gibt das zweite Argument zurück, wenn das erste Argument als wahr ausgewertet wird. Gibt andernfalls das dritte Argument zurück.

GREATEST

Gibt den größten Wert aus einer Liste von Ganzzahlen oder Parameter-Formeln zurück.

Syntax

```
GREATEST(argument1, argument2,...argumentn)
```

Gibt eine Ganzzahl zurück.

LEAST

Gibt den kleinsten Wert aus einer Liste von Ganzzahlen oder Parameter-Formeln zurück.

Syntax

```
LEAST(argument1, argument2,...argumentn)
```

Gibt eine Ganzzahl zurück.

SUM

Addiert die Werte der festgelegten Ganzzahlen oder Parameter-Formeln.

Syntax

```
SUM(argument1, argument2,...argumentn)
```

Gibt eine Ganzzahl zurück.

Protokollausdrücke von DB-Parametern

Sie können einen DB-Parameterwert, der eine Ganzzahl ist, auf einen Protokollausdruck festlegen. Sie schließen den Ausdruck in Klammern ein: {}. Zum Beispiel:

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

Die `log`-Funktion repräsentiert die Protokollbasis 2. In diesem Beispiel wird auch die `DBInstanceClassMemory`-Formelvariable verwendet. Siehe DB-Parameter-Formel-Variablen.

Beispiele für DB-Parameterwerte

Diese Beispiele zeigen die Verwendung von Formeln, Funktionen und Ausdrücken für die Werte von DB-Parametern.

> **Warning**
>
> Wenn die Parameter in einer DB-Parametergruppe unpassend eingestellt werden, kann dies unbeabsichtigte unerwünschte Auswirkungen haben. Dies kann eine gestörte Leistung und Systeminstabilität umfassen. Gehen Sie mit Bedacht vor, wenn Sie Datenbank-Parameter ändern und sichern Sie Ihre Daten bevor Sie Ihre DB-Parametergruppe ändern. Führen Sie Änderungen an einer Parametergruppe immer zuerst auf einer Test-DB-Instance aus, für die zeitpunktbezogene Wiederherstellungen vorhanden sind, bevor Sie diese Änderungen für eine Produktions-DB-Instance übernehmen.

Example Verwenden der DB-Parameter-Funktion LEAST

Sie können die `LEAST`-Funktion in einem Aurora MySQL `table_definition_cache`-Parameter-Wert angeben. Verwenden Sie es, um die Anzahl der Tabellendefinitionen, die im Definitionscache gespeichert werden können, auf den kleineren Wert von `DBInstanceClassMemory` geteilt durch 393040 oder 20 000 festzulegen.

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

Migrieren von Daten zu einem Amazon Aurora-DB-Cluster

Sie haben mehrere Möglichkeiten, Daten von einer vorhandenen Datenbank in einen Amazon Aurora-DB-Cluster zu migrieren, abhängig von der Datenbank-Engine-Kompatibilität. Die verfügbaren Migrationsoptionen sind auch von der Quelldatenbank und der Menge der zu migrierenden Daten abhängig.

Migrieren von Daten zu einem Amazon Aurora MySQL-DB-Cluster

Sie können Daten von einer der folgenden Quellen in einen Amazon Aurora MySQL-DB-Cluster migrieren.

- Eine RDS für MySQL-DB-Instance
- einer MySQL-Datenbank außerhalb von Amazon RDS
- einer Datenbank, die nicht MySQL-kompatibel ist

Weitere Informationen finden Sie unter Migrieren von Daten zu einem Amazon Aurora MySQL-DB-Cluster.

Migrieren von Daten zu einem Amazon Aurora PostgreSQL-DB-Cluster

Sie können Daten von einer der folgenden Quellen in einen Amazon Aurora PostgreSQL-DB-Cluster migrieren.

- Amazon-RDS-PostgreSQL-DB-Instance
- Einer Datenbank, die nicht PostgreSQL-kompatibel ist

Weitere Informationen finden Sie unter Migrieren von Daten nach Amazon Aurora mit PostgreSQL-Kompatibilität.

Erstellen eines Amazon-ElastiCache-Caches mithilfe der Einstellungen des Aurora-DB-Clusters

ElastiCache ist ein vollständig verwalteter In-Memory-Cache-Dienst, der Lese- und Schreiblatenzen im Mikrosekundenbereich bietet und flexible Anwendungsfälle in Echtzeit unterstützt. ElastiCache kann Ihnen helfen, die Anwendungs- und Datenbankleistung zu beschleunigen. Sie können ElastiCache als primären Datenspeicher für Anwendungsfälle verwenden, für die keine Datenbeständigkeit erforderlich ist, z. B. Gaming Leaderboards, Streaming und Datenanalysen. Gleichzeitig trägt ElastiCache dazu bei, die mit der Bereitstellung und Verwaltung einer verteilten Datenverarbeitungsumgebung verbundene Komplexität zu beseitigen. Weitere Informationen finden Sie unter Häufige Anwendungsfälle von ElastiCache und wie ElastiCache für Memcached helfen kann sowie unter Häufige Anwendungsfälle von ElastiCache und wie ElastiCache für Redis OSS helfen kann. Sie können die Amazon-RDS-Konsole zum Erstellen von ElastiCache-Caches verwenden.

Sie können Amazon ElastiCache in zwei Formaten betreiben. Sie können mit einem Serverless-Cache beginnen oder einen eigenen Cache-Cluster entwerfen. Wenn Sie Ihren eigenen Cache-Cluster entwerfen möchten: ElastiCache funktioniert sowohl mit der Redis-OSS- als auch mit der Memcached-Engine. Wenn Sie sich nicht sicher sind, welche Engine Sie verwenden möchten, finden Sie weitere Informationen unter Memcached und Redis OSS im Vergleich. Weitere Informationen über Amazon ElastiCache finden Sie im Amazon-ElastiCache-Benutzerhandbuch.

Themen

- Überblick über die ElastiCache-Cache-Erstellung mit den Einstellungen des Aurora-DB-Clusters
- Erstellen eines ElastiCache-Caches mit den Einstellungen eines Aurora-DB-Clusters

Überblick über die ElastiCache-Cache-Erstellung mit den Einstellungen des Aurora-DB-Clusters

Sie können einen ElastiCache-Cache von Amazon RDS aus erstellen und dabei dieselben Konfigurationseinstellungen verwenden wie die eines neu erstellten oder vorhandenen Aurora DB-Clusters.

Nachfolgend sind einige Anwendungsfälle aufgeführt, um einen ElastiCache-Cache mit Ihrem DB-Cluster zu verknüpfen:

- Sie können Kosten sparen und Ihre Leistung verbessern, indem Sie ElastiCache mit RDS verwenden, anstatt RDS allein auszuführen.
- Sie können den ElastiCache-Cache als primären Datenspeicher für Anwendungen verwenden, für die keine Datenbeständigkeit erforderlich ist. Vorhandene Anwendungen, die Redis OSS oder Memcached verwenden, können ElastiCache fast ohne Änderungen verwenden.

Wenn Sie einen ElastiCache-Cache von RDS erstellen, erbt dieser die folgenden Einstellungen von dem zugehörigen Aurora-DB-Cluster:

- ElastiCache-Konnektivitätseinstellungen
- ElastiCache-Sicherheitseinstellungen

Sie können auch die Cache-Konfigurationseinstellungen gemäß Ihren Anforderungen festlegen.

Einrichten von ElastiCache in Ihren Anwendungen

Ihre Anwendungen müssen für die Nutzung von ElastiCache-Caches eingerichtet sein. Sie können die Cache-Leistung auch optimieren und verbessern, indem Sie Ihre Anwendungen so einrichten, dass sie je nach Ihren Anforderungen Caching-Strategien verwenden.

- Informationen zum Zugriff auf Ihren ElastiCache-Cache und zu den ersten Schritten finden Sie unter Erste Schritte mit ElastiCache (Redis OSS) und Erste Schritte mit ElastiCache (Memcached).
- Weitere Informationen zu Caching-Strategien finden Sie unter Caching-Strategien und bewährte Methoden für Memcached und Caching-Strategien und bewährte Methoden für Redis OSS.
- Weitere Informationen zur Hochverfügbarkeit in Clustern von ElastiCache (Redis OSS) finden Sie unter Hochverfügbarkeit mithilfe von Replikationsgruppen.
- Möglicherweise fallen Kosten für den Backup-Speicher, den Datentransfer innerhalb oder zwischen Regionen oder die Verwendung von AWS Outposts an. Weitere Details finden Sie unter Amazon ElastiCache: Preise.

Erstellen eines ElastiCache-Caches mit den Einstellungen eines Aurora-DB-Clusters

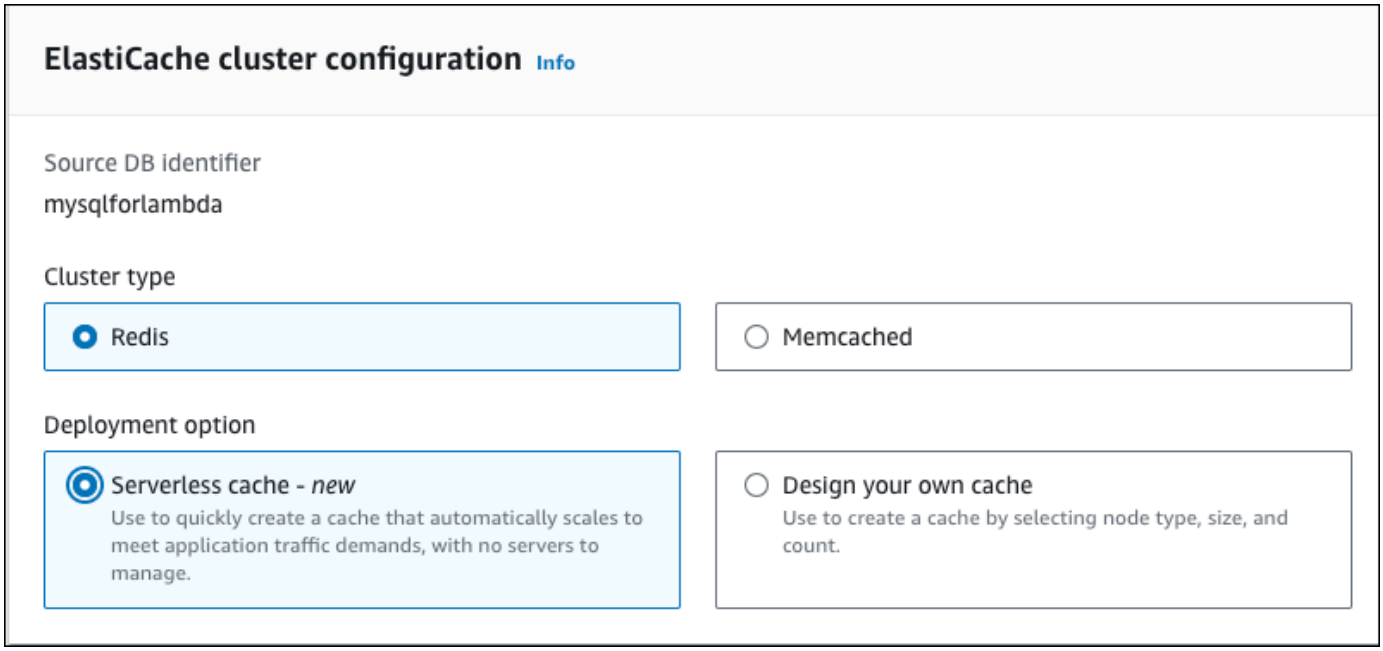
Sie können einen ElastiCache-Cache für Ihre Aurora-DB-Cluster mit Einstellungen erstellen, die Sie von dem DB-Cluster übernehmen.

Erstellen eines ElastiCache-Caches mit den Einstellungen eines DB-Clusters

1. Befolgen Sie die Anleitungen unter Erstellen eines Amazon Aurora-DB Clusters, um einen DB-Cluster zu erstellen.
2. Nach dem Erstellen eines Aurora-DB-Clusters zeigt die Konsole das Fenster Vorgeschlagene Add-ons an. Wählen Sie ElastiCache-Cluster aus RDS mit den DB-Einstellungen erstellen aus.

   Wählen Sie für eine bestehende Datenbank auf der Seite Datenbanken den erforderlichen DB-Cluster aus. Wählen Sie im Dropdown-Menü Aktionen die Option ElastiCache-Cluster erstellen aus, um einen ElastiCache-Cache in RDS zu erstellen, der dieselben Einstellungen wie Ihr vorhandener Aurora-DB-Cluster hat.

   Im ElastiCache-Konfigurationsabschnitt zeigt die Quell-DB-ID an, von welchem DB-Cluster der ElastiCache-Cache Einstellungen erbt.
3. Wählen Sie aus, ob Sie einen Redis-OSS- oder Memcached-Cluster erstellen möchten. Weitere Informationen finden Sie unter Memcached und Redis OSS im Vergleich.



4. Wählen Sie anschließend aus, ob Sie einen Serverless-Cache erstellen oder Ihren eigenen Cache entwerfen möchten. Weitere Informationen finden Sie unter Auswählen aus den Bereitstellungsoptionen.

   Wenn Sie Serverless-Cache wählen:

   a. Geben Sie in den Cache-Einstellungen Werte für Name und Beschreibung ein.

b. Behalten Sie unter Standardeinstellungen anzeigen die Standardeinstellungen bei, um die Verbindung zwischen Ihrem Cache und dem DB-Cluster herzustellen.

c. Sie können die Standardeinstellungen auch bearbeiten, indem Sie Standardeinstellungen anpassen wählen. Wählen Sie die ElastiCache-Konnektivitätseinstellungen, die ElastiCache-Sicherheitseinstellungen und Maximale Nutzungsbeschränkungen aus.

5. Wenn Sie Eigenen Cache entwerfen wählen:

a. Wenn Sie Redis-OSS-Cluster auswählen, geben Sie an, ob der Cluster-Modus aktiviert oder deaktiviert bleiben soll. Weitere Informationen finden Sie unter Replikation: Redis OSS (Cluster-Modus deaktiviert) im Vergleich zu Redis OSS (Cluster-Modus aktiviert).

b. Geben Sie Werte für Name, Beschreibung und Engine-Version ein.

   Für Engine-Version wird als Standardwert die neueste Engine-Version empfohlen. Sie können auch eine Engine-Version für den ElastiCache-Cache auswählen, die Ihren Anforderungen am besten entspricht.

c. Wählen Sie den Knotentyp in der Option Knotentyp aus. Weitere Informationen finden Sie unter Verwalten von Knoten.

   Wenn Sie einen Redis-OSS-Cluster erstellen möchten, dessen Cluster-Modus auf Aktiviert festgelegt ist, geben Sie die Anzahl der Shards (Partitionen/Knotengruppen) für die Option Anzahl der Shards ein.

   Geben Sie im Feld Anzahl der Replikate die Zahl der Replikate jedes Shards ein.

   > **Note**
   >
   > Der ausgewählte Knotentyp, die Anzahl der Shards und die Anzahl der Replikate wirken sich alle auf die Cache-Leistung und die Ressourcenkosten aus. Stellen Sie sicher, dass diese Einstellungen Ihren Datenbankanforderungen entsprechen. Information zu den Preisen finden auf der Seite Preise für Amazon ElastiCache.

d. Wählen Sie die ElastiCache-Konnektivitätseinstellungen und die ElastiCache-Sicherheitseinstellungen aus. Sie können die Standardeinstellungen beibehalten oder diese Einstellungen entsprechend Ihrer Anforderungen anpassen.

6. Überprüfen Sie die Standardeinstellungen und die übernommenen Einstellungen Ihres ElastiCache-Caches. Einige Einstellungen können nach der Erstellung nicht geändert werden.

> **Note**
>
> RDS passt möglicherweise das Backup-Fenster Ihres ElastiCache-Caches an, um die Mindestanforderung von 60 Minuten zu erfüllen. Das Backup-Fenster Ihrer Quelldatenbank bleibt gleich.

7. Wenn Sie bereit sind, klicken Sie auf ElastiCache-Cache erstellen.

Die Konsole zeigt ein Bestätigungsbanner für die Erstellung des ElastiCache-Caches an. Folgen Sie dem Link im Banner zur ElastiCache-Konsole, um die Cache-Details einzusehen. Die ElastiCache-Konsole zeigt den neu erstellten ElastiCache-Cache an.

Automatische Migration von EC2-Datenbanken zu Amazon Aurora mithilfe von AWS Database Migration Service

Sie können die Aurora-Konsole verwenden, um eine EC2-Datenbank zu Aurora zu migrieren. Aurora verwendet AWS Database Migration Service (AWS DMS), um Ihre EC2-Quelldatenbank zu migrieren. AWS DMS ermöglicht Ihnen die Migration relationaler Datenbanken in Ihre AWS Cloud. Weitere Informationen zu AWS Database Migration Service, finden Sie unter Was ist AWS Database Migration Service? im AWS Database Migration Service-Benutzerhandbuch.

Um mit der Migration zu beginnen, müssen Sie einen entsprechenden DB-Cluster von Aurora erstellen, in die die Daten migriert werden sollen. Nachdem Sie Ihre Zieldatenbank erstellt haben, können Sie Ihre EC2-Datenbank in diese importieren. Bei Quelldatenbanken mit weniger als 1 TiB reduziert diese Migrationsaktion den Zeit- und Ressourcenaufwand für die Migration Ihrer Daten zu Aurora.

Übersicht

Mit der Konsole von Aurora können Sie EC2-Datenbanken in entsprechende Datenbanken von Aurora migrieren. Sie müssen eine Aurora-Datenbank erstellen, um die Migration über die Konsole zu ermöglichen.

Sie können EC2-Datenbanken für die folgenden Datenbank-Engines migrieren:

- MySQL
- PostgreSQL

Der Migrationsprozess umfasst die folgenden Schritte:

- Erstellen Sie eine äquivalente Datenbank in Aurora. Damit die Datenbanken gleichwertig sind, müssen sie dieselbe Datenbank-Engine und dieselben kompatiblen Engine-Versionen aufweisen. Sie müssen sich auch in derselben VPC befinden. Anleitungen zum Erstellen Ihrer Datenbank finden Sie unter Erstellen eines Amazon Aurora-DB Clusters .
- Wählen Sie den Replikationstyp für Ihre Datenbank aus:
  - Volllastmigration – Aurora kopiert die vollständige Quelldatenbank in die Zieldatenbank und erstellt bei Bedarf neue Tabellen in der Zieldatenbank.

> **Note**
>
> Diese Option verursacht einen Ausfall in Ihrer Aurora-Datenbank.

- Volllast- und CDC-Migration (Change Data Capture) – Ähnlich wie bei der Volllastmigration kopiert Aurora mit dieser Option die gesamte Quelldatenbank in die Zieldatenbank. Nach der Volllastmigration wendet Aurora jedoch alle erfassten Änderungen in der Quelle auf die Zieldatenbank an. Die Erfassung von Datenänderungen sammelt Änderungen an den Datenbankprotokollen über die native API der Datenbank-Engine.

  > **Note**
  >
  > Diese Option verursacht einen Ausfall in Ihrer Aurora-Datenbank.

- Erfassung von Datenänderungen (Change Data Capture, CDC) – Verwenden Sie diese Option, um Ihre Zieldatenbank während der Migration verfügbar zu halten. Aurora migriert fortlaufende Änderungen in Ihrer Quelldatenbank in die Zieldatenbank.

- Aurora erstellt die erforderlichen Netzwerkressourcen, um die Migration zu erleichtern. Sobald Aurora die erforderlichen Ressourcen erstellt hat, werden Sie über die erstellten Ressourcen informiert und Sie können die Datenübertragung initiieren.

  Die Zeit, die für den Abschluss der Migration benötigt wird, hängt vom Typ der Replikation und der Größe der Quelldatenbank ab.

Voraussetzungen

MySQL

Vergewissern Sie sich, dass Sie die folgenden Voraussetzungen erfüllen, Bevor Sie eine MySQL-Datenbank als Quelldatenbank verwenden. Diese Voraussetzungen gelten sowohl für Quellen, die von AWS verwaltet werden.

Sie müssen über ein Konto für AWS DMS mit der Replication Admin-Rolle verfügen. Die Rolle benötigt die folgenden Berechtigungen:

- REPLICATION CLIENT – Diese Berechtigung ist nur für CDC-Aufgaben erforderlich. Mit anderen Worten: "Vollständiges Laden"-Aufgaben benötigen diese Berechtigung nicht.

- REPLICATION SLAVE – Diese Berechtigung ist nur für CDC-Aufgaben erforderlich. Mit anderen Worten: "Vollständiges Laden"-Aufgaben benötigen diese Berechtigung nicht.

Der AWS DMS-Benutzer muss zudem über SELECT-Berechtigungen für die Tabellen der Quelldatenbank verfügen, die repliziert werden sollen.

Erteilen Sie die folgenden Berechtigungen, wenn Sie MySQL-spezifische Bewertungen vor der Migration verwenden.

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
```

PostgreSQL

Führen Sie vor der Migration von Daten aus einer von AWS verwalteten PostgreSQL-Quelldatenbank die folgenden Schritte aus:

- Wir empfehlen, ein AWS-Benutzerkonto mit den erforderlichen Mindestberechtigungen für die PostgreSQL-DB-Instance als Benutzerkonto für den PostgreSQL-Quellendpunkt für AWS DMS zu verwenden. Von der Verwendung des Hauptkontos wird abgeraten. Das Konto muss über die Rolle `rds_superuser` und die Rolle `rds_replication` verfügen. Die Rolle `rds_replication` erteilt Berechtigungen zur Verwaltung von logischen Slots und zum Streamen von Daten mithilfe von logischen Slots.

> **Note**
>
> Einige AWS DMS-Transaktionen sind für einige Zeit im Leerlauf, bevor die DMS-Engine sie erneut nutzt. Über den Parameter `idle_in_transaction_session_timeout` in PostgreSQL-Versionen 9.6 und höher können Sie für Transaktionen im Leerlauf ein Timeout und Fehlschlagen veranlassen.

Einschränkungen

Die folgenden Einschränkungen gelten für den automatischen Migrationsprozess:

- Ihr Zieldatenbankstatus muss Verfügbar lauten, um mit der Quelldatenbankmigration zu beginnen.
- Bei der Migration von einer MySQL-Quelldatenbank muss Ihr Aurora-Konto über die Replication Admin-Rolle verfügen. Sie müssen auch über die entsprechenden Berechtigungen für diese Rolle verfügen.
- Ihre EC2-Instance und die Zieldatenbank müssen sich in derselben VPC befinden.
- Sie können Ihre EC2-Datenbank nicht zu den folgenden Zieldatenbanken migrieren, wenn Sie die Aktion Daten aus EC2-Datenbank migrieren verwenden:
  - Aurora global database
  - Aurora Limitless database
  - Aurora Serverless v1
  - Datenbanken mit MySQL Version niedriger als 5.7
  - Datenbanken mit PostgreSQL Version niedriger als 10.4

Erstellen von IAM-Ressourcen für homogene Datenmigrationen

Aurora verwendet AWS DMS , um Ihre Daten zu migrieren. Damit Sie auf Ihre Datenbanken zugreifen und Daten migrieren können, AWS DMS wird eine serverlose Umgebung für homogene Datenmigrationen geschaffen. In dieser Umgebung ist Zugriff auf VPC-Peering, Routing-Tabellen, Sicherheitsgruppen und andere AWS Ressourcen AWS DMS erforderlich. AWS DMS Speichert außerdem Protokolle, Metriken und den Fortschritt für jede Datenmigration in Amazon CloudWatch. Um ein Datenmigrationsprojekt zu erstellen, AWS DMS ist Zugriff auf diese Dienste erforderlich.

AWS DMS Erfordert außerdem Zugriff auf die Geheimnisse, die eine Reihe von Benutzeranmeldeinformationen darstellen, um die Datenbankverbindung für die Quell- und Zielverbindung zu authentifizieren.

> **Note**
>
> Mithilfe der Aktion Daten aus EC2 Instance migrieren können Sie die Aurora verwenden, um diese IAM-Ressourcen zu generieren. Überspringen Sie diesen Schritt, wenn Sie die von der Konsole generierten IAM-Ressourcen verwenden.

Für diesen Prozess benötigen Sie die folgenden IAM-Ressourcen:

Themen

- Erstellen einer IAM-Richtlinie für homogene Datenmigrationen
- Erstellen einer IAM-Rolle für homogene Datenmigrationen
- Erstellen einer Richtlinie und Rolle für den geheimen Zugriff
- Erstellen einer IAM-Rolle für AWS DMS zur Verwaltung von Amazon VPC

Erstellen einer IAM-Richtlinie für homogene Datenmigrationen

In diesem Schritt erstellen Sie eine IAM-Richtlinie, die den Zugriff auf Amazon EC2 und CloudWatch Ressourcen ermöglicht AWS DMS . Anschließend erstellen Sie eine IAM-Rolle und fügen dieser diese Richtlinie an.

So erstellen Sie eine IAM-Richtlinie für homogene Datenmigrationen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie im Navigationsbereich Richtlinien.
3. Wählen Sie Richtlinie erstellen aus.
4. Wählen Sie auf der Seite Richtlinie erstellen die Registerkarte JSON aus.
5. Fügen Sie den folgenden JSON-Code in den Editor ein.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeVpcs",
                "ec2:DescribePrefixLists",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "arn:aws:servicequotas:*:*:vpc/L-0EA8095F"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*:log-
stream:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateRoute",
                "ec2:DeleteRoute"
            ],
            "Resource": "arn:aws:ec2:*:*:route-table/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:security-group-rule/*",
```

```
                "arn:aws:ec2:*:*:route-table/*",
                "arn:aws:ec2:*:*:vpc-peering-connection/*",
                "arn:aws:ec2:*:*:vpc/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group-rule/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AcceptVpcPeeringConnection",
                "ec2:ModifyVpcPeeringConnectionOptions"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-peering-connection/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:AcceptVpcPeeringConnection",
            "Resource": "arn:aws:ec2:*:*:vpc/*"
        }
    ]
}
```

6. Wählen Sie Weiter: Tags und Weiter: Prüfen aus.
7. Geben Sie unter Name* **HomogeneousDataMigrationsPolicy** ein und wählen Sie dann Richtlinie erstellen aus.

Erstellen einer IAM-Rolle für homogene Datenmigrationen

In diesem Schritt erstellen Sie eine IAM-Rolle, die Zugriff auf AWS Secrets Manager Amazon EC2 und CloudWatch bietet.

So erstellen Sie eine IAM-Rolle für Datenmigrationen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie im Navigationsbereich Rollen aus.
3. Wählen Sie Create role (Rolle erstellen) aus.
4. Wählen Sie auf der Seite Vertrauenswürdige Entität auswählen unter Typ der vertrauenswürdigen Entität die Option AWS -Service aus. Unter Anwendungsfälle für andere AWS -Services wählen Sie DMS.
5. Markieren Sie das Kontrollkästchen DMS und wählen Sie Weiter.
6. Wählen Sie auf der Seite „Berechtigungen hinzufügen“ HomogeneousDataMigrationsPolicydie Option aus, die Sie zuvor erstellt haben. Wählen Sie Weiter aus.
7. Geben Sie auf der Seite Rollenname festlegen und überprüfen **`HomogeneousDataMigrationsRole`** für Rollenname ein und wählen Sie Rolle erstellen aus.
8. Geben Sie auf der Seite Rollen **`HomogeneousDataMigrationsRole`** für Rollenname ein. Wählen Sie HomogeneousDataMigrationsRole.
9. Wählen Sie auf der HomogeneousDataMigrationsRoleSeite die Registerkarte Vertrauensbeziehungen aus. Wählen Sie Vertrauensrichtlinie bearbeiten aus.
10. Fügen Sie auf der Seite Vertrauensrichtlinie bearbeiten den folgenden JSON-Code in den Editor ein und ersetzen Sie den vorhandenen Text.

    JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "dms-data-migrations.amazonaws.com",
```

```
                    "dms.your_region.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Ersetzen Sie es im vorherigen Beispiel *your_region* durch den Namen Ihres AWS-Region.

Die oben genannte ressourcenbasierte Richtlinie gewährt AWS DMS Service Principals Berechtigungen zur Ausführung von Aufgaben gemäß der vom Kunden verwalteten Richtlinie. HomogeneousDataMigrationsPolicy

11. Wählen Sie Richtlinie aktualisieren.

Erstellen einer Richtlinie und Rolle für den geheimen Zugriff

Gehen Sie wie folgt vor, um Ihre Richtlinie und Rolle für den geheimen Zugriff zu erstellen, die es DMS ermöglichen, auf die Benutzeranmeldeinformationen für Ihre Quell- und Zieldatenbanken zuzugreifen.

Um die geheime Zugriffsrichtlinie und Rolle zu erstellen, die Amazon RDS den AWS Secrets Manager Zugriff auf Ihr entsprechendes Geheimnis ermöglicht

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die AWS Identity and Access Management (IAM-) Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie Richtlinien und anschließend Richtlinie erstellen aus.
3. Wählen Sie JSON aus und geben Sie die folgende Richtlinie ein, um den Zugriff auf Ihr Secret und dessen Entschlüsselung zu ermöglichen.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
```

```
            "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SecretName-ABCDEF"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

Hier *secret_arn* ist der ARN Ihres Geheimnisses, den Sie je nach `SecretsManagerSecretId` Bedarf von beiden abrufen können, und *kms_key_arn* der ARN des AWS KMS Schlüssels, mit dem Sie Ihr Geheimnis verschlüsseln, wie im folgenden Beispiel.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-2:123456789012:secret:MySQLTestSecret-qeHamH"
        },
        {
             "Effect": "Allow",
             "Action": [
                       "kms:Decrypt",
                       "kms:DescribeKey"
                      ],
             "Resource": "arn:aws:kms:us-
east-2:123456789012:key/761138dc-0542-4e58-947f-4a3a8458d0fd"
        }
     ]
}
```

> **Note**
>
> Wenn Sie den standardmäßigen Verschlüsselungsschlüssel verwenden AWS Secrets Manager, der von erstellt wurde, müssen Sie die AWS KMS Berechtigungen für *kms_key_arn* nicht angeben.
> Wenn Sie möchten, dass Ihre Richtlinie Zugriff auf beide Geheimnisse gewährt, geben Sie einfach ein zusätzliches JSON-Ressourcenobjekt für das andere an*secret_arn*.

4. Überprüfen und erstellen Sie die Richtlinie mit einem Anzeigenamen und einer optionalen Beschreibung.
5. Wählen Sie Rollen und anschließend Rolle erstellen aus.
6. Wählen Sie als Typ der vertrauenswürdigen Entität AWS -Service aus.
7. Wählen Sie in der Liste der Services DMS als vertrauenswürdigen Service aus und wählen Sie dann Weiter: Berechtigungen aus.
8. Suchen Sie nach der Richtlinie, die Sie in Schritt 4 erstellt haben, und fügen Sie sie an. Fügen Sie dann alle Tags hinzu und überprüfen Sie Ihre Rolle. Bearbeiten Sie an dieser Stelle die Vertrauensstellungen für die Rolle, um Ihren regionalen Service-Prinzipal für Amazon RDS als vertrauenswürdige Entität zu verwenden. Dieser Prinzipal weist das folgende Format auf:

```
dms.region-name.amazonaws.com
```

Hier ist *region-name* der Name Ihrer Region, z. B. `us-east-1`. Dementsprechend folgt ein regionaler Service-Prinzipal von Amazon RDS für diese Region.

```
dms.us-east-1.amazonaws.com
dms-data-migrations.amazonaws.com
```

Erstellen einer IAM-Rolle für AWS DMS zur Verwaltung von Amazon VPC

Sie müssen eine IAM-Rolle für AWS DMS erstellen, um die VPC-Einstellungen für Ihre Ressourcen zu verwalten. Diese Rolle muss für eine erfolgreiche Migration verfügbar sein.

Erstellen der **`dms-vpc-role`** für die Datenbankmigration

<result>

Dadurch wird die Rolle für DMS zur Verwaltung der VPC-Einstellungen für die Migration erstellt.
</result>

1. Melden Sie sich bei der AWS-Managementkonsole an, und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Klicken Sie im Navigationsbereich der Konsole auf Roles (Rollen) und wählen Sie dann Create role (Rolle erstellen).
3. Wählen Sie die Option AWS-Service für die Option Vertrauenswürdige Entität auswählen.

   Wählen Sie als Anwendungsfall die Option DMS aus.
4. Wählen Sie für den Schritt Berechtigungen hinzufügen die Option `AmazonDMSVPCManagementRole` und dann Weiter aus.
5. Geben Sie auf der Seite Benennen, überprüfen und erstellen `dms-vpc-role` für Rollenname an und wählen Sie Rolle erstellen aus.

Einrichten der Datenmigration für die EC2-Datenbank

Um mit der Migration von Daten aus Ihrer EC2-Quelldatenbank zu beginnen, müssen Sie eine entsprechende Aurora-Datenbank erstellen. Anleitungen zum Erstellen Ihrer Datenbank finden Sie unter Erstellen eines Amazon Aurora-DB Clusters .

Nachdem Sie Ihre Zieldatenbank erstellt haben, gehen Sie wie folgt vor, um die Datenmigration einzurichten:

Einrichten des Datenmigrationsprojekts

1. Wählen Sie die Zieldatenbank auf der Seite Datenbanken in der RDS-Konsole aus.
2. Wählen Sie das Dropdown-Menü Aktionen und dann die Option Daten aus EC2-Datenbank migrieren. Die unterstützten Zieldatenbanken finden Sie unter Einschränkungen.
3. Gehen Sie im Abschnitt EC2-Quelldatenbank auswählen wie folgt vor:

   1. Überprüfen Sie den Engine-Typ und stellen Sie sicher, dass er mit dem Ihrer Quelldatenbank übereinstimmt.

      Überprüfen Sie außerdem, ob die Engine-Versionen kompatibel sind.
   2. Wählen Sie als EC2-Instance die EC2-Instance aus, in der sich Ihre Quelldatenbank befindet.
   3. Geben Sie als Port den Port ein, auf dem Ihre Quelldatenbank Datenverkehr zulässt.

4. Wählen Sie unter Geheimnis die Option Neues Geheimnis erstellen und verwenden, falls Sie noch kein Geheimnis haben. Fügen Sie den Benutzernamen und das Passwort für Ihre Quelldatenbank hinzu. Wählen Sie außerdem den KMS-Schlüssel aus, mit dem Sie Ihr Geheimnis verschlüsseln möchten.

   Wenn Sie ein vorhandenes Geheimnis verwenden, wählen Sie ein Ein vorhandenes Geheimnis verwenden und dann das Geheimnis aus dem Dropdown-Menü aus.

5. Wenn Sie über eine IAM-Rolle verfügen, wählen Sie unter IAM-Rolle für das Geheimnis die Option Eine vorhandene IAM-Rolle verwenden aus und dann aus dem Dropdown-Menü die IAM-Rolle, die Zugriff auf die geheime ID aus dem vorherigen Schritt hat.

   Wenn Sie über keine IAM-Rolle verfügen, wählen Sie Neue IAM-Rolle erstellen und verwenden aus. Geben Sie den neuen Namen für Ihre neue Rolle in das Feld IAM-Rollenname ein. Die mit dieser Rolle verknüpften Berechtigungen finden Sie unter dem folgenden Link.

4. Gehen Sie im Abschnitt RDS-Zieldatenbank anzeigen wie folgt vor:

   1. Bestätigen Sie die Einstellungen Ihrer Zieldatenbank oben im Abschnitt.
   2. Wählen Sie unter Geheimnis die Option Neues Geheimnis erstellen und verwenden, falls Sie noch kein Geheimnis haben, das die Anmeldeinformationen Ihrer Zieldatenbank enthält.

      Wenn Sie ein vorhandenes Geheimnis verwenden, wählen Sie das Geheimnis aus dem Dropdown-Menü aus.

   3. Wählen Sie unter IAM-Rolle für das Geheimnis eine IAM-Rolle aus, die Zugriff auf das Geheimnis aus dem vorherigen Schritt hat. Sie können auch eine neue IAM-Rolle erstellen, wenn Sie noch keine IAM-Rolle haben.

      Wenn das Dropdown-Menü die IAM-Rollen nicht ausfüllt, geben Sie den ARN der IAM-Rolle im Format `arn:aws:iam:`*`account_id`*`:role/`*`roleName`* an.

5. Gehen Sie im Abschnitt Datenmigration konfigurieren wie folgt vor:

   1. Wählen Sie die Art der Datenmigration aus, indem Sie zwischen Vollständig geladen, Vollständig geladen und Change Data Capture (CDC) und Change Data Capture (CDC) wählen. Weitere Informationen zu diesen Optionen finden Sie unter Übersicht.

      Sie können den Migrationstyp nicht ändern, nachdem die Migration gestartet wurde.

2. Wenn Sie über eine bestehende IAM-Rolle verfügen, wählen Sie unter IAM-Rolle für die Datenmigration die Option Eine vorhandene IAM-Rolle verwenden und dann eine IAM-Rolle aus dem Dropdown-Menü aus, die DMS die Berechtigungen zum Erstellen der für die Migration erforderlichen Ressourcen gewährt. Wenn Sie über keine vorhandene IAM-Rolle verfügen, wählen Sie Neue IAM-Rolle erstellen und verwenden aus.

6. Vergewissern Sie sich, dass auf der Registerkarte Migrationseinstellungen anzeigen die erforderlichen Einstellungen angezeigt werden, damit Ihre Datenmigration erfolgreich eingerichtet werden kann.
7. Wählen Sie Migrieren aus, um die Einrichtung der Migration abzuschließen.

Nachdem Sie diese Schritte abgeschlossen haben, können Sie sehen, welche Ressourcen für die Datenmigration eingerichtet werden, indem Sie im Fortschrittsbanner in der Konsole auf Details anzeigen klicken. Sobald die erforderlichen Ressourcen eingerichtet sind, wird die Migration automatisch gestartet. Wenn Sie erstellen

Um mehrere Datenbanken in die Zieldatenbank zu migrieren, starten Sie diesen Vorgang erneut mit Details zur neuen EC2-Datenbank.

Verwalten von Datenmigrationen

Nachdem Sie die Aktion Daten aus EC2-Datenbank migrieren von der RDS-Konsole aus verwendet haben, startet Aurora die Migration automatisch.

Wenn Sie die AWS DMS-Konsole zur Erstellung der Migrationsressourcen verwendet haben, können Sie den Migrationsprozess starten.

Starten der Datenmigration

Gehen Sie folgendermaßen vor, um die Datenmigration zu starten:

Starten einer Datenmigration

1. Wählen Sie die Zieldatenbank auf der Seite Datenbanken in der RDS-Konsole aus.
2. Wählen Sie auf der Seite mit den Datenbankdetails die Registerkarte Datenmigrationen aus.
3. Auf der Registerkarte Datenmigrationen werden unter Zugehörige Datenmigrationen die verfügbaren Datenmigrationen aufgeführt.

Migrationen, die mit der Aurora-Konsole eingerichtet wurden, werden automatisch gestartet, sobald die erforderlichen Ressourcen eingerichtet sind.

Migrationen, die mit der DMS-Konsole eingerichtet wurden, werden auf Bereit gesetzt.

Um mit diesen Migrationen zu beginnen, wählen Sie das Dropdown-Menü Aktionen und dann Starten aus.

4. Damit wird die Datenmigration für Ihre EC2-Datenbank gestartet.

Anhalten der Datenmigration

Bei Datenmigrationen, deren Replikationstyp Volllast ist, führt das Anhalten der Migration dazu, dass der Prozess angehalten wird und nicht wieder aufgenommen werden kann. Nach dem Anhalten müssen Sie die Migration neu starten.

Bei Migrationen, bei denen der Replikationstyp auf Change Data Capture (CDC) oder Volllast und CDC eingestellt ist, können Sie den kontinuierlichen Replikationsprozess anhalten und ihn später fortsetzen.

Anhalten einer Datenmigration

1. Wählen Sie die Zieldatenbank auf der Seite Datenbanken in der RDS-Konsole aus.
2. Wählen Sie auf der Seite mit den Datenbankdetails die Registerkarte Datenmigrationen aus.
3. Auf der Registerkarte Datenmigrationen werden unter Zugehörige Datenmigrationen die laufenden Datenmigrationen aufgeführt.

   Um eine Migration anzuhalten, wählen Sie eine Datenmigration und dann in der Dropdown-Liste Aktionen die Option Anhalten aus.
4. Damit wird die Datenmigration für Ihre EC2-Datenbank angehalten.

Fortsetzen der Datenmigration

Bei Datenmigrationen, deren Replikationstyp Volllast und Change Data Capture (CDC) oder Change Data Capture (CDC)-Migration ist, können Sie den CDC-Prozess am letzten Stopppunkt fortsetzen.

Fortsetzen einer Datenmigration

1. Wählen Sie die Zieldatenbank auf der Seite Datenbanken in der RDS-Konsole aus.

2. Wählen Sie auf der Seite mit den Datenbankdetails die Registerkarte Datenmigrationen aus.
3. Auf der Registerkarte Datenmigrationen werden unter Zugehörige Datenmigrationen die angehaltenen Datenmigrationen aufgeführt.

   Um eine Migration fortzusetzen, wählen Sie eine Datenmigration und dann in der Dropdown-Liste Aktionen die Option Bearbeitung fortsetzen aus.
4. Damit wird die Datenmigration für Ihre EC2-Datenbank fortgesetzt.

Löschen der Datenmigration

Gehen Sie folgendermaßen vor, um eine zugehörige Datenmigration zu löschen

Löschen einer Datenmigration

1. Wählen Sie die Zieldatenbank auf der Seite Datenbanken in der RDS-Konsole aus.
2. Wählen Sie auf der Seite mit den Datenbankdetails die Registerkarte Datenmigrationen aus.
3. Um eine Migration zu löschen, wählen Sie eine Datenmigration und dann in der Dropdown-Liste Aktionen die Option Löschen aus.
4. Dadurch wird die Datenmigration gelöscht.

Das Löschen einer laufenden Datenmigration hat keine Auswirkungen auf Daten, die bereits in die Zieldatenbank geladen wurden.

Neustarten der Datenmigration

Gehen Sie wie folgt vor, um eine zugehörige Datenmigration von einem CDC-Startpunkt aus neu zu starten

Neustarten einer Datenmigration

1. Wählen Sie die Zieldatenbank auf der Seite Datenbanken in der RDS-Konsole aus.
2. Wählen Sie auf der Seite mit den Datenbankdetails die Registerkarte Datenmigrationen aus.
3. Um eine Migration neu zu starten, wählen Sie eine Datenmigration und dann in der Dropdown-Liste Aktionen die Option Neu starten aus.
4. Dadurch wird die Datenmigration von einem CDC-Startpunkt aus neu gestartet.

Das Neustarten einer laufenden Datenmigration hat keine Auswirkungen auf Daten, die bereits in die Zieldatenbank geladen wurden.

Überwachen Ihrer Datenmigrationen

Nachdem die homogene Datenmigration gestartet wurde, können Sie deren Status und Fortschritt überwachen. Die Durchführung von Datenmigrationen großer Datenmengen dauert mehrere Stunden. Überwachen Sie den Fortschritt Ihrer Datenmigration regelmäßig, um die Zuverlässigkeit, Verfügbarkeit und hohe Leistung Ihrer Datenmigration aufrechtzuerhalten.

So prüfen Sie den Status und den Fortschritt Ihrer Datenmigration

1. Wählen Sie die Zieldatenbank auf der Seite Datenbanken in der RDS-Konsole aus.
2. Wählen Sie auf der Seite mit den Datenbankdetails die Registerkarte Datenmigrationen aus.
3. Im Abschnitt Zugehörige Datenmigrationen sind Ihre Datenmigrationen aufgeführt. Überprüfen Sie die Spalte Status.
4. Bei fortlaufenden Datenmigrationen wird in der Spalte Migrationsfortschritt der Prozentsatz der migrierten Daten angezeigt.
5. Um den Prozess in CloudWatch zu überwachen, verwenden Sie den Link in der Spalte CloudWatch.

Migrationsstatus

Die Konsole von Aurora zeigt für jede Datenmigration, die Sie ausführen, den Status an. Die folgende Liste enthält die folgenden Status:

- `Ready`: Die Datenmigration ist startbereit.
- `Starting`: Aurora erstellt die Serverless-Umgebung für Ihre Datenmigration.
- `Load running`: Aurora führt die Volllastmigration durch.
- `Load complete, replication ongoing`: Aurora hat den vollen Ladevorgang abgeschlossen und repliziert nun die fortlaufenden Änderungen. Dieser Status gilt nur für Volllast- und CDC-Migrationen.
- `Replication ongoing`: Aurora repliziert fortlaufende Änderungen. Dieser Status gilt nur für Migrationen vom Typ CDC.
- `Stopping`: Aurora stoppt die Datenmigrationen. Dieser Status gilt, wenn Sie die Datenmigration über das Menü Aktionen beenden möchten.

- `Stopped`: Aurora hat die Datenmigration beendet.
- `Failed`: Die Datenmigration ist fehlgeschlagen. Weitere Informationen finden Sie in den Protokolldateien.
- `Restarting`: Die Datenmigration hat eine fortlaufende Datenreplikation über einen CDC-Startpunkt neu gestartet.

Tutorial: Erstellen einer eines DB-Clusters für MySQL mit einem benutzerdefinierten Parameter

In diesem Tutorial erstellen Sie eine einen DB-Cluster mit dem benutzerdefinierten Parameter . Weitere Informationen zu Parameter finden Sie unter DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster.

> **Important**
>
> Die Einrichtung eines AWS-Kontos ist kostenlos. Bei Durchführung dieses Tutorials können jedoch Kosten für die von Ihnen verwendeten AWS-Ressourcen anfallen. Sie können diese Ressourcen nach Abschluss des Tutorials löschen, wenn sie nicht mehr benötigt werden.

Um eine einen DB-Cluster mit benutzerdefinierten Konfigurationen und Einstellungen zu erstellen, können Sie benutzerdefinierte Parameter verwenden. Benutzerdefinierte Parameter sind besonders hilfreich, wenn Sie mit mehreren Datenbanken arbeiten und die Einstellungen für diese einheitlich konfigurieren möchten.

Wenn Sie diese Schritte ausführen, lernen Sie Folgendes:

- So verwenden Sie Amazon Aurora, um eine einen DB-Cluster von MySQL mit einem benutzerdefinierten Parameter zu erstellen.
- So verwenden Sie spezifische Parameter für DB-Cluster von MySQL.

Führen Sie für dieses Tutorial die folgenden Aufgaben aus:

1. Erstellen Sie eine DB-Cluster-Parametergruppe mit dem SQL-Parameter `default_password_lifetime`.
2. Erstellen Sie eine einen DB-Cluster mit der benutzerdefinierten DB-Cluster-Parametergruppe , die Sie erstellt haben.

Themen

- Voraussetzungen
- Erstellen einer Aurora-DB-Cluster-Parametergruppe
- Ändern Sie den Parameterwert in Ihrer benutzerdefinierten Parametergruppe

- Erstellen einer eines DB-Clusters von MySQL mit einem DB-Cluster-Parameter

Voraussetzungen

Für dieses Tutorial benötigen Sie ein AWS-Konto und einen Benutzer mit Administratorzugriff. Wenn Sie diese noch nicht eingerichtet haben, führen Sie die Schritte in den folgenden Abschnitten aus:

- Melden Sie sich an für ein AWS-Konto
- Erstellen eines Benutzers mit Administratorzugriff

Erstellen einer Aurora-DB-Cluster-Parametergruppe

In diesem Tutorial erfahren Sie, wie Sie eine benutzerdefinierte Parametergruppe mit default_password_lifetime für eine einen DB-Cluster von MySQL in der Konsole erstellen. Der Parameter `default_password_lifetime` steuert die Anzahl der Tage, bevor das Client-Passwort automatisch abläuft. Weitere Informationen zu anderen Parametern, die für MySQL-DB-Cluster verfügbar sind, finden Sie unter Aurora MySQL Konfigurationsparameter .

So erstellen Sie eine Parametergruppe

1. Öffnen Sie die Amazon-RDS-Konsole und wählen Sie Parametergruppen aus.
2. Wählen Sie für Benutzerdefinierte Parametergruppen die Option Parametergruppe erstellen aus.
3. Legen Sie die Details der Parametergruppe fest.
    1. Geben Sie einen Namen für die Parametergruppe ein.
    2. Geben Sie eine Beschreibung der Parametergruppe ein.
    3. Wählen Sie für Engine-Typ die Option aus. Aurora MySQL.
    4. Wählen Sie für Parametergruppenfamilie die Option aurora-mysql8.0. aus.
    5. Wählen Sie für Typ die Option DB-Cluster-Parametergruppe aus.
4. Wählen Sie Erstellen aus.

Die neue DB-Cluster-Parametergruppe wird auf der Seite Parametergruppen in der Amazon-RDS-Konsole angezeigt. Die folgenden Schritte veranschaulichen, wie Sie Parameterwerte ändern, um Ihre Parametergruppe anzupassen.

Ändern Sie den Parameterwert in Ihrer benutzerdefinierten Parametergruppe

Gehen Sie wie folgt vor, um den Parameterwert in der Parametergruppe zu ändern, die Sie in Erstellen einer Aurora-DB-Cluster-Parametergruppe erstellt haben.

So ändern Sie die Parameterwerte in Ihrer Parametergruppe

1. Öffnen Sie die Amazon-RDS-Konsole und wählen Sie Parametergruppen aus.
2. Wählen Sie für Benutzerdefinierte Parametergruppen den Namen der DB-Cluster-Parametergruppe, die Sie erstellt haben.
3. Wählen Sie Bearbeiten aus.
4. Suchen Sie im Suchfeld Filterparameter nach dem benutzerdefinierten Parameter `default_password_lifetime`.
5. Aktivieren Sie das Kontrollkästchen neben dem Parameter und geben Sie einen Wert für die Anzahl der Tage ein, die für diesen Parameter für die Gültigkeitsdauer des Kennworts festgelegt werden sollen.
6. Wählen Sie Änderungen speichern aus.

Die benutzerdefinierte Parametergruppe kann jetzt mit der dem DB-Cluster von Amazon Aurora für MySQL 8.0 verknüpft werden.

Erstellen einer eines DB-Clusters von MySQL mit einem DB-Cluster-Parameter

Erstellen Sie abschließend eine einen DB-Cluster von MySQL mit dem benutzerdefinierten Parameter , die Sie in den vorherigen Schritten erstellt haben. Im Folgenden wird beschrieben, wie Sie die den DB-Cluster von MySQL mit Ihrem benutzerdefinierten Parameter erstellen.

Erstellen einer eines DB-Clusters mit einer benutzerdefinierten Parametergruppe und einer neuen Optionsgruppe

1. Öffnen Sie die Amazon-RDS-Konsole und wählen Sie Datenbanken aus.
2. Wählen Sie Create database (Datenbank erstellen) aus.
3. Wählen Sie unter Datenbank-Erstellungsmethode auswählen die Option Standarderstellung aus.
4. Wählen Sie unter Engine-Optionen die Option Aurora (MySQL-kompatibel) aus.

5. Wählen Sie Zusätzliche Konfiguration aus.

   - Wählen Sie für Anfänglicher Datenbankname einen Namen für die den DB-Cluster aus.
   - Wählen Sie in der Dropdown-Liste DB-Cluster-Parametergruppe den Namen der DB-Cluster–Parametergruppe aus, die Sie zuvor erstellt haben.
6. Für dieses Tutorial können Sie die Standardeinstellungen für alle anderen DB-Einstellungen beibehalten oder nach Bedarf ändern.
7. Wählen Sie Create database (Datenbank erstellen) aus.

RDS erstellt eine neue einen neuen DB-Cluster von MySQL mit einer benutzerdefinierten Parametergruppe. Weitere Informationen zu dieser Datenbank finden Sie auf der Seite Datenbanken der Amazon- RDS-Konsole.

In diesem Tutorial erstellen Sie DB-Cluster von MySQL mit einem benutzerdefinierten Parameter . Diese neu erstellten DB-Cluster von MySQL verwalten die Lebensdauer des Benutzerkennworts mithilfe des Parameters `default_password_lifetime`. Um Ihre Datenbank zu optimieren, können Sie zusätzliche Einstellungen in Ihrer benutzerdefinierten Parametergruppe vornehmen und Optionen hinzufügen.

Nachdem Sie die Erstellung der benutzerdefinierten des benutzerdefinierten DB-Clusters abgeschlossen haben, sollten Sie Ihre Ressourcen löschen, um unerwünschte Kosten zu vermeiden. Um eine einen DB-Cluster zu löschen, folgen Sie den Anleitungen in Löschen von Aurora-DB-Clustern und -DB-Instances..

Verwalten eines Amazon Aurora-DB-Clusters

In diesem Abschnitt erfahren Sie, wie Sie Ihren Aurora-DB-Cluster verwalten. Aurora umfasst Cluster von Datenbankservern, die mittels Replikationstopologie verbunden sind. Daher beinhaltet die Verwaltung von Aurora häufig das Bereitstellen von Änderungen auf mehreren Servern und das Sicherstellen, dass alle Aurora-Replikate mit dem Quell-Server Schritt halten. Da Aurora den zugrunde liegenden Speicher bei wachsender Datenmenge transparent skaliert, erfordert die Verwaltung von Aurora relativ wenig Verwaltung von Festplattenspeicher. Ähnlich gilt: Da Aurora automatisch fortlaufend Sicherungen durchführt, erfordert ein Aurora-Cluster keine umfangreichen Planungen oder Ausfallzeiten für die Durchführung von Sicherungen.

Themen

- Stoppen und Starten eines Amazon Aurora-DB-Clusters
- Automatisches Verbinden einer EC2-Instance mit einem Aurora-DB-Cluster
- Automatisches Verbinden einer Lambda-Funktion mit einem Aurora-DB-Cluster
- Ändern eines Amazon Aurora-DB-Clusters
- Hinzufügen von Aurora-Replicas zu einem DB-Cluster
- Verwalten von Performance und Skalierung für einen Aurora-DB-Cluster
- Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster
- Integration von Aurora mit anderen AWS Diensten
- Warten eines Amazon Aurora-DB-Clusters
- Neustart eines Amazon Aurora DB-Clusters oder einer Amazon Aurora DB-Instance
- Ausführen eines Failovers eines DB-Clusters von Amazon Aurora
- Löschen von Aurora-DB-Clustern und -DB-Instances
- Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen
- Amazon-Ressourcennamen (ARN) in Amazon RDS
- Amazon-Aurora-Updates

Stoppen und Starten eines Amazon Aurora-DB-Clusters

Mithilfe des Stoppens und Startens von Aurora–DB-Clustern können Sie die Kosten für Entwicklungs- und Testumgebungen verwalten. Sie können vorübergehend alle DB-Instances in Ihrem Cluster stoppen, anstatt alle DB-Instances jedes Mal, wenn Sie den Cluster verwenden, einzurichten und zu entfernen.

Themen

- Übersicht über das Stoppen und Starten eines Aurora-DB-Clusters
- Einschränkungen für das Stoppen und Starten eines Aurora-DB-Clusters
- Stoppen eines Aurora-DB-Clusters
- Mögliche Operationen, während ein Aurora-DB-Cluster gestoppt ist
- Starten eines Aurora-DB-Clusters

Übersicht über das Stoppen und Starten eines Aurora-DB-Clusters

In Zeiten, in denen Sie keinen Aurora-DB-Cluster benötigen, können Sie alle Instances in dem betreffenden Cluster gleichzeitig stoppen. Sie können den Cluster jederzeit erneut starten, wenn notwendig. Durch das Starten und Stoppen werden die Einrichtungs- und Entfernungsvorgänge für Cluster erleichtert, die in der Entwicklung, für Tests oder ähnliche Aktivitäten verwendet werden, die keine kontinuierliche Verfügbarkeit erfordern. Unabhängig davon, wie viele Instances in dem Cluster enthalten sind, können Sie alle beteiligten AWS-Managementkonsole-Prozeduren mit einer einzigen Aktion ausführen.

Während Ihr DB-Cluster gestoppt ist, werden Ihnen nur Gebühren für Cluster-Speicherspeicherplatz, manuelle Snapshots und automatisierten Sicherungsspeicher innerhalb des von Ihnen festgelegten Aufbewahrungsfensters berechnet. Für DB-Instance-Stunden werden Ihnen keine Gebühren berechnet.

> **Important**
>
> Sie können einen DB-Cluster für bis zu sieben Tage anhalten. Wenn Sie Ihren DB-Cluster nach sieben Tagen nicht manuell starten, wird der Cluster automatisch gestartet, damit dieser in Bezug auf erforderliche Wartungsupdates nicht in Verzug gerät.

Um die Gebühren für einen Aurora-Cluster mit geringer Auslastung zu minimieren, können Sie den Cluster stoppen, anstatt alle seine Aurora-Replicas zu löschen. Bei Clustern mit mehr als nur ein oder zwei Instances ist ein häufiges Löschen und Neuerstellen der DB-Instances nur über die AWS CLI oder die Amazon-RDS-API praktikabel. Es kann auch schwierig sein, bei der Ausführung einer solchen Folge von Operationen die richtige Reihenfolge einzuhalten, also beispielsweise vor dem Löschen der primären Instance alle Aurora-Replicas zu löschen, um eine Aktivierung des Failover-Mechanismus zu vermeiden.

Verwenden Sie die Funktion zum Starten und Stoppen nicht, wenn Ihr DB-Cluster aktiv bleiben muss, jedoch überschüssige Kapazitäten hat. Wenn Ihr Cluster zu teuer oder zu wenig ausgelastet ist, löschen Sie eine oder mehrere DB-Instances oder ändern Sie alle Ihre DB-Instances und ordnen Sie sie einer Small-Instance-Klasse zu. Es ist nicht möglich, eine einzelne Aurora-DB-Instance zu stoppen.

Die Zeit für das Stoppen Ihres DB-Clusters hängt von Faktoren wie DB-Instance-Klassen, Netzwerkstatus, DB-Engine-Typ und Datenbankstatus ab. Der Prozess kann mehrere Minuten in Anspruch nehmen. Der Amazon-RDS-Service führt die folgenden Aktionen aus:

- Er fährt die Datenbank-Engine-Prozesse herunter.
- Er fährt die RDS-Plattformprozesse herunter.
- Beenden von Amazon-EC2-Instances

Die Zeit für den Neustart Ihres DB-Clusters hängt von Faktoren wie Datenbankgröße, DB-Instance-Klassen, Netzwerkstatus, DB-Engine-Typ und Datenbankstatus beim Herunterfahren des Clusters ab. Der Startvorgang kann von mehreren Minuten bis hin zu mehreren Stunden dauern. In der Regel dauert er mehrere Minuten. Wir empfehlen Ihnen, die Variabilität der Startzeit bei der Erstellung Ihres Verfügbarkeitsplans zu berücksichtigen.

Zum Starten des DB-Clusters führt der Service Aktionen wie die folgenden aus:

- Er stellt die zugrundeliegenden Amazon-EC2-Instances bereit.
- Er startet die RDS-Plattformprozesse.
- Er startet die Datenbank-Engine-Prozesse.
- Er stellt die DB-Instances wieder her (die Wiederherstellung erfolgt auch nach einem regulären Herunterfahren).

Einschränkungen für das Stoppen und Starten eines Aurora-DB-Clusters

Einige Aurora-Cluster können nicht gestoppt und wieder gestartet werden:

- Sie können nur einen Cluster stoppen und starten, der Teil einer globalen Aurora-Datenbank ist, wenn es sich um den einzigen Cluster in der globalen Datenbank handelt.
- Sie können einen Cluster, der über ein regionsübergreifendes Lesereplikat verfügt, nicht anhalten und starten.
- Cluster, die Teil einer Blau/Grün-Bereitstellung sind, können nicht gestoppt und gestartet werden.
- Sie können einen Aurora Serverless v1-Cluster nicht stoppen und starten. Mit Aurora Serverless v2 können Sie einen Cluster stoppen und starten.

Stoppen eines Aurora-DB-Clusters

Für die Verwendung eines Aurora-DB-Clusters oder die Durchführung von administrativen Aufgaben beginnen Sie immer mit einem aktiven Aurora-DB-Cluster, stoppen diesen und starten ihn anschließend erneut. Während Ihr Cluster gestoppt ist, werden Ihnen Cluster-Speicherplatz, manuelle Snapshots und automatisierter Sicherungsspeicher innerhalb des von Ihnen festgelegten Aufbewahrungsfensters in Rechnung gestellt, nicht jedoch DB-Instance-Stunden.

Beim Stoppvorgang werden zuerst die Aurora-Replica-Instances und dann die primäre Instance gestoppt, damit der Failover-Mechanismus nicht aktiviert wird.

Konsole

So stoppen Sie einen Aurora-Cluster:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Databases (Datenbanken) und anschließend einen Cluster aus. Sie können den Stoppvorgang von dieser Seite ausführen oder zur Detailseite für den zu stoppenden DB-Cluster navigieren.
3. Wählen Sie für Actions (Aktionen) die Option Stop temporarily (Temporär anhalten) aus.
4. Wählen Sie im Fenster Stop DB cluster temporarily (DB-Cluster vorübergehend anhalten) die Bestätigung aus, dass der DB-Cluster nach 7 Tagen automatisch neu gestartet wird.
5. Klicken Sie auf Stop temporarily (Temporär anhalten), um den DB-Cluster anzuhalten, oder auf Cancel (Abbrechen), wenn Sie den Vorgang abbrechen möchten.

AWS CLI

Zum Stoppen einer DB-Instance über die AWS CLI rufen Sie den Befehl stop-db-cluster mit den folgenden Parametern auf:

- `--db-cluster-identifier` – Der Name des Aurora-Clusters.

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

RDS-API

Zum Anhalten einer DB-Instance über die Amazon-RDS-API rufen Sie die Aktion StopDBCluster mit dem folgenden Parameter auf:

- `DBClusterIdentifier` – Der Name des Aurora-Clusters.

Mögliche Operationen, während ein Aurora-DB-Cluster gestoppt ist

Während ein Aurora-Cluster gestoppt ist, können Sie eine zeitpunktbezogene Wiederherstellung auf einen Zeitpunkt innerhalb des von Ihnen angegebenen Zeitfensters für die Aufbewahrung der automatischen Sicherung durchführen. Weitere Informationen zur Durchführung einer zeitpunktbezogenen Wiederherstellung finden Sie unter Wiederherstellen von Daten.

Die Konfiguration des Aurora-DB-Clusters oder seiner DB-Instances kann nicht geändert werden, solange der Cluster gestoppt ist. Es ist auch nicht möglich, DB-Instances zu dem Cluster hinzuzufügen oder daraus zu entfernen oder den Cluster zu löschen, wenn ihm noch DB-Instances zugeordnet sind. Vor solchen administrativen Aktionen müssen Sie den Cluster starten.

Das Stoppen eines DB-Clusters entfernt ausstehende Aktionen, mit Ausnahme der DB-Cluster-Parametergruppe oder der DB-Parametergruppen der DB-Cluster-Instances.

Aurora wendet alle geplanten Wartungen auf Ihren gestoppten Cluster an, nachdem dieser wieder gestartet wurde. Denken Sie daran: Nach sieben Tagen startet Aurora gestoppte Cluster automatisch, damit diese in Bezug auf ihren Wartungsstatus nicht zu weit zurückfallen.

Aurora führt auch keine automatischen Sicherungen durch, da sich die zugrunde liegenden Daten nicht ändern können, solange der Cluster gestoppt ist. Aurora verlängert den Aufbewahrungszeitraum für Backups für den DB-Cluster nicht, während dieser gestoppt ist.

Starten eines Aurora-DB-Clusters

Ausgangspunkt für den Start eines Aurora-DB-Clusters ist immer ein bereits im gestoppten Zustand befindlicher Aurora-Cluster. Wenn Sie den Cluster starten, werden alle seine DB-Instances wieder verfügbar. Der Cluster behält seine Konfigurationseinstellungen, wie z. B. Endpunkte, Parametergruppen und VPC-Sicherheitsgruppen, bei.

Der Neustart eines DB-Clusters dauert in der Regel mehrere Minuten.

Konsole

So starten Sie einen Aurora-Cluster:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Databases (Datenbanken) und anschließend einen Cluster aus. Sie können den Startvorgang von dieser Seite ausführen oder zur Detailseite für den zu startenden DB-Cluster navigieren.
3. Wählen Sie für Actions (Aktionen) die Option Start.

AWS CLI

Zum Starten eines DB-Clusters über die AWS CLI, rufen Sie den Befehl start-db-cluster mit den folgenden Parametern auf:

- `--db-cluster-identifier` – Der Name des Aurora-Clusters. Bei diesem Namen handelt es sich entweder um eine bestimmte Clusterkennung, die Sie beim Erstellen des Clusters ausgewählt haben, oder um die von Ihnen ausgewählte DB-Instance-Kennung mit dem Anhang `-cluster`.

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

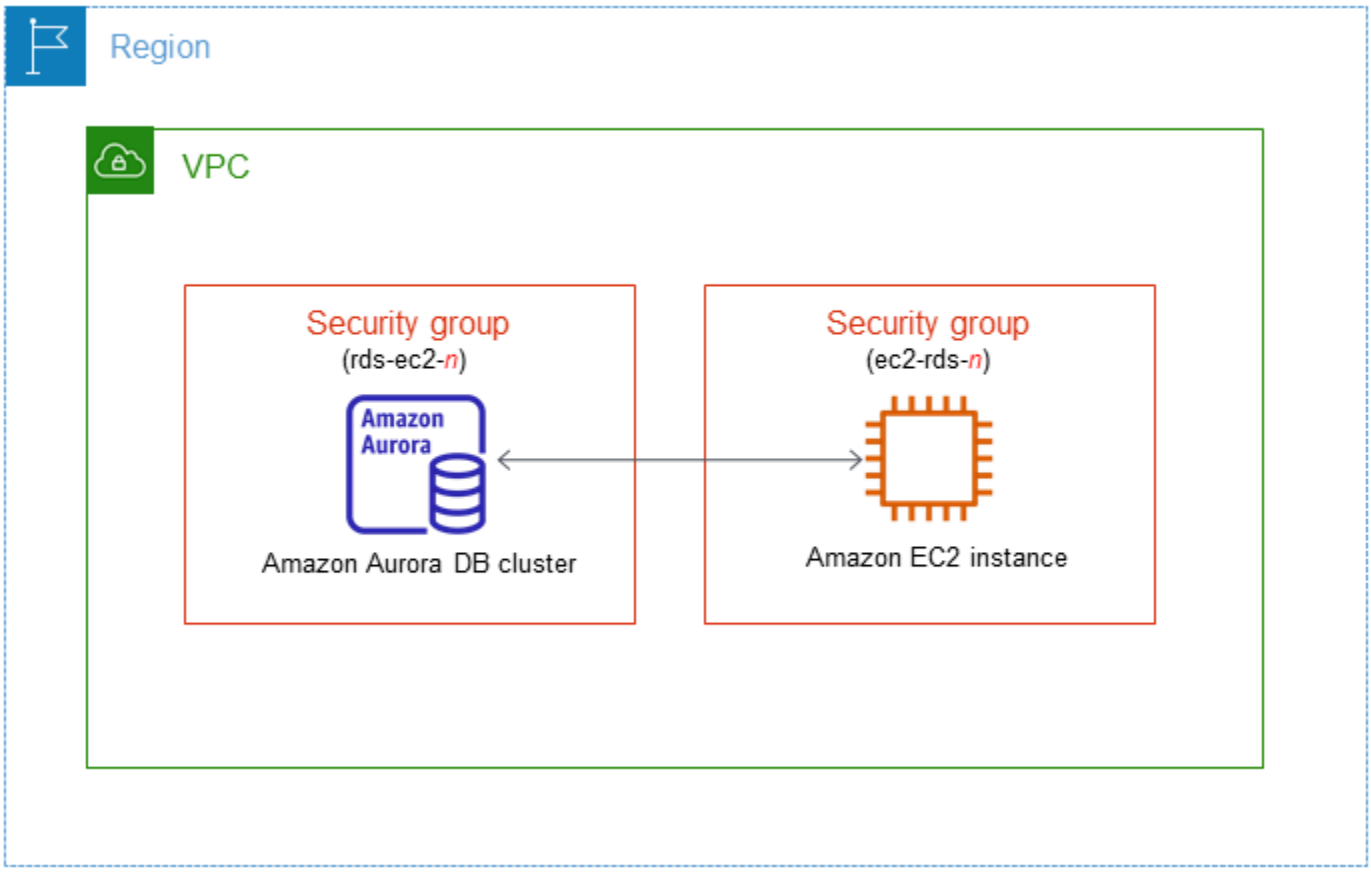
RDS-API

Zum Starten eines Aurora-DB-Clusters über die Amazon-RDS-API, rufen Sie die Aktion StartDBCluster mit dem folgenden Parameter auf:

- `DBCluster` – Der Name des Aurora-Clusters. Bei diesem Namen handelt es sich entweder um eine bestimmte Clusterkennung, die Sie beim Erstellen des Clusters ausgewählt haben, oder um die von Ihnen ausgewählte DB-Instance-Kennung mit dem Anhang `-cluster`.

Automatisches Verbinden einer EC2-Instance mit einem Aurora-DB-Cluster

Sie können die Amazon-RDS-Konsole verwenden, um das Einrichten einer Verbindung zwischen einer Amazon Elastic Compute Cloud (Amazon-EC2)-Instance und einer DB–Instance zu vereinfachen. Häufig befindet sich Ihr DB-Cluster in einem privaten Subnetz und Ihre EC2-Instance in einem öffentlichen Subnetz innerhalb einer VPC. Sie können einen SQL-Client auf Ihrer EC2-Instance verwenden, um eine Verbindung mit Ihrem DB-Cluster herzustellen. Die EC2-Instance kann auch Webserver oder Anwendungen ausführen, die auf Ihren privaten DB-Cluster zugreifen.



Wenn Sie eine Verbindung mit einer EC2-Instance herstellen möchten, die sich nicht in derselben VPC wie der Aurora-DB-Cluster befindet, sehen Sie sich die entsprechenden Szenarien unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC an.

Themen

- Übersicht über die automatische Verbindung mit einer EC2-Instance
- Automatisches Verbinden einer EC2-Instance und einem Aurora-DB-Cluster

- Anzeigen verbundener Rechenressourcen
- Herstellen einer Verbindung mit einer DB-Instance, die eine bestimmte DB-Engine ausführt

Übersicht über die automatische Verbindung mit einer EC2-Instance

Wenn Sie eine Verbindung zwischen einer EC2-Instance und einer/m einem Aurora-DB-Cluster einrichten, konfiguriert Amazon RDS automatisch die VPC-Sicherheitsgruppe für Ihre EC2-Instance und für Ihre Ihren DB-Cluster.

Im Folgenden sind Anforderungen für die Verbindung einer EC2-Instance mit einer einem Aurora-DB-Cluster aufgeführt:

- Die EC2-Instance muss sich in derselben VPC wie der DB-Cluster befinden.

  Wenn keine EC2-Instances in derselben VPC vorhanden sind, dann bietet die Konsole einen Link zum Erstellen einer solchen Instance.
- Derzeit kann der DB-Cluster kein DB-Cluster von Aurora Serverless oder Teil einer globalen Aurora-Datenbank sein.
- Der Benutzer, der die Verbindung einrichtet, muss über Berechtigungen zum Ausführen der folgenden Amazon-EC2-Vorgänge verfügen:
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Auf Ihrem Konto fallen ggf. Kosten über Availability Zones hinweg an, wenn sich die DB-Instance und die EC2-Instance in unterschiedlichen Availability Zones befinden.

Wenn Sie eine Verbindung mit einer EC2-Instance einrichten, führt Amazon RDS eine Aktion aus, die auf der aktuellen Konfiguration der Sicherheitsgruppen basiert, die dem DB-Cluster und der EC2-Instance zugeordnet sind, wie in der folgenden Tabelle beschrieben.

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle EC2-Sicherheitsgruppenkonfiguration	RDS-Aktion
Dem DB-Cluster sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rds-ec2-`*n* entspricht (wobei *n* für eine Zahl steht). Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der EC2-Instance als Quelle.	Der EC2-Instance sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `ec2-rds-`*n* entspricht (wobei *n* für eine Zahl steht). Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Quelle.	RDS führt keine Aktion aus. Es wurde bereits automatisch eine Verbindung zwischen der EC2-Instance und dem DB-Cluster konfiguriert. Da bereits eine Verbindung zwischen der EC2-Instance und der RDS-Datenbank besteht, werden die Sicherheitsgruppen nicht geändert.
Es gilt eine der folgenden Bedingungen: • Dem DB-Cluster sind keine Sicherheitsgruppen zugeordnet, deren Name dem Muster `rds-ec2-`*n* entspricht. • Dem DB-Cluster sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rds-ec2-`*n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit der EC2-Instance verwenden. Amazon RDS kann eine Sicherheitsgruppe	Es gilt eine der folgenden Bedingungen: • Der EC2-Instance ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `ec2-rds-`*n* entspricht. • Der EC2-Instance sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `ec2-rds-`*n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster verwenden. Amazon RDS kann eine Sicherhei	RDS action: create new security groups

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle EC2-Sicherheitsgruppenkonfiguration	RDS-Aktion
nicht verwenden, wenn sie keine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der EC2-Instance als Quelle enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde. Beispiele für Änderungen sind das Hinzufügen einer Regel oder das Ändern des Ports einer vorhandenen Regel.	tsgruppe nicht verwenden, wenn diese keine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Quelle enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	
Dem DB-Cluster sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rds-ec2-`*n* entspricht. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der EC2-Instance als Quelle.	Der EC2-Instance sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `ec2-rds-`*n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn diese keine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Quelle enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	RDS action: create new security groups

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle EC2-Sicherheitsgruppenkonfiguration	RDS-Aktion
Dem DB-Cluster sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rds-ec2-`*n* entspricht. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der EC2-Instance als Quelle.	Eine gültige EC2-Sicherheitsgruppe für die Verbindung ist vorhanden, jedoch nicht mit der EC2-Instance verknüpft. Die Sicherheitsgruppe trägt einen Namen, der dem Muster `ec2-rds-`*n* entspricht. Sie wurde nicht geändert. Sie enthält nur eine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Quelle.	RDS action: associate EC2 security group

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle EC2-Sicherheitsgruppenkonfiguration	RDS-Aktion
Es gilt eine der folgenden Bedingungen: • Dem DB-Cluster sind keine Sicherheitsgruppen zugeordnet, deren Name dem Muster `rds-ec2-`*n* entspricht. • Dem DB-Cluster sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rds-ec2-`*n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit der EC2-Instance verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn sie keine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der EC2-Instance als Quelle enthält. Amazon RDS kann außerdem keine Sicherheitsgruppe verwenden, die geändert wurde.	Der EC2-Instance sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `ec2-rds-`*n* entspricht. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Quelle.	RDS action: create new security groups

RDS-Aktion : neue Sicherheitsgruppen erstellen

Amazon RDS führt die folgenden Aktionen durch:

- Erstellt eine neue Sicherheitsgruppe, die dem Muster `rds-ec2-`*n* entspricht. Diese Sicherheitsgruppe enthält eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der EC2-Instance als Quelle. Diese Sicherheitsgruppe ist dem DB-Cluster zugeordnet und ermöglicht der EC2-Instance den Zugriff auf den DB-Cluster.
- Erstellt eine neue Sicherheitsgruppe, die dem Muster `ec2-rds-`*n* entspricht. Diese Sicherheitsgruppe enthält eine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Ziel. Diese Sicherheitsgruppe ist der EC2-Instance zugeordnet und ermöglicht es der EC2-Instance, Datenverkehr an den DB-Cluster zu senden.

RDS-Aktion : EC2-Sicherheitsgruppe zuordnen

Amazon RDS ordnet die gültige, vorhandene EC2-Sicherheitsgruppe der EC2-Instance zu. Diese Sicherheitsgruppe ermöglicht es der EC2-Instance, Datenverkehr an den DB-Cluster zu senden.

Automatisches Verbinden einer EC2-Instance und einem Aurora-DB-Cluster

Bevor Sie eine Verbindung zwischen einer EC2-Instance und einem Aurora-DB-Cluster einrichten, stellen Sie sicher, dass Sie die unter Übersicht über die automatische Verbindung mit einer EC2-Instance beschriebenen Anforderungen erfüllen.

Wenn Sie nach dem Konfigurieren der Verbindung Änderungen an diesen Sicherheitsgruppen vornehmen, können sich die Änderungen auf die Verbindung zwischen der EC2-Instance und dem Aurora-DB-Cluster auswirken.

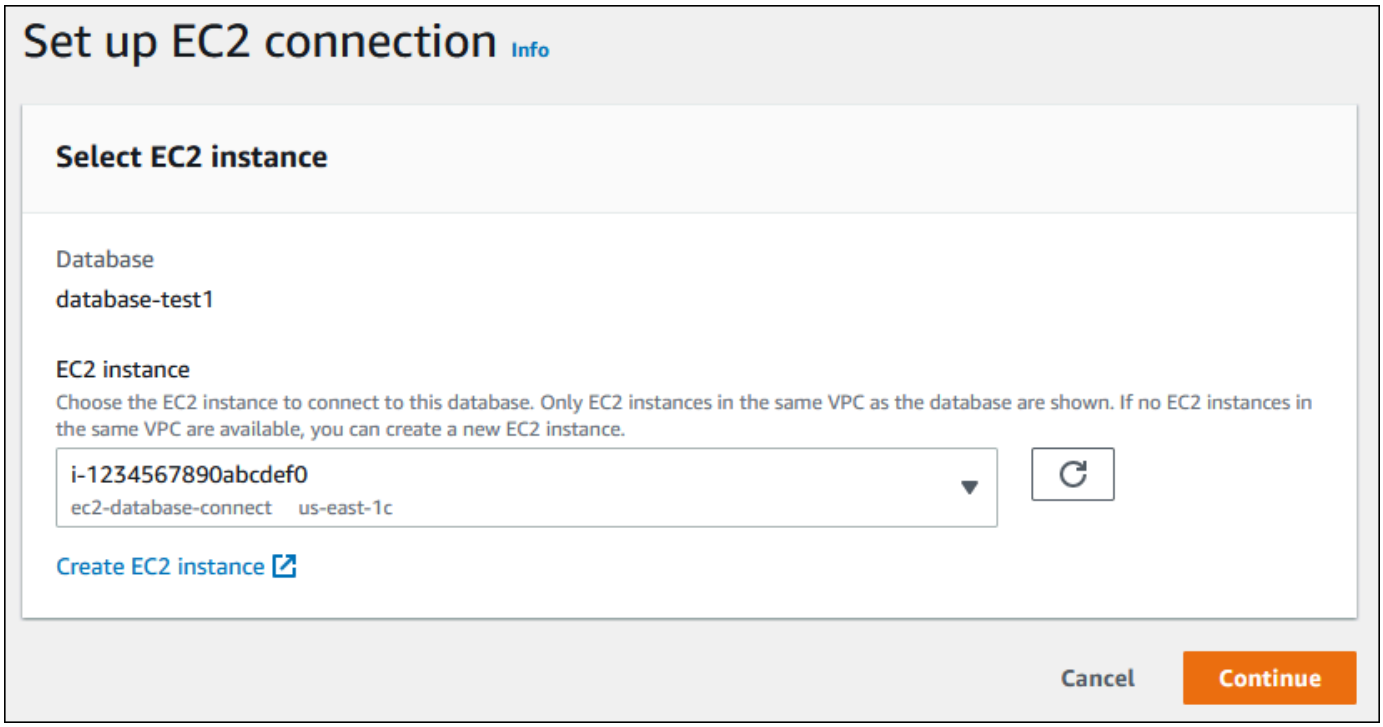
> **Note**
>
> Sie können eine Verbindung zwischen einer EC2-Instance und einem Aurora-DB-Cluster nur automatisch einrichten, indem Sie die AWS-Managementkonsole verwenden. Eine Verbindung können Sie nicht mit der AWS CLI oder der RDS-API automatisch einrichten.

So verbinden Sie eine EC2-Instance und einen Aurora-DB-Cluster automatisch

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die RDS-Datenbank aus.
3. Wählen Sie für Aktionen die Option EC2-Verbindung einrichten aus.

Die Seite Set up EC2 connection (EC2-Verbindung einrichten) wird angezeigt.

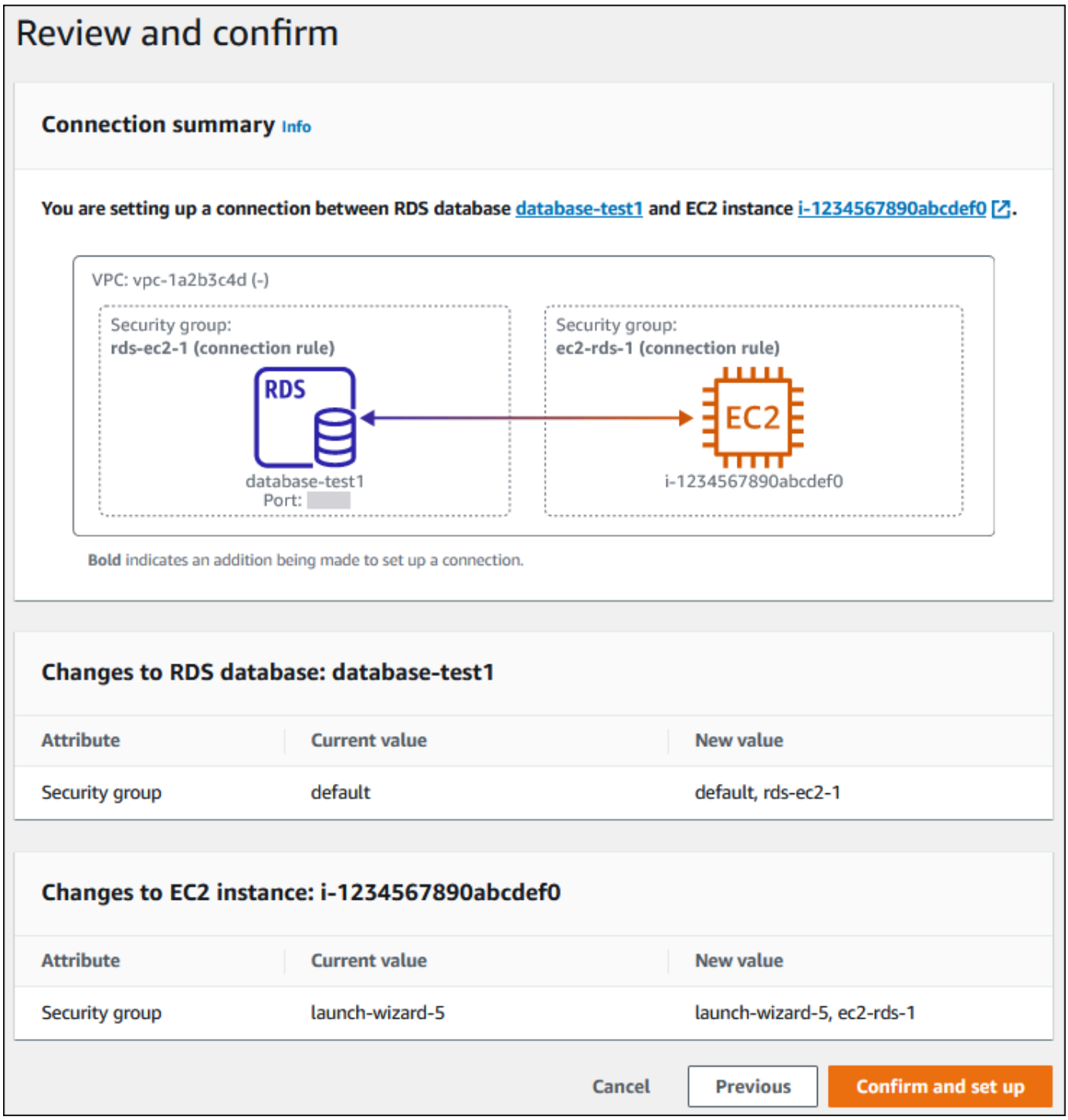
4. Wählen Sie auf der Seite Set up EC2 connection (EC2-Verbindung einrichten) die EC2-Instance aus.



Wenn keine EC2-Instances in derselben VPC vorhanden sind, wählen Sie Create EC2 instance (EC2-Instance erstellen) aus, um eine solche Instance zu erstellen. Stellen Sie in diesem Fall sicher, dass sich die neue EC2-Instance in derselben VPC wie der DB-Cluster befindet.

5. Klicken Sie auf Weiter.

Die Seite Review and confirm (Überprüfen und bestätigen) wird angezeigt.



6. Sehen Sie sich auf der Seite Review and confirm (Überprüfen und bestätigen) die Änderungen an, die RDS beim Einrichten der Verbindung mit der EC2-Instance vornehmen wird.

   Wenn die Änderungen korrekt sind, wählen Sie Bestätigen und einrichten.

   Sind die Änderungen nicht korrekt, wählen Sie Previous (Zurück) oder Cancel (Abbrechen) aus.

Anzeigen verbundener Rechenressourcen

Sie können die AWS-Managementkonsole verwenden, um die Rechenressourcen anzuzeigen, die mit einem Aurora-DB-Cluster verbunden sind. Zu den angezeigten Ressourcen gehören Rechenressourcenverbindungen, die automatisch eingerichtet wurden. Sie können die Konnektivität mit Rechenressourcen auf folgende Weise automatisch einrichten:

- Sie können die Rechenressource auswählen, wenn Sie die Datenbank erstellen.

  Weitere Informationen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.
- Sie können die Konnektivität zwischen einer vorhandenen Datenbank und einer Rechenressource einrichten.

  Weitere Informationen finden Sie unter Automatisches Verbinden einer EC2-Instance und einem Aurora-DB-Cluster.

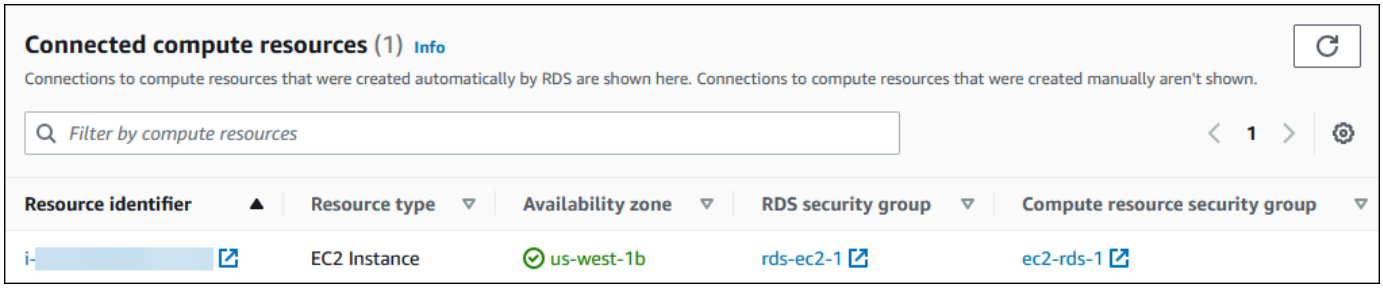
Die aufgelisteten Rechenressourcen enthalten keine Ressourcen, die manuell mit der Datenbank verbunden wurden. Sie können beispielsweise einer Rechenressource den manuellen Zugriff auf eine Datenbank erlauben, indem Sie der VPC-Sicherheitsgruppe, die der Datenbank zugeordnet ist, eine Regel hinzufügen.

Für die Auflistung einer Rechenressource müssen die folgenden Bedingungen erfüllt sei:

- Der Name der Sicherheitsgruppe, die der Rechenressource zugeordnet ist, entspricht dem Muster `ec2-rds-`*n* (wobei *n* für eine Zahl steht).
- Die Sicherheitsgruppe, die der Rechenressource zugeordnet ist, hat eine Regel für ausgehenden Datenverkehr, wobei der Portbereich auf den Port festgelegt ist, den der DB-Cluster verwendet.
- Die Sicherheitsgruppe, die der Rechenressource zugeordnet ist, hat eine Regel für ausgehenden Datenverkehr, wobei die Quelle auf eine Sicherheitsgruppe festgelegt ist, die dem DB-Cluster zugeordnet ist.
- Der Name der Sicherheitsgruppe, die dem DB-Cluster zugeordnet ist, entspricht dem Muster `rds-ec2-`*n* entspricht (wobei *n* für eine Zahl steht).
- Die Sicherheitsgruppe, die dem DB-Cluster zugeordnet ist, hat eine Regel für eingehenden Datenverkehr, wobei der Portbereich auf den Port festgelegt ist, den der DB-Cluster verwendet.
- Die Sicherheitsgruppe, die dem DB-Cluster zugeordnet ist, hat eine Regel für eingehenden Datenverkehr, wobei die Quelle auf eine Sicherheitsgruppe festgelegt ist, die der Rechenressource zugeordnet ist.

So zeigen Sie Rechenressourcen an, die mit einem Aurora-DB-Cluster verbunden sind

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den Namen der RDS-Datenbank aus.
3. Sehen Sie sich auf der Registerkarte Connectivity & security (Konnektivität und Sicherheit) die Rechenressourcen unter Verbundene Rechenressourcen an.



Herstellen einer Verbindung mit einer DB-Instance, die eine bestimmte DB-Engine ausführt

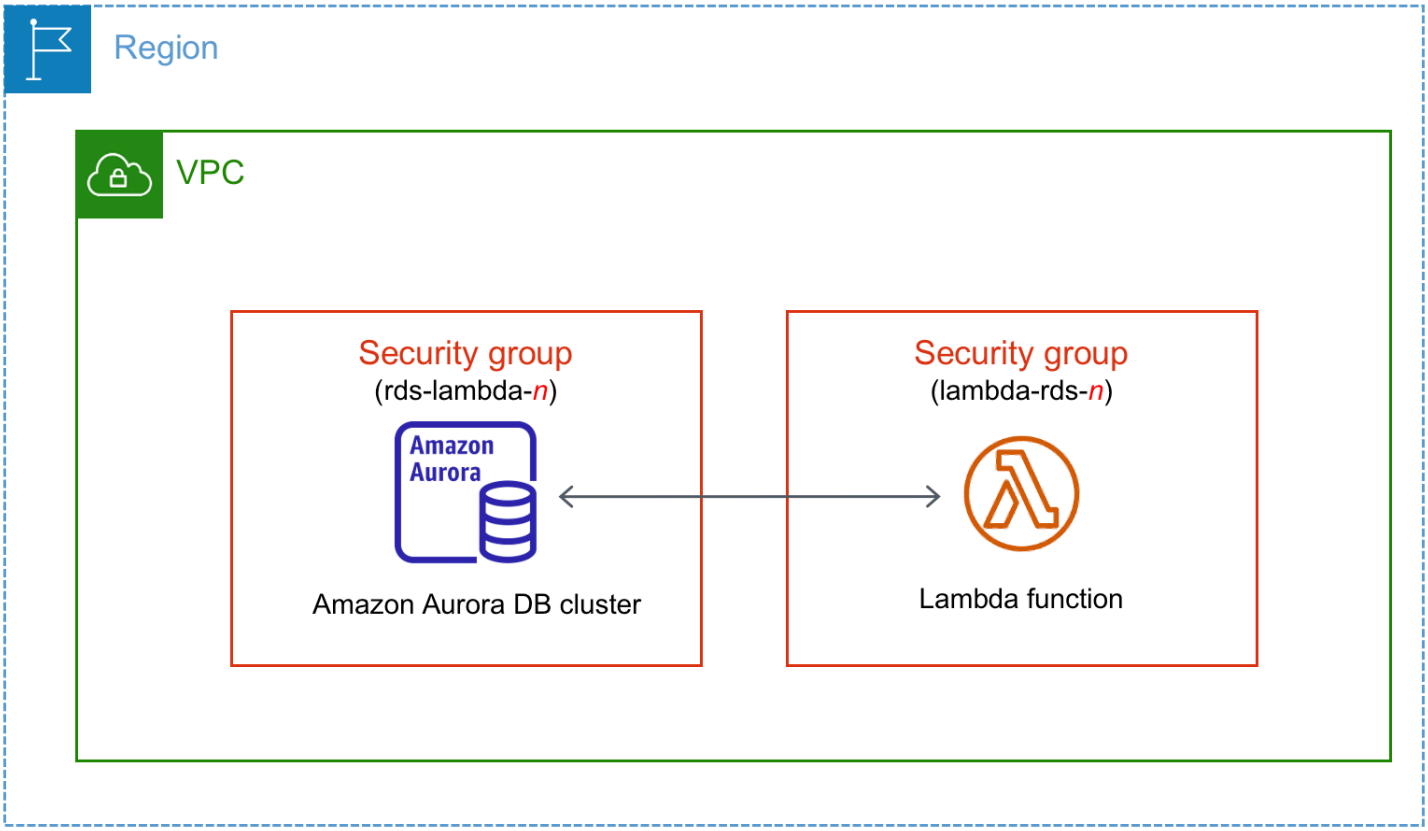
Informationen zum Herstellen einer Verbindung mit einer DB-Instance, die eine bestimmte DB-Engine ausführt, finden Sie in den Anweisungen für Ihre DB-Engine:

- Herstellen einer Verbindung mit einem Amazon Aurora MySQL-DB-Cluster
- Herstellen einer Verbindung mit einem Amazon-Aurora-PostgreSQL-DB-Cluster

Automatisches Verbinden einer Lambda-Funktion mit einem Aurora-DB-Cluster

Sie können die Amazon-RDS-Konsole verwenden, um das Einrichten einer Verbindung zwischen einer Lambda-Funktion und einer DB-Instance zu vereinfachen. Häufig befindet sich Ihr DB-Cluster in einem privaten Subnetz innerhalb einer VPC. Die Lambda-Funktion kann von Anwendungen verwendet werden, um auf Ihren privaten DB-Cluster zuzugreifen.

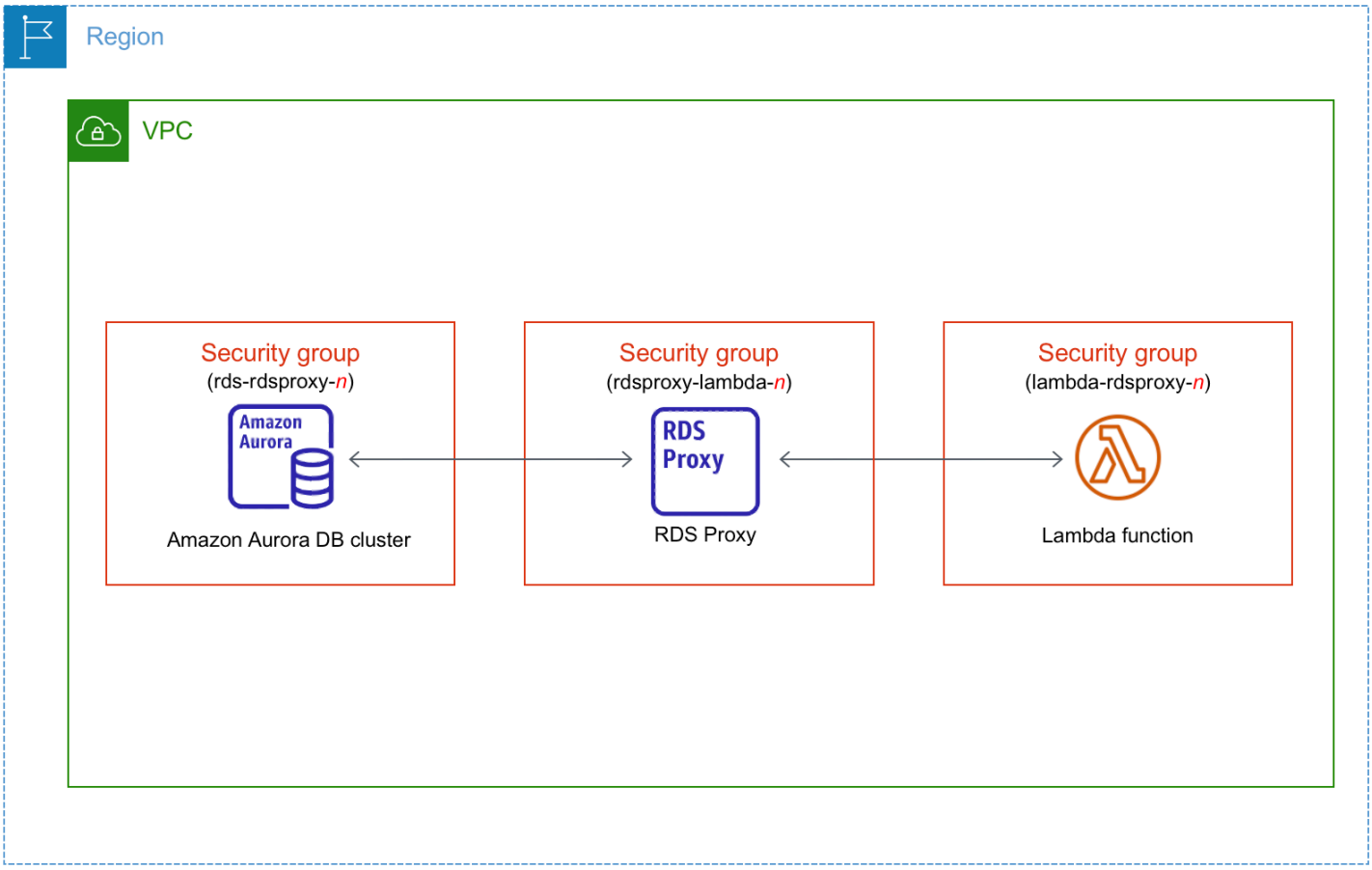
Das folgende Bild zeigt eine direkte Verbindung zwischen Ihrem DB-Cluster und Ihrer Lambda-Funktion.



Sie können die Verbindung zwischen Ihrer Lambda-Funktion und Ihrem DB-Cluster über RDS-Proxy einrichten, um die Leistung und Stabilität Ihrer Datenbank zu verbessern. Oft stellen Lambda-Funktionen häufige, kurze Datenbankverbindungen her, die von dem von RDS-Proxy angebotenen Verbindungspooling profitieren. Sie können alle AWS Identity and Access Management (IAM)-Authentifizierungen nutzen, die Sie bereits für Lambda-Funktionen haben, anstatt Datenbankanmeldeinformationen im Lambda-Anwendungscode zu verwalten. Weitere Informationen finden Sie unter Amazon-RDS-Proxy für Aurora.

Wenn Sie die Konsole verwenden, um eine Verbindung mit einem vorhandenen Proxy herzustellen, aktualisiert Amazon RDS die Proxy-Sicherheitsgruppe, um Verbindungen von Ihrem DB-Cluster und der Lambda-Funktion zuzulassen.

Sie können auf dieser Konsolenseite auch einen neuen Proxy erstellen. Wenn Sie in der Konsole einen Proxy erstellen, um auf den DB-Cluster zuzugreifen, müssen Sie Ihre Datenbankanmeldeinformationen eingeben oder ein Secret von AWS Secrets Manager auswählen.



Tip

Um eine Lambda-Funktion schnell mit einem Aurora-DB-Cluster zu verbinden, können Sie auch den Assistenten verwenden, der in die Konsole integriert ist. Gehen Sie wie folgt vor, um den Assistenten zu öffnen:

1. Öffnen Sie die Seite Funktionen der Lambda-Konsole.
2. Wählen Sie die Funktion aus, mit der Sie eine Datenbank verbinden möchten.
3. Wählen Sie auf der Registerkarte Konfiguration die Option RDS-Datenbanken aus.
4. Wählen Sie Verbindung zur RDS-Datenbank herstellen aus.

Nachdem Sie Ihre Funktion mit einer Datenbank verbunden haben, können Sie über die Option Proxy hinzufügen einen Proxy erstellen.

Themen

- Überblick über die automatische Konnektivität mit einer Lambda-Funktion
- Automatisches Verbinden einer Lambda-Funktion und eines Aurora-DB-Clusters
- Anzeigen verbundener Rechenressourcen

Überblick über die automatische Konnektivität mit einer Lambda-Funktion

Im Folgenden sind Anforderungen für die Verbindung einer Lambda-Funktion mit einem Aurora-DB-Cluster aufgeführt:

- Die Lambda-Funktion muss sich in derselben VPC befinden wie der DB-Cluster.
- Derzeit kann der DB-Cluster kein DB-Cluster von Aurora Serverless oder Teil einer globalen Aurora-Datenbank sein.
- Der Benutzer, der die Verbindung einrichtet, muss über Berechtigungen zum Ausführen der folgenden Vorgänge von Amazon RDS, Amazon EC2, Lambda, Secrets Manager und IAM verfügen:
  - Amazon RDS
    - `rds:CreateDBProxies`
    - `rds:DescribeDBClusters`
    - `rds:DescribeDBProxies`
    - `rds:ModifyDBCluster`
    - `rds:ModifyDBProxy`
    - `rds:RegisterProxyTargets`
  - Amazon EC2
    - `ec2:AuthorizeSecurityGroupEgress`
    - `ec2:AuthorizeSecurityGroupIngress`
    - `ec2:CreateSecurityGroup`
    - `ec2:DeleteSecurityGroup`

    - `ec2:DescribeSecurityGroups`
    - `ec2:RevokeSecurityGroupEgress`
    - `ec2:RevokeSecurityGroupIngress`
- Lambda
    - `lambda:CreateFunctions`
    - `lambda:ListFunctions`
    - `lambda:UpdateFunctionConfiguration`
- Secrets Manager
    - `secretsmanager:CreateSecret`
    - `secretsmanager:DescribeSecret`
- IAM
    - `iam:AttachPolicy`
    - `iam:CreateRole`
    - `iam:CreatePolicy`
- AWS KMS
    - `kms:describeKey`

> **Note**
>
> Wenn sich der DB-Cluster und die Lambda-Funktion in unterschiedlichen Availability Zones befinden, fallen auf Ihrem Konto ggf. Kosten über Availability Zones hinweg an.

Wenn Sie eine Verbindung zwischen einer Lambda-Funktion und einem Aurora-DB-Cluster einrichten, konfiguriert Amazon RDS die VPC-Sicherheitsgruppe für Ihre Funktion und für Ihren DB-Cluster automatisch. Wenn Sie RDS-Proxy verwenden, konfiguriert Amazon RDS auch die VPC-Sicherheitsgruppe für den Proxy. Amazon RDS handelt gemäß der aktuellen Konfiguration der Sicherheitsgruppen, die dem DB-Cluster, der Lambda-Funktion und dem Proxy zugeordnet sind, wie in der folgenden Tabelle beschrieben.

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Es gibt eine oder mehrere Sicherheitsgruppen, die dem DB-Cluster mit einem Namen zugeordnet sind, der dem Muster `rds-lambda-` *n* entspricht. Wenn bereits ein Proxy mit Ihrem DB-Cluster verbunden ist, prüft RDS, ob `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der Lambda-Funktion oder dem Proxy als Quelle.	Der Lambda-Funktion sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht (wobei *n* für eine Zahl steht). Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe hat nur eine Regel für ausgehenden Datenverkehr, wobei entweder die VPC-Sicherheitsgruppe des DB-Clusters oder der Proxy das Ziel ist.	Dem Proxy sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rdsproxy-lambda-` *n* entspricht (wobei *n* für eine Zahl steht). Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe verfügt über Regeln für ein- und ausgehenden Datenverkehr mit den VPC-Sicherheitsgruppen der Lambda-Funktion und des DB-Clusters.	Amazon RDS führt keine Aktion aus. Es wurde bereits automatisch eine Verbindung zwischen der Lambda-Funktion, dem Proxy (optional) und dem DB-Cluster konfiguriert. Da bereits eine Verbindung zwischen der Funktion, dem Proxy und der Datenbank besteht, werden die Sicherheitsgruppen nicht geändert.

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Es gilt eine der folgenden Bedingungen: • Dem DB-Cluster ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. • Eine oder mehrere Sicherheitsgruppen sind dem DB-Cluster zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. Keine dieser Sicherheitsgruppen kann jedoch für die Verbindung mit der	Es gilt eine der folgenden Bedingungen: • Der Lambda-Funktion ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht. • Der Lambda-Funktion sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster verwenden.	Es gilt eine der folgenden Bedingungen: • Dem Proxy ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `rdsproxy-lambda-` *n* entspricht. • Dem Proxy sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rdsproxy-lambda-` *n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster oder der Lambda-Funktion verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn	RDS action: create new security groups

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Lambda-Funktion verwendet werden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn sie keine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der Lambda-Funktion oder dem Proxy als Quelle enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde. Beispiele für Änderungen sind das Hinzufügen einer Regel oder das Ändern des Ports einer vorhandenen Regel.	Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn diese keine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Ziel enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	diese keine Regel für ein- und ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters oder der Lambda-Funktion enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Eine oder mehrere Sicherheitsgruppen sind dem DB-Cluster zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der Lambda-Funktion oder dem Proxy als Quelle.	Der Lambda-Funktion sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn diese keine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters als Ziel enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	Dem Proxy sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rdsproxy-lambda-` *n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster oder der Lambda-Funktion verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn diese keine Regel für ein- und ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters oder der Lambda-Funktion enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	RDS action: create new security groups

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Eine oder mehrere Sicherheitsgruppen sind dem DB-Cluster zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der Lambda-Funktion oder dem Proxy als Quelle.	Eine gültige Lambda-Sicherheitsgruppe für die Verbindung ist vorhanden, jedoch nicht mit der Lambda-Funktion verknüpft. Die Sicherheitsgruppe hat einen Namen, der dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht. Sie wurde nicht geändert. Sie hat nur eine Regel für ausgehenden Datenverkehr, wobei die VPC-Sicherheitsgruppe des DB-Clusters oder der Proxy das Ziel ist.	Eine gültige Proxy-Sicherheitsgruppe für die Verbindung ist vorhanden, jedoch nicht mit dem Proxy verknüpft. Die Sicherheitsgruppe trägt einen Namen, der dem Muster `rdsproxy-lambda-` *n* entspricht. Sie wurde nicht geändert. Sie verfügt über Regeln für ein- und ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters und der Lambda-Funktion.	RDS action: associate Lambda security group

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Es gilt eine der folgenden Bedingungen: • Dem DB-Cluster ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. • Eine oder mehrere Sicherheitsgruppen sind dem DB-Cluster zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit der Lambda-	Der Lambda-Funktion sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe enthält nur eine Regel für ausgehenden Datenverkehr, wobei entweder die VPC-Sicherheitsgruppe des DB-Clusters oder der Proxy das Ziel ist.	Dem Proxy sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rdsproxy-lambda-` *n* entspricht. Eine Sicherheitsgruppe, die dem Muster entspricht, wurde nicht geändert. Diese Sicherheitsgruppe verfügt über Regeln für ein- und ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters und der Lambda-Funktion.	RDS action: create new security groups

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Funktion oder dem Proxy verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn sie keine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der Lambda-Funktion oder dem Proxy als Quelle enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.			

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Es gilt eine der folgenden Bedingungen: • Dem DB-Cluster ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. • Eine oder mehrere Sicherheitsgruppen sind dem DB-Cluster zugeordnet, deren Name dem Muster `rds-lambda-` *n* entspricht, oder wenn `TargetHealth` eines zugehörigen Proxys `AVAILABLE` ist. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit der Lambda-	Es gilt eine der folgenden Bedingungen: • Der Lambda-Funktion ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht. • Der Lambda-Funktion sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `lambda-rds-` *n* oder `lambda-rdsproxy-` *n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster verwenden.	Es gilt eine der folgenden Bedingungen: • Dem Proxy ist keine Sicherheitsgruppe zugeordnet, deren Name dem Muster `rdsproxy-lambda-` *n* entspricht. • Dem Proxy sind eine oder mehrere Sicherheitsgruppen zugeordnet, deren Name dem Muster `rdsproxy-lambda-` *n* entspricht. Amazon RDS kann jedoch keine dieser Sicherheitsgruppen für die Verbindung mit dem DB-Cluster oder der Lambda-Funktion verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn	RDS action: create new security groups

Aktuelle RDS-Sicherheitsgruppenkonfiguration	Aktuelle Konfiguration der Lambda-Sicherheitsgruppe	Aktuelle Konfiguration der Proxy-Sicherheitsgruppe	RDS-Aktion
Funktion oder dem Proxy verwenden. Amazon RDS kann eine Sicherheitsgruppe nicht verwenden, wenn sie keine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der Lambda-Funktion oder dem Proxy als Quelle enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	Eine Sicherheitsgruppe kann nicht verwendet werden, wenn sie keine Regel für ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters oder dem Proxy als Quelle enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	diese keine Regel für ein- und ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters oder der Lambda-Funktion enthält. Amazon RDS kann auch keine Sicherheitsgruppe verwenden, die geändert wurde.	

RDS-Aktion : neue Sicherheitsgruppen erstellen

Amazon RDS führt die folgenden Aktionen durch:

- Erstellt eine neue Sicherheitsgruppe, die dem Muster `rds-lambda-`*n* oder `rds-rdsproxy-`*n* entspricht (wenn Sie RDS-Proxy verwenden). Diese Sicherheitsgruppe enthält eine Regel für eingehenden Datenverkehr mit der VPC-Sicherheitsgruppe der Lambda-Funktion oder dem Proxy als Quelle. Diese Sicherheitsgruppe ist dem DB-Cluster zugeordnet und ermöglicht der Funktion oder dem Proxy den Zugriff auf den DB-Cluster.
- Erstellt eine neue Sicherheitsgruppe, die dem Muster `lambda-rds-`*n* oder `lambda-rdsproxy-`*n* entspricht. Diese Sicherheitsgruppe enthält eine Regel für ausgehenden Datenverkehr, wobei entweder die VPC-Sicherheitsgruppe des DB-Clusters oder der Proxy das

Ziel ist. Diese Sicherheitsgruppe ist der Lambda-Funktion zugeordnet und ermöglicht es der Funktion, Datenverkehr an den DB-Cluster zu senden oder Datenverkehr über einen Proxy zu senden.

- Erstellt eine neue Sicherheitsgruppe, die dem Muster `rdsproxy-lambda-`*n* entspricht. Diese Sicherheitsgruppe verfügt über Regeln für ein- und ausgehenden Datenverkehr mit der VPC-Sicherheitsgruppe des DB-Clusters und der Lambda-Funktion.

RDS-Aktion : Lambda-Sicherheitsgruppe zuordnen

Amazon RDS ordnet die gültige, vorhandene Lambda-Sicherheitsgruppe der Lambda-Funktion zu. Diese Sicherheitsgruppe ermöglicht es der Funktion, Datenverkehr an den DB-Cluster zu senden oder Datenverkehr über einen Proxy zu senden.

Automatisches Verbinden einer Lambda-Funktion und eines Aurora-DB-Clusters

Sie können die Amazon-RDS-Konsole verwenden, um eine Lambda-Funktion automatisch mit Ihrem DB-Cluster zu verbinden. Dies vereinfacht das Einrichten einer Verbindung zwischen diesen Ressourcen.

Sie können den RDS-Proxy auch verwenden, um einen Proxy in Ihre Verbindung aufzunehmen. Lambda-Funktionen stellen häufige, kurze Datenbankverbindungen her, die von dem von RDS-Proxy angebotenen Verbindungspooling profitieren. Sie können auch eine IAM-Authentifizierung nutzen, die Sie bereits für Lambda-Funktionen eingerichtet haben, anstatt Datenbankanmeldeinformationen im Lambda-Anwendungscode zu verwalten.

Sie können einen vorhandenen DB-Cluster mit neuen oder bestehenden Lambda-Funktionen unter Verwendung der Seite Lambda-Verbindung einrichtenverbinden. Beim Einrichtungsvorgang werden automatisch die erforderlichen Sicherheitsgruppen für Sie eingerichtet.

Vor dem Einrichten einer Verbindung zwischen einer Lambda-Funktion und einem DB-Cluster stellen Sie Folgendes sicher:

- Ihre Lambda-Funktion und Ihr DB-Cluster befinden sich in derselben VPC.
- Sie verfügen über die richtigen Berechtigungen für Ihr Benutzerkonto. Weitere Informationen zu den Anforderungen finden Sie unter Überblick über die automatische Konnektivität mit einer Lambda-Funktion.

Wenn Sie Sicherheitsgruppen nach dem Konfigurieren der Verbindung ändern, können sich die Änderungen auf die Verbindung zwischen der Lambda-Funktion und dem DB-Cluster auswirken.

> **Note**
>
> Sie können eine Verbindung zwischen einem DB-Cluster und einer Lambda-Funktion nur in der AWS-Managementkonsole automatisch einrichten. Zum Herstellen einer Verbindung mit einer Lambda-Funktion müssen sich alle Instances im DB-Cluster im Status Verfügbar befinden.

So verbinden Sie eine Lambda-Funktion automatisch mit einem DB-Cluster

<result>

Nachdem Sie die Einrichtung bestätigt haben, beginnt Amazon RDS mit dem Herstellen der Verbindung Ihrer Lambda-Funktion, Ihres RDS-Proxys (falls Sie einen Proxy verwendet haben) und Ihres DB-Clusters. Die Konsole zeigt das Dialogfeld Verbindungsdetails an, in dem die Änderungen der Sicherheitsgruppe aufgeführt sind, die Verbindungen zwischen Ihren Ressourcen ermöglichen.
</result>

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken und dann den DB-Cluster zum Verbinden mit einer Lambda-Funktion aus.
3. Wählen Sie für Aktionen die Option Lambda-Verbindung einrichten aus.
4. Führen Sie auf der Seite Lambda-Verbindung einrichten unter Lambda-Funktion auswählen einen der folgenden Schritte aus:
   - Wenn eine Lambda-Funktion in derselben VPC wie Ihr DB-Cluster vorhanden ist, wählen Sie Vorhandene Funktion auswählen aus und wählen Sie dann die Funktion aus.
   - Wenn Sie keine Lambda-Funktion in derselben VPC vorhanden ist, wählen Sie Neue Funktion erstellen aus und geben Sie dann einen Namen im Feld Funktionsname ein. Die Standard-Laufzeit ist auf Nodejs.18 festgelegt. Sie können die Einstellungen für Ihre neue Lambda-Funktion in der Lambda-Konsole ändern, nachdem Sie die Verbindungseinrichtung abgeschlossen haben.

5. (Optional) Wählen Sie unter RDS-Proxy die Option Über RDS-Proxy verbinden aus und führen Sie dann einen der folgenden Schritte aus:
   - Wenn Sie einen vorhandenen Proxy haben, den Sie verwenden möchten, klicken Sie auf Vorhandenen Proxy auswählen und wählen Sie dann den Proxy aus.
   - Wenn Sie keinen Proxy haben und möchten, dass Amazon RDS automatisch einen für Sie erstellt, wählen Sie Neuen Proxy erstellen aus. Führen Sie dann für Datenbankanmeldeinformationen einen der folgenden Schritte aus:

     a. Wählen Sie Datenbankbenutzername und Passwort aus und geben Sie dann den Benutzernamen und das Passwort für Ihren DB-Cluster ein.

     b. Wählen Sie Secrets-Manager-Secret aus. Wählen Sie dann unter Secret auswählen ein Secret von AWS Secrets Manager aus. Wenn Sie kein Secrets-Manager-Secret haben, wählen Sie Neues Secrets-Manager-Secret erstellen aus, um ein neues Secret zu erstellen. Nachdem Sie das Secret erstellt haben, wählen Sie das neue Secret unter Secret auswählen aus.

     Nachdem Sie den neuen Proxy erstellt haben, wählen Sie Vorhandenen Proxy auswählen aus und wählen Sie dann den Proxy aus. Beachten Sie, dass es einige Zeit dauern kann, bis Ihr Proxy für die Verbindung verfügbar ist.
6. (Optional) Erweitern Sie die Verbindungsübersicht und überprüfen Sie die hervorgehobenen Updates für Ihre Ressourcen.
7. Wählen Sie Set up (Festlegen).

Anzeigen verbundener Rechenressourcen

Sie können die AWS-Managementkonsole verwenden, um die Lambda-Funktionen anzuzeigen, die mit Ihrem DB-Cluster verbunden sind. Zu den angezeigten Ressourcen gehören Rechenressourcenverbindungen, die von Amazon RDS automatisch eingerichtet wurden.

Die aufgelisteten Rechenressourcen umfassen keine Ressourcen, die manuell mit dem DB-Cluster verbunden sind. Sie können beispielsweise einer Rechenressource den manuellen Zugriff auf Ihren DB-Cluster erlauben, indem Sie der VPC-Sicherheitsgruppe, die der Datenbank zugeordnet ist, eine Regel hinzufügen.

Damit die Konsole eine Lambda-Funktion auflistet, müssen die folgenden Bedingungen gelten:

- Der Name der Sicherheitsgruppe, die der Rechenressource zugeordnet ist, entspricht dem Muster `lambda-rds-`*n* oder `lambda-rdsproxy-`*n* (wobei *n* für eine Zahl steht).
- Die Sicherheitsgruppe, die der Rechenressource zugeordnet ist, hat eine Regel für ausgehenden Datenverkehr, wobei der Portbereich auf den Port des DB-Clusters oder einen zugeordneten Proxy festgelegt ist. Das Ziel für die Regel für ausgehende Nachrichten muss auf eine dem DB-Cluster zugeordneten Sicherheitsgruppe oder auf einen zugeordneten Proxy festgelegt werden.
- Wenn die Konfiguration einen Proxy enthält, entspricht der Name der Sicherheitsgruppe, die an den mit Ihrer Datenbank verknüpften Proxy angefügt ist, dem Muster `rdsproxy-lambda-`*n* (wobei *n* für eine Zahl steht).
- Die Sicherheitsgruppe, die der Funktion zugeordnet ist, hat eine Regel für ausgehenden Datenverkehr, wobei der Port auf den Port festgelegt ist, den der DB-Cluster oder ein zugeordneter Proxy verwendet. Das Ziel muss auf eine dem DB-Cluster zugeordneten Sicherheitsgruppe oder auf einen zugeordneten Proxy festgelegt werden.

So zeigen Sie Rechenressourcen an, die automatisch mit einem DB-Cluster verbunden sind

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken und dann den DB-Cluster aus.
3. Sehen Sie sich auf der Registerkarte Konnektivität und Sicherheit die Rechenressourcen unter Verbundene Rechenressourcen an.

Ändern eines Amazon Aurora-DB-Clusters

Sie können die Einstellungen eines DB-Clusters ändern, um Aufgaben durchzuführen, beispielsweise das Ändern des Aufbewahrungszeitraums für Backups oder des Datenbankports. Außerdem können Sie DB-Instances in einem DB-Cluster ändern, um Aufgaben durchzuführen, beispielsweise das Ändern der DB-Instance-Klasse oder das Aktivieren von Performance-Insights für diese. Dieses Thema führt Sie durch die Schritte zum Ändern eines Aurora-DB-Instance-Clusters und seiner DB-Instances und beschreibt die jeweiligen Einstellungen.

Wir empfehlen, alle Änderungen auf einem Test-DB-Cluster oder einer Test-DB-Instance zu testen, bevor Sie einen Produktions-DB-Cluster oder eine Produktions-DB-Instance ändern, damit Sie die Auswirkungen jeder Änderung detailliert nachvollziehen können. Dies ist besonders bei Upgrades der Datenbankversionen wichtig.

Themen

- Ändern des DB-Clusters über die Konsole, die CLI und die API
- Ändern einer DB-Instance in einem DB-Cluster
- Ändern des Passworts für den Master-Benutzer der Datenbank
- Einstellungen für Amazon Aurora
- Einstellungen, die nicht für Amazon-Aurora-DB-Cluster gelten
- Einstellungen, die nicht für Amazon-Aurora-DB-Instances gelten

Ändern des DB-Clusters über die Konsole, die CLI und die API

Sie können einen DB-Cluster über die AWS-Managementkonsole, die AWS CLI oder die RDS-API ändern.

> **Note**
>
> Die meisten Änderungen können sofort oder während des nächsten Wartungszeitraums angewendet werden. Einige Änderungen, z. B. das Aktivieren des Löschschutzes, werden sofort umgesetzt – unabhängig davon, wann Sie sie anwenden.
> Änderungen des Hauptpassworts in der AWS-Managementkonsole werden immer sofort angewendet.

Wenn Sie SSL-Endpunkte verwenden und die DB-Cluster-ID ändern, beenden Sie den DB-Cluster und starten Sie ihn neu, um die SSL-Endpunkte zu aktualisieren. Weitere Informationen finden Sie unter Stoppen und Starten eines Amazon Aurora-DB-Clusters.

Konsole

So ändern Sie einen DB-Cluster:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den zu ändernden DB-Cluster aus.
3. Wählen Sie Ändern aus. Die Seite DB-Cluster ändern wird angezeigt.
4. Ändern Sie alle Einstellungen nach Bedarf. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Amazon Aurora.

   > **Note**
   >
   > In der AWS-Managementkonsole werden einige Änderungen auf Instance-Ebene nur auf die aktuelle DB-Instance angewendet, während andere auf das gesamte DB-Cluster angewendet werden. Informationen dazu, wann eine Einstellung auf die DB-Instance oder das DB-Cluster angewendet wird, finden Sie im Abschnitt zum Geltungsbereich der Einstellung in Einstellungen für Amazon Aurora. Um eine Einstellung zu ändern, die das gesamte DB-Cluster auf Instance-Ebene in der AWS-Managementkonsole ändert, befolgen Sie die Anweisungen in Ändern einer DB-Instance in einem DB-Cluster.

5. Nachdem Sie die gewünschten Änderungen vorgenommen haben, wählen Sie Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
6. Klicken Sie auf Apply immediately, um die Änderungen sofort anzuwenden.
7. Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen. Wenn sie korrekt sind, wählen Sie Modify cluster (Cluster ändern) aus, um Ihre Änderungen zu speichern.

   Klicken Sie anderenfalls auf Zurück, um Ihre Änderungen zu bearbeiten, oder klicken Sie auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

Um ein DB-Cluster über die AWS CLI zu ändern, rufen Sie den Befehl modify-db-cluster auf. Geben Sie die DB-Cluster-Kennung und die Werte für die Einstellungen an, die geändert werden sollen. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Amazon Aurora.

> **Note**
>
> Einige Einstellungen werden nur auf DB-Instances angewendet. Um diese Einstellungen zu ändern, folgen Sie den Anweisungen in Ändern einer DB-Instance in einem DB-Cluster.

Example

Der folgende Befehl ändert `mydbcluster`, indem der Aufbewahrungszeitraum für Backups auf 1 Woche (7 Tage) festgelegt wird.

Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --backup-retention-period 7
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 7
```

RDS-API

Um einen DB-Cluster über die Amazon-RDS-API zu ändern, rufen Sie die Aktion ModifyDBCluster auf. Geben Sie die DB-Cluster-Kennung und die Werte für die Einstellungen an, die geändert werden sollen. Weitere Informationen zu den einzelnen Parametern finden Sie unter Einstellungen für Amazon Aurora.

> **Note**
>
> Einige Einstellungen werden nur auf DB-Instances angewendet. Um diese Einstellungen zu ändern, befolgen Sie die Anleitungen in Ändern einer DB-Instance in einem DB-Cluster.

Ändern einer DB-Instance in einem DB-Cluster

Sie können eine DB-Instance in einem DB-Cluster über die AWS-Managementkonsole, die AWS CLI oder die RDS-API ändern.

Wenn Sie eine DB-Instance ändern, können Sie die Änderungen sofort anwenden. Um Änderungen sofort anzuwenden, können Sie in der AWS-Managementkonsole die Option Apply Immediately (Sofort anwenden) auswählen, die AWS CLI mit dem Parameter `--apply-immediately` aufrufen oder, wenn Sie die Amazon-RDS-API verwenden, für den Parameter `ApplyImmediately` den Wert `true` festlegen.

Wenn Sie sich nicht dafür entscheiden, Änderungen sofort anzuwenden, werden die Änderungen auf das nächste Wartungsfenster verschoben. Während des nächsten Wartungsfensters werden alle diese verzögerten Änderungen angewendet. Wenn Sie Änderungen sofort anwenden, werden Ihre neuen Änderungen und alle zuvor verzögerten Änderungen angewendet.

Verwenden Sie zum Anzeigen der Änderungen, die für das nächste Wartungsfenster noch ausstehen, den AWS CLI-Befehl describe-db-clusters und überprüfen Sie das Feld `PendingModifiedValues`.

> **Important**
>
> Wenn ausstehende Änderungen eine Ausfallzeit erfordern, kann die Auswahl von Apply immediately einen unerwarteten Ausfall für die DB-Instance verursachen. Es gibt keine Ausfallzeiten für die anderen DB-Instances im DB-Cluster.
> Änderungen, die Sie verschieben, sind in der Ausgabe des `describe-pending-maintenance-actions`-CLI-Befehls nicht aufgeführt. Wartungsmaßnahmen umfassen nur System-Upgrades, die Sie für das nächste Wartungsfenster planen.

Konsole

So ändern Sie eine DB-Instance in einem DB-Cluster:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die zu ändernde DB-Instance aus.
3. Wählen Sie für Actions (Aktionen) die Option Modify (Ändern) aus. Die Seite Modify DB instance (DB-Instance ändern) wird angezeigt.
4. Ändern Sie alle Einstellungen nach Bedarf. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Amazon Aurora.

   > **Note**
   >
   > Einige Einstellungen gelten für das gesamte DB-Cluster und müssen auf Cluster-Ebene geändert werden. Um diese Einstellungen zu ändern, folgen Sie den Anweisungen in Ändern des DB-Clusters über die Konsole, die CLI und die API.
   > In der AWS-Managementkonsole werden einige Änderungen auf Instance-Ebene nur auf die aktuelle DB-Instance angewendet, während andere auf das gesamte DB-Cluster angewendet werden. Informationen dazu, wann eine Einstellung auf die DB-Instance oder das DB-Cluster angewendet wird, finden Sie im Abschnitt zum Geltungsbereich der Einstellung in Einstellungen für Amazon Aurora.

5. Nachdem Sie die gewünschten Änderungen vorgenommen haben, wählen Sie Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
6. Klicken Sie auf Apply immediately, um die Änderungen sofort anzuwenden.
7. Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen. Wenn sie korrekt sind, wählen Sie Modify DB Instance (DB-Instance ändern) aus, um Ihre Änderungen zu speichern.

   Klicken Sie anderenfalls auf Zurück, um Ihre Änderungen zu bearbeiten, oder klicken Sie auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

Um eine DB-Instance in einem DB-Cluster über die AWS CLI zu ändern, rufen Sie den Befehl modify-db-instance auf. Geben Sie die DB-Instance-Kennung und die Werte für die Einstellungen an, die

geändert werden sollen. Weitere Informationen zu den einzelnen Parametern finden Sie unter Einstellungen für Amazon Aurora.

> **Note**
>
> Einige Einstellungen werden auf das gesamte DB-Cluster angewendet. Um diese Einstellungen zu ändern, folgen Sie den Anweisungen in Ändern des DB-Clusters über die Konsole, die CLI und die API.

Example

Der folgende Code ändert `mydbinstance`, indem die Klasse der DB-Instance in `db.r4.xlarge` geändert wird. Die Änderungen werden während des nächsten Wartungsfensters (mit) übernomme `--no-apply-immediately`. Verwenden Sie `--apply-immediately`, damit Änderungen sofort angewendet werden.

Für Linux, macOS oder Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r4.xlarge \
    --no-apply-immediately
```

Für Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.r4.xlarge ^
    --no-apply-immediately
```

RDS-API

Rufen Sie die Operation ModifyDBInstance auf, um eine DB-Instance mithilfe der Amazon-RDS-API zu ändern. Geben Sie die DB-Instance-Kennung und die Werte für die Einstellungen an, die geändert werden sollen. Weitere Informationen zu den einzelnen Parametern finden Sie unter Einstellungen für Amazon Aurora.

> **Note**
>
> Einige Einstellungen werden auf das gesamte DB-Cluster angewendet. Um diese Einstellungen zu ändern, befolgen Sie die Anleitungen in Ändern des DB-Clusters über die Konsole, die CLI und die API.

Ändern des Passworts für den Master-Benutzer der Datenbank

Verwenden Sie die AWS-Managementkonsole oder die AWS CLI zum Ändern des Master-Benutzerpassworts.

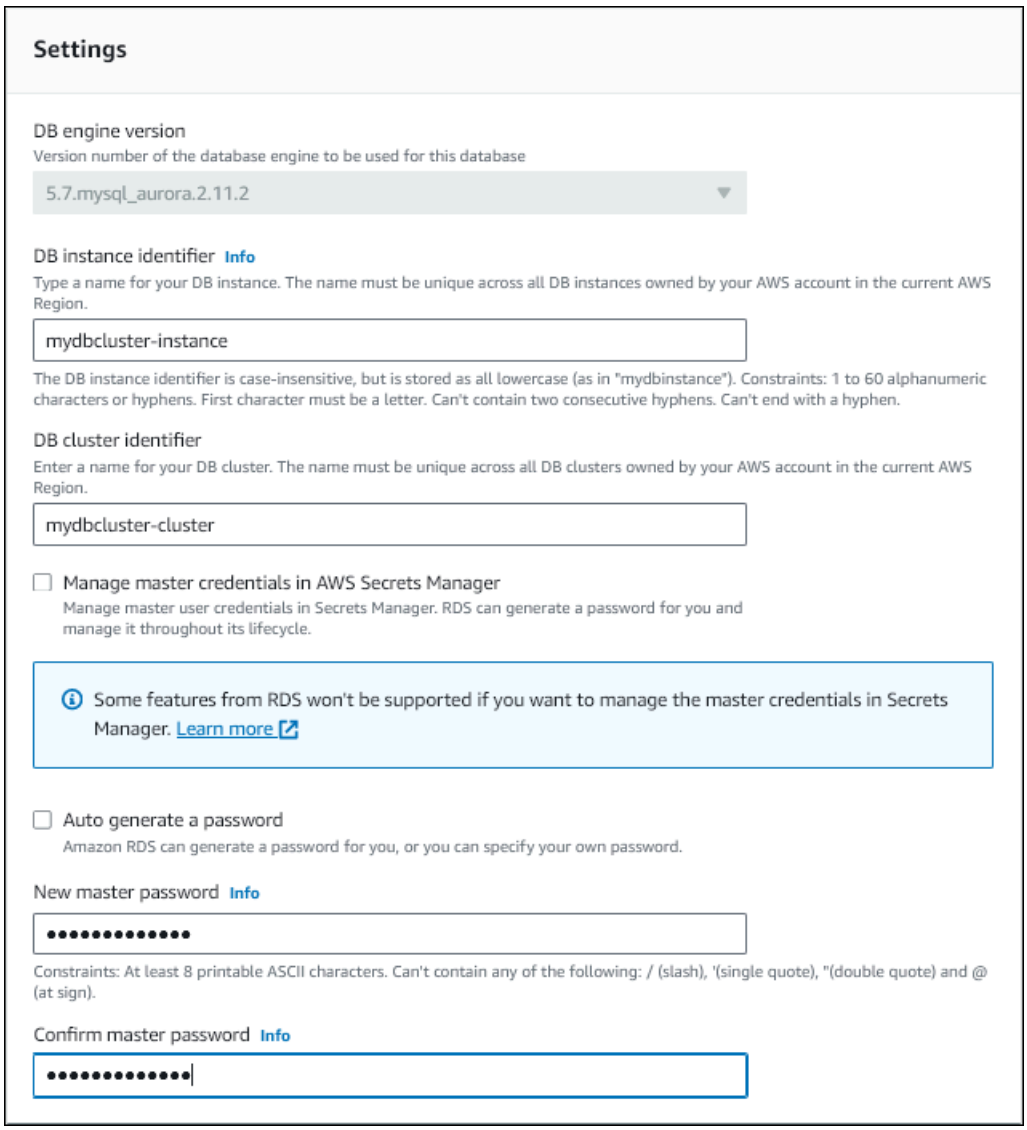
Konsole

Sie ändern die Schreiber-DB-Instance, um das Master-Benutzerpasswort mit der AWS-Managementkonsole zu ändern.

So ändern Sie das Master-Benutzerpasswort

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die zu ändernde DB-Instance aus.
3. Wählen Sie für Actions (Aktionen) die Option Modify (Ändern) aus.

   Die Seite Modify DB instance (DB-Instance ändern) wird angezeigt.
4. Geben Sie ein Neues Master-Passwort ein.
5. Geben Sie unter Master-Passwort bestätigen dasselbe neue Passwort ein.



6. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.

> **Note**
>
> Passwortänderungen werden immer sofort übernommen.

7. Klicken Sie auf der Bestätigungsseite auf Modify DB instance (DB-Instance ändern).

CLI

Sie rufen den Befehl modify-db-cluster auf, um das Master-Benutzerpasswort mit der AWS CLI zu ändern. Geben Sie die DB-Cluster-ID und das neue Passwort wie in den folgenden Beispielen an.

Sie müssen `--apply-immediately|--no-apply-immediately` nicht angeben, da Passwortänderungen immer sofort übernommen werden.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --master-user-password mynewpassword
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --master-user-password mynewpassword
```

Einstellungen für Amazon Aurora

Die folgende Tabelle enthält Details dazu, welche Einstellungen Sie ändern können, zu den Methoden, mit denen Einstellungen geändert werden, und zum Geltungsbereich der jeweiligen Einstellungen. Der Geltungsumfang legt fest, ob die Einstellung auf den gesamten DB-Cluster angewendet wird oder nur für spezifische DB-Instances festgelegt werden kann.

> **Note**
>
> Wenn Sie einen DB-Cluster von Aurora Serverless v1 oder Aurora Serverless v2 bearbeiten, stehen zusätzliche Einstellungen zur Verfügung. Weitere Informationen zu diesen Einstellungen finden Sie unter Ändern eines Aurora Serverless v1-DB-Clusters und Verwalten von Aurora Serverless v2-DB-Clustern.
> Außerdem sind einige Einstellungen für Aurora Serverless v1 und Aurora Serverless v2 aufgrund ihrer Einschränkungen nicht verfügbar. Weitere Informationen erhalten Sie unter Einschränkungen von Aurora Serverless v1 und Anforderungen und Einschränkungen für Aurora Serverless v2.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Automatischer kleiner Version-Upgrade Ob Sie möchten, dass die DB-Instance automatisch Upgrades auf bevorzugte Engine-Unterversionen erhält, wenn sie verfügbar werden. Upgrades werden nur während Ihres geplanten Wartungsfensters installiert. Weitere Informationen über Engine-Updates für finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL und Datenbank-Engine-Updates für Amazon Aurora MySQL. Weitere Informationen zur Einstellung für automatische Nebenversions-Upgrades für Aurora MySQL finden Sie unter Aktivieren von automatischen	**Note** Diese Einstellung ist standardmäßig aktiviert. Wählen Sie für jeden neuen Cluster den entsprechenden Wert für diese Einstellung basierend auf der Wichtigkeit, der erwarteten Lebensdauer und dem Umfang der Verifizierungstests, die Sie nach jedem Upgrade durchführen. Wenn Sie diese Einstellung ändern, führen Sie diese Änderung für jede DB-Instance in Ihrem	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall. Wenn Aurora automatische Upgrades durchführt, treten bei zukünftigen Wartungsfenstern Ausfälle auf.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Upgrades zwischen Aurora MySQL-Nebenversionen.	Aurora-Cluster durch. Wenn für eine DB-Instance in Ihrem Cluster diese Einstellung deaktiviert ist, wird der Cluster nicht automatisch aktualisiert. Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie modify-db-instance aus und legen Sie die Option `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` fest. Bei Verwendung der RDS-API: Rufen Sie ModifyDBInstance auf und legen Sie den Parameter `AutoMinor`		

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
	`VersionUpgrade` fest.		
Aufbewahrungszeitraum für Backups Die Anzahl der Tage, für die automatische Sicherungen aufbewahrt werden. Der Mindestwert ist 1. Weitere Informationen finden Sie unter Backups.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie modify-db-cluster aus und legen Sie die Option `--backup-retention-period` fest. Bei Verwendung der RDS-API: Rufen Sie ModifyDBCluster auf und legen Sie den Parameter `BackupRetentionPeriod` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Sicherungsfenster (Startzeit) Der Zeitraum, in dem automatisierte Sicherungen der Datenbank erstellt wird. Das Sicherungsfenster wird mit einer Startzeit (in koordinierter Weltzeit (UTC)) und einer Dauer (in Stunden) angegeben. Aurora Backups sind kontinuierlich und inkrementell. Das Backup-Fenster wird aber verwendet, um eine tägliche Systemsicherung zu erstellen, die innerhalb des Aufbewahrungszeitraums für Backups gespeichert wird. Sie können es kopieren, um es außerhalb des Aufbewahrungszeitraums zu behalten. Das Wartungsfenster und das Sicherungsfenster für den DB-	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie modify-db-cluster aus und legen Sie die Option `--preferred-backup-window` fest. Bei Verwendung der RDS-API: Rufen Sie ModifyDBCluster auf und legen Sie den Parameter `PreferredBackupWindow` fest.	Gesamter DB-Cluster.	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Cluster können sich nicht überschneiden. Weitere Informationen finden Sie unter Backup window.			
Kapazitätseinstellungen Die Skalierungseigenschaften eines DB-Clusters von Aurora Serverless v1. Sie können die Skalierungseigenschaften für DB-Cluster nur im `serverless` DB-Engine-Modus skalieren. Weitere Informationen zu Aurora Serverless v1 finden Sie unter Verwenden von Amazon Aurora Serverless v1.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie modify-db-cluster aus und legen Sie die Option `--scaling-configuration` fest. Bei Verwendung der RDS-API: Rufen Sie ModifyDBCluster auf und legen Sie den Parameter `ScalingConfiguration` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall. Die Änderung wird sofort übernommen. Diese Einstellung ignoriert die Einstellung „Apply Immediately (Sofort Anwenden)“.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Zertifizierungsstelle Die Zertifizierungsstelle (CA) für das Serverzertifikat, das von der DB-Instance verwendet wird.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie modify-db-instance aus und legen Sie die Option `--ca-certificate-identifier` fest. Bei Verwendung der RDS-API: Rufen Sie ModifyDBInstance auf und legen Sie den Parameter `CACertificateIdentifier` fest.	Nur die angegebene DB-Instance	Ein Ausfall tritt nur auf, wenn die DB-Engine keine Rotation ohne Neustart unterstützt. Sie können den AWS CLI-Befehl describe-db-engine-versions verwenden, um festzustellen, ob die DB-Engine Rotation ohne Neustart unterstützt.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Cluster-Speicherkonfiguration Der Speichertyp für den DB-Cluster: Aurora I/O-Optimized oder Aurora Standard. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie modify-db-cluster aus und legen Sie die Option `--storage-type` fest. Bei Verwendung der RDS-API: Rufen Sie ModifyDBCluster auf und legen Sie den Parameter `StorageType` fest.	Gesamter DB-Cluster	Das Ändern des Speichertyps eines DB-Clusters von Aurora PostgreSQL mit Instance-Klassen, die für Lesevorgänge optimiert sind, führt zu einem Ausfall. Dies tritt nicht auf, wenn Speichertypen für Cluster mit anderen Instance-Klassentypen geändert werden. Weitere Informationen zu den DB-Instance-Klassentypen finden Sie unter DB-Instance-Klassenarten.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Tags zu Snapshots kopieren Wählen Sie dies Option, um festzulegen, dass die für diesen DB-Cluster definierten Tags in DB-Snapshots kopiert werden sollen, die von diesem DB-Cluster erstellt wurden. Weitere Informationen finden Sie unter Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Führen Sie mithilfe der AWS CLI https://docs.aws.amazon.com/cli/latest/reference/rds/modify-db-cluster.html aus und legen Sie die Option `--copy-tags-to-snapshot` oder `--no-copy-tags-to-snapshot` fest. Bei Verwendung der RDS API: Rufen Sie https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_ModifyDBCluster.html auf und legen Sie den Parameter `CopyTagsToSnapshot` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Daten-API Sie können mit webservicebasierten Anwendungen wie AWS Lambda und AWS AppSync auf Aurora Serverless v1 zugreifen. Diese Einstellung gilt nur für Aurora Serverless v1-DB-Cluster. Weitere Informationen finden Sie unter Verwenden der Daten-API von Amazon RDS.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--enable-http-endpoint` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `EnableHttpEndpoint` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Datenbank-Authentifizierung Die Datenbank-Authentifizierung, die Sie verwenden möchten. Für MySQL: • Wählen Sie Passwortauthentifizierung aus, um Datenbankbenutzer ausschließlich mit Datenbankpasswörtern zu authentifizieren. • Wählen Sie Passwort- und IAM-Datenbank-Authentifizierung aus, um Datenbankbenutzer mit Datenbankpasswörtern und Benutzeranmeldeinformationen über IAM-Benutzer und -Rollen zu authentifizieren. Weitere Informationen finden Sie unter IAM-Daten	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Zur Verwendung von AWS CLI, starten Sie modify-db-cluster und legen Sie die folgenden Optionen fest: • Stellen Sie für die IAM-Authentifizierung die `--enable-iam-database-authentication|--no-enable-iam-database-authentication` -Option ein. • Für die Kerberos-Authentifizierung stellen Sie die `--domain-` und `--domain-iam-role-name` -Optionen ein.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
bankauthentifizierung. Für PostgreSQL: • Wählen Sie IAM database authentication (IAM-Datenbank-Authentifizierung) aus, um Datenbankbenutzer mit Datenbankpasswörtern und Benutzeranmeldeinformationen über Benutzer und Rollen zu authentifizieren. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung. • Wählen Sie Kerberos-Authentifizierung zur Authentifizierung von Datenbankkennwörtern und Benutzeranmeldeinformationen mit Kerberos-Authentifizierung. Weitere Informationen	Rufen Sie über die RDS-API ModifyDBCluster auf und setzen Sie die folgenden Parameter: • Stellen Sie für die IAM-Authentifizierung den `EnableIAMDatabaseAuthentication` -Parameter ein. • Für die Kerberos-Authentifizierung legen Sie den `Domain`- und `DomainIAMRoleName` -Parameter fest.		

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
finden Sie unter Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL.			
Datenbankport Der Port, den Sie für den Zugriff auf das DB-Cluster verwenden möchten.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--port` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `Port` fest.	Gesamter DB-Cluster	Diese Änderung verursacht einen Ausfall. Alle DB-Instances im DB-Cluster werden sofort neu gestartet.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
DB-Cluster-Kennung Die DB-Cluster-ID. Dieser Wert wird als Zeichenfolge in Kleinbuchstaben gespeichert. Wenn Sie die DB-Cluster-ID ändern, werden die DB-Cluster-Endpunkte geändert. Die Endpunkte der DB-Instances im DB-Cluster bleiben unverändert.	Bei Verwendung der AWS-Management konsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--new-db-cluster-identifier` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `NewDBClusterIdentifier` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
DB-Cluster-Parametergruppe Die DB-Clusterparametergruppe, die Sie mit dem DB-Cluster verknüpfen möchten. Weitere Informationen finden Sie unter Parametergruppen für Amazon Aurora.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--db-cluster-parameter-group-name` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `DBClusterParameterGroupName` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall. Wenn Sie die Parametergruppe ändern, werden die Änderungen an einigen Parametern ohne Neustart sofort auf die DB-Instances im DB-Cluster angewendet. Änderungen an anderen Parametern werden erst angewendet, nachdem die DB-Instances im DB-Cluster neu gestartet wurden.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
DB-Instance-Klasse Die zu verwendende DB-Instance-Klasse. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--db-instance-class` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `DBInstanceClass` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht einen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
DB-Instance-Kennung Die DB-Instance-Kennung. Dieser Wert wird als Zeichenfolge in Kleinbuchstaben gespeichert.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--new-db-instance-identifier` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `NewDBInstanceIdentifier` fest.	Nur die angegebene DB-Instance	Während dieser Änderung treten Ausfallzeiten auf. RDS startet die DB-Instance neu, um Folgendes zu aktualisieren: • Aurora MySQL – die Spalte `SERVER_ID` in der Tabelle `information_schema.replica_host_status` • Aurora PostgreSQL – die Spalte `server_id` in der Funktion `aurora_replica_status()`

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
DB-Parametergruppe Die DB-Parametergruppe, die Sie mit der DB-Instance verknüpfen möchten. Weitere Informationen finden Sie unter Parametergruppen für Amazon Aurora.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--db-parameter-group-name` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `DBParameterGroupName` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht keinen Ausfall. Wenn Sie eine neue DB-Parametergruppe mit einer DB-Instance verknüpfen, werden die geänderten statischen und dynamischen Parameter erst nach Neustart der DB-Instance angewendet. Wenn Sie jedoch dynamische Parameter in der DB-Parametergruppe ändern, nachdem Sie sie der DB-Instance zugeordnet haben, werden diese Änderungen sofort ohne Neustart angewendet. Weitere Informationen erhalten Sie unter Parametergruppen für Amazon Aurora und Neustart eines Amazon Aurora DB-Clusters oder einer

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
			Amazon Aurora DB-Instance.
Löschschutz Um zu verhindern, dass Ihr DB-Cluster gelöscht wird, können Sie die Option Löschschutz aktivieren aktivieren. Weitere Informationen finden Sie unter Löschschutz für Aurora-DB-Cluster.	Bei Verwendung der AWS-Management konsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--deletion-protection|--no-deletion-protection` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `DeletionProtection` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Engine-Version Die Versionsnummer der DB-Engine, die Sie verwenden möchten. Bevor Sie ein Upgrade für Ihr Produktions-DB-Cluster ausführen, sollten Sie den Upgrade-Prozess auf einer DB-Test-Instance testen, um die Dauer zu prüfen und Ihre Anwendungen zu validieren.	Bei Verwendung der AWS-Management konsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--engine-version` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `EngineVersion` fest.	Gesamter DB-Cluster	Diese Änderung verursacht einen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
„Enhanced Monitoring“ (Verbesserte Überwachung Wählen Sie Erweiterte Überwachung aktivieren, um Metriken in Echtzeit für das Betriebssystem zu erhalten, in dem Ihre DB-Instance ausgeführt wird. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` und legen Sie die Optionen `--monitoring-role-arn` und `--monitoring-interval` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie die Parameter `MonitoringRoleArn` und `MonitoringInterval` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Protokollexporte Wählen Sie die Protokolltypen aus , die zu Amazon CloudWatch Logs veröffentlicht werden sollen. Weitere Informationen finden Sie unter Aurora-MySQL-Datenbank-Protokolldateien.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--cloudwatch-logs-export-configuration` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `CloudwatchLogsExportConfiguration` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Wartungsfenster Der Zeitraum, in dem die Systemwartung durchgeführt wird. Die Systemwartung umfasst auch Upgrades (sofern verfügbar). Das Wartungsfenster wird mit einer Startzeit (in koordinierter Weltzeit (UTC)) und einer Dauer (in Stunden) angegeben. Falls Sie das Fenster auf die aktuelle Zeit einstellen, müssen mindestens 30 Minuten zwischen der aktuellen Zeit und der Endzeit des Fensters liegen, um sicherzustellen, dass alle ausstehenden Änderungen übernommen werden. Sie können die Wartungsfenster für das DB-Cluster und die einzelnen DB-Instances im DB-	Bei Änderung des Wartungsfensters für das DB-Cluster über die AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Änderung des Wartungsfensters für eine DB-Instance über die AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Änderung des Wartungsfensters für das DB-Cluster über die AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option auf `--preferred-maintenance-window` fest. Bei Änderung des Wartungsfensters für eine DB-Instance über die AWS CLI: Führen Sie `modify-db-`	Das gesamte DB-Cluster oder eine einzelne DB-Instance	Wenn eine oder mehrere ausstehende Aktionen einen Ausfall verursachen und das Wartungsfenster auf die aktuelle Zeit geändert wird, werden die ausstehenden Aktionen sofort angewendet und es kommt zu einem Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Cluster unabhängig voneinander festlegen. Wenn eine Änderung das gesamte DB-Cluster betrifft, wird die Änderung während des Wartungsfensters für das DB-Cluster ausgeführt. Wenn eine Änderung eine einzelne DB-Instance betrifft, wird die Änderung während des Wartungsfensters für diese DB-Instance ausgeführt. Das Wartungsfenster und das Sicherungsfenster für den DB-Cluster können sich nicht überschneiden. Weitere Informationen finden Sie unter Amazon-RDS-Wartungsfenster.	`instance` aus und legen Sie die Option auf `--preferred-maintenance-window` fest. Bei Änderung des Wartungsfensters für das DB-Cluster über die RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `PreferredMaintenanceWindow` fest. Bei Änderung des Wartungsfensters für eine DB-Instance über die RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `PreferredMaintenanceWindow` fest.		

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Master-Anmeldeinformationen verwalten in AWS Secrets Manager Wählen Sie Master-Anmeldeinformationen verwalten in AWS Secrets Manager aus, um das Hauptbenutzerpasswort in Secrets Manager geheim zu verwalten. Wählen Sie optional einen KMS-Schlüssel zum Schutz des Secrets aus. Wählen Sie aus den KMS-Schlüsseln in Ihrem Konto oder geben Sie den Schlüssel eines anderen Kontos ein. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager. Wenn Aurora bereits das Hauptbenutzerpasswort für den	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` und legen Sie die Optionen `--manage-master-user-password | --no-manage-master-user-password` und `--master-user-secret-kms-key-id` fest. Legen Sie die Option `--rotate-master-user-password` fest, um das Hauptbenutzerpasswort sofort zu rotieren. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie die Parameter `ManageMasterUserPa`	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
DB-Cluster verwaltet , können Sie dieses Passwort mit der Option Rotate secret immediately (Sofortige Secret-Drehung) rotieren. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager.	`ssword` und `MasterUserSecretKmsKeyId` fest. Legen Sie den Parameter `RotateMasterUserPassword` auf `true` fest, um das Hauptbenutzerpasswort sofort zu rotieren.		

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Netzwerktyp Die vom DB-Cluster unterstützten IP-Adressierungsprotokolle. IPv4, um anzugeben, dass Ressourcen mit dem DB-Cluster nur über das IPv4-Adressierungsprotokoll kommunizieren können. Dual-Stack-Modus, um anzugeben, dass Ressourcen mit dem DB-Cluster über IPv4, IPv6 oder beiden kommunizieren können. Verwenden Sie den Dual-Stack-Modus, wenn Sie über Ressourcen verfügen, die über das IPv6-Adressierungsprotokoll mit Ihrem DB-Cluster kommunizieren müssen. Wenn Sie den Dual-Stack-Modus verwenden möchten, stellen Sie sicher, dass	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--network-type` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `NetworkType` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
mindestens zwei Subnetze über zwei Availability Zones verteilt sind die sowohl das IPv4- als auch das IPv6-Netzwerkprotokoll unterstützen. Stellen Sie außerdem sicher, dass Sie einen IPv6-CIDR-Block mit allen Subnetzen in der von Ihnen angegebenen DB-Subnetzgruppe verknüpfen. Weitere Informati onen finden Sie unter Amazon-Aurora-IP-Adressierung.			

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Neues Master-Passwort Das Passwort für den Hauptbenutzer. • Für Aurora MySQL muss das Passwort 8–41 druckbare ASCII-Zeichen enthalten. • Für Aurora PostgreSQL muss es 8–99 druckbare ASCII-Zeichen enthalten. • Es darf weder /, ", @ noch ein Leerzeichen enthalten.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--master-user-password` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `MasterUserPassword` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Performance Insights Ob Performance-Insights aktiviert werden soll; dabei handelt es sich um ein Tool, das den Workload Ihrer DB-Instance überwacht, sodass Sie die Leistung Ihrer Datenbank analysieren und korrigieren können. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--enable-performance-insights|--no-enable-performance-insights` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `EnablePerformanceInsights` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Performance Insights AWS KMS key Die AWS KMS key Kennung für die Verschlüsselung von Performance-Insights-Daten. Der KMS-Schlüsselbezeichner ist der Amazon Resource Name (ARN), der Schlüsselbezeichner oder der Schlüsselalias für den KMS-Schlüssel. Weitere Informationen finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--performance-insights-kms-key-id` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `PerformanceInsightsKMSKeyId` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Performance-Insights-Aufbewahrungszeitraum Der Zeitraum in Tagen, über den Performance-Insights-Daten aufbewahrt werden sollen. Die Aufbewahrungseinstellung ist Standard (7 Tage). Um Ihre Leistungsdaten länger aufzubewahren, geben Sie 1–24 Monate an. Weitere Informationen zum Aufbewahrungszeitraum finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights. Weitere Informationen finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--performance-insights-retention-period` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `PerformanceInsightsRetentionPeriod` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Erhebungsstufe Ein Wert, der die Reihenfolge angibt, in der eine Aurora Replica zur primären Instance in einem DB-Cluster hochgestuft wird, wenn die vorhandene primäre Instance ausfällt. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--promotion-tier` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `PromotionTier` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Öffentlicher Zugriff Publicly accessible (Öffentlich zugänglich), um der DB-Instance eine öffentliche IP-Adresse zuzuweisen (was bedeutet, dass sie von außerhalb der VPC aus zugänglich ist). Damit der öffentliche Zugriff für eine DB-Instance möglich ist, muss sie sich auch in einem öffentlichen Subnetz der VPC befinden. Not publicly accessible (Nicht öffentlich zugänglich), um die DB-Instance nur innerhalb der VPC zugänglich zu machen. Weitere Informationen finden Sie unter Ausblenden einer DB-Clusters in einer VPC vor dem Internet. Um eine Verbindung zu einer DB-Instance von außerhalb	Bei Verwendung der AWS-Managementkonsole: Ändern einer DB-Instance in einem DB-Cluster. Bei Verwendung der AWS CLI: Führen Sie `modify-db-instance` aus und legen Sie die Option `--publicly-accessible|--no-publicly-accessible` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBInstance` auf und legen Sie den Parameter `PubliclyAccessible` fest.	Nur die angegebene DB-Instance	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
ihrer Amazon VPC herzustellen, muss die DB-Instance öffentlich zugänglich sein und der Zugriff muss unter Anwendung der Regeln für den eingehenden Datenverkehr der Sicherheitsgruppe der DB-Instance gewährt werden. Darüber hinaus müssen weitere Anforderungen erfüllt werden. Weitere Informationen finden Sie unter Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden. Wenn Ihre DB-Instance nicht öffentlich zugänglich ist, können Sie auch eine AWS-Site-to-Site-VPN-Verbindung oder eine Direct Connect-Verbindung verwenden, um über ein privates Netzwerk darauf zuzugreifen. Weitere Informationen finden			

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Sie unter Richtlinie für den Datenverkehr zwischen Netzwerken.			
Kapazitätseinstellungen von Serverless v2 Die in Aurora Capacity Units (ACUs) gemessene Datenbankkapazität eines DB-Clusters von Aurora Serverless v2. Weitere Informationen finden Sie unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--serverless-v2-scaling-configuration` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `ServerlessV2ScalingConfiguration` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall. Die Änderung wird sofort übernommen. Diese Einstellung ignoriert die Einstellung „Apply Immediately (Sofort Anwenden)“.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Sicherheitsgruppe Die Sicherheitsgruppe, die Sie mit dem DB-Cluster verknüpfen möchten. Weitere Informationen finden Sie unter Zugriffskontrolle mit Sicherheitsgruppen.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--vpc-security-group-ids` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `VpcSecurityGroupIds` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellung und Beschreibung	Art	Scope	Hinweise zur Ausfallzeit
Backtrack-Zielfenster Der Zeitraum in Sekunden, über den Sie Ihr DB-Cluster zurückverfolgen möchten. Diese Einstellung ist nur für Aurora MySQL verfügbar und nur dann, wenn während der Erstellung des DB-Clusters die Rückverfolgung aktiviert wurde.	Bei Verwendung der AWS-Managementkonsole: Ändern des DB-Clusters über die Konsole, die CLI und die API. Bei Verwendung der AWS CLI: Führen Sie `modify-db-cluster` aus und legen Sie die Option `--backtrack-window` fest. Bei Verwendung der RDS API: Rufen Sie `ModifyDBCluster` auf und legen Sie den Parameter `BacktrackWindow` fest.	Gesamter DB-Cluster	Diese Änderung verursacht keinen Ausfall.

Einstellungen, die nicht für Amazon-Aurora-DB-Cluster gelten

Die folgenden Einstellungen im AWS CLI-Befehl `modify-db-cluster` und in der RDS-API-Operation `ModifyDBCluster` werden nicht auf Amazon-Aurora-DB-Cluster angewendet.

> **Note**
>
> Sie können die AWS-Managementkonsole nicht verwenden, um diese Einstellungen für Aurora-DB-Cluster zu ändern.

AWS CLI-Einstellung	RDS-API-Einstellung
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`

Einstellungen, die nicht für Amazon-Aurora-DB-Instances gelten

Die folgenden Einstellungen im AWS CLI-Befehl `modify-db-instance` und in der RDS-API-Operation `ModifyDBInstance` werden nicht auf Amazon-Aurora-DB-Instances angewendet.

> **Note**
>
> Sie können das AWS-Managementkonsole nicht verwenden, um diese Einstellungen für Aurora-DB-Instances zu ändern.

AWS CLI-Einstellung	RDS-API-Einstellung
`--allocated-storage`	`AllocatedStorage`
`--allow-major-version-upgrade|--no-allow-major-version-upgrade`	`AllowMajorVersionUpgrade`
`--copy-tags-to-snapshot|--no-copy-tags-to-snapshot`	`CopyTagsToSnapshot`
`--domain`	`Domain`
`--db-security-groups`	`DBSecurityGroups`
`--db-subnet-group-name`	`DBSubnetGroupName`
`--domain-iam-role-name`	`DomainIAMRoleName`
`--multi-az|--no-multi-az`	`MultiAZ`
`--iops`	`Iops`
`--license-model`	`LicenseModel`
`--network-type`	`NetworkType`
`--option-group-name`	`OptionGroupName`
`--processor-features`	`ProcessorFeatures`
`--storage-type`	`StorageType`
`--tde-credential-arn`	`TdeCredentialArn`
`--tde-credential-password`	`TdeCredentialPassword`
`--use-default-processor-features|--no-use-default-processor-features`	`UseDefaultProcessorFeatures`

Hinzufügen von Aurora-Replicas zu einem DB-Cluster

Ein Aurora-DB-Cluster mit Replikation hat eine primäre DB-Instance und bis zu 15 Aurora-Replicas. Die primäre DB-Instance Unterstützt Lese- und Schreiboperationen und führt alle Datenänderungen im Cluster-Volume durch. Aurora-Replicas stellen eine Verbindung zu demselben Speichervolume wie die primäre DB-Instance her, aber unterstützen nur Lesevorgänge. Sie nutzen Aurora-Replicas, um schreibgeschützte Workloads von der primären DB-Instance auszulagern. Informationen finden Sie unter Aurora-Replikate.

Amazon Aurora Replicas haben die folgenden Einschränkungen:

- Sie können keine Aurora Replica für einen Aurora Serverless v1-DB-Cluster erstellen. Aurora Serverless v1 verfügt über eine einzelne DB-Instance, die automatisch nach oben und unten skaliert wird, um alle Lese- und Schreibvorgänge der Datenbank zu unterstützen.

  Sie können jedoch Aurora Serverless v2-DB-Clustern Reader-Instances hinzufügen. Weitere Informationen finden Sie unter Hinzufügen eines Aurora Serverless v2-Readers.

Wir empfehlen Ihnen, die primäre Instance und die Aurora-Replicas Ihres DB-Clusters über mehrere Availability Zones zu verteilen, um die Verfügbarkeit Ihres DB-Clusters zu verbessern. Weitere Informationen finden Sie unter Verfügbarkeit in Regionen.

Zum Entfernen einer Aurora Replica aus einem Aurora-DB-Cluster löschen Sie die Aurora Replica anhand der Anweisungen in Löschen einer DB-Instance aus einem Aurora-DB-Cluster.

> **Note**
>
> Amazon Aurora unterstützt auch die Replikation mit einer externen Datenbank wie einer RDS-DB-Instance. Die RDS-DB-Instance muss sich in derselben AWS Region wie Amazon Aurora befinden. Weitere Informationen finden Sie unter Replikation mit Amazon Aurora.

Sie können Aurora Replicas mithilfe der, der oder der AWS-Managementkonsole RDS-API zu einem DB-Cluster hinzufügen. AWS CLI

Konsole

So fügen Sie eine Aurora Replica zu einem DB-Cluster hinzu:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus. Wählen Sie dann den DB-Cluster aus, in dem Sie die neue DB-Instance hinzufügen möchten.
3. Stellen Sie sicher, dass sowohl der Cluster als auch die primäre Instance den Status Verfügbar aufweisen. Wenn sich der DB-Cluster oder die primäre Instance in einem Übergangszustand wie Erstellen befindet, können Sie kein Replikat hinzufügen.

   Wenn der Cluster keine primäre Instance hat, erstellen Sie eine mit dem create-db-instance AWS CLI Befehl. Diese Situation kann auftreten, wenn Sie die CLI verwendet haben, um einen DB-Cluster-Snapshot wiederherzustellen und dann den Cluster in anzeige AWS-Managementkonsole.
4. Wählen Sie für Actions (Aktionen) Add reader (Reader hinzufügen) aus.

   Die Seite Add reader (Reader hinzufügen) wird angezeigt.
5. Geben Sie auf der Seite Add reader (Reader hinzufügen) Optionen für Ihr Aurora-Replica an. In der folgenden Tabelle werden die Einstellungen für ein Aurora Replica angezeigt.

Option	Vorgehensweise
Availability Zone	Legen Sie fest, ob Sie eine bestimmte Availability Zone angeben möchten. Die Liste beinhaltet nur jene Availability Zones, die der DB-Subnetzgruppe zugeordnet sind, die Sie beim Erstellen des DB-Cluste rs ausgewählt haben. Weitere Informationen über Availability Zones finden Sie unter Regionen und Availability Zones.
Öffentlich zugänglich	Wählen Sie `Yes` aus, um der Aurora-Replica eine öffentliche IP-Adresse zuzuweisen. Wählen Sie andernfalls `No` aus. Weitere Informationen darüber, wie Sie den öffentlichen Zugriff für Aurora-Replicas

Option	Vorgehensweise
	deaktivieren, finden Sie unter Ausblenden einer DB-Clusters in einer VPC vor dem Internet.
Verschlüsselung	Klicken Sie auf `Enable encryption` , um die Verschlüsselung im Ruhezustand für diese Aurora-Replica zu aktivieren. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
DB-Instance-Klasse	Wählen Sie eine DB-Instance-Klasse aus, die die Verarbeitungs- und Speicheranforderungen für das Aurora Replica definiert. Weitere Informationen zu den Optionen für DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
Aurora-Replikatquelle	Wählen Sie eine Kennung für die primäre Instance aus, für die Sie ein Aurora Replica erstellen möchten.
DB-Instance-Kennung	Geben Sie einen Namen für die Instanz ein, der für Ihr Konto in der ausgewählten AWS Region eindeutig ist. Sie können dem Namen zusätzliche Informationen hinzufügen, z. B. die AWS Region und die DB-Engine , die Sie ausgewählt haben**`aurora-read-instance1`** .
Priorität	Wählen Sie eine Failover-Priorität für die Instance aus. Wenn Sie keinen Wert auswählen, wird als Standard tier-1 eingestellt. Diese Priorität bestimmt die Reihenfolge, in der Aurora-Replikate bei der Wiederherstellung nach einem Ausfall der primären Instance hochgestuft werden. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.
Datenbankport	Der Port für ein Aurora Replica ist derselbe wie der für das DB-Cluster.

Option	Vorgehensweise
DB-Parametergruppe	Wählen Sie eine Parametergruppe aus. Aurora verfügt über eine Standardparametergruppe, die Sie verwenden können, oder Sie können Ihre eigene Parametergruppe erstellen. Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.
Performance Insights	Das Kontrollkästchen Performance Insights aktivieren ist standardmäßig aktiviert. Der Wert wird nicht von der Writer-Instance geerbt. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .
Verbesserte Überwachung	Wählen Sie Erweiterte Überwachung aktivieren aus, um die Erfassung von Metriken in Echtzeit für das Betriebssystem zu aktivieren, in dem Ihr DB-Cluster ausgeführt wird. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).
Überwachungsrolle	Nur verfügbar, wenn Verbesserte Überwachung auf Erweiterte Überwachung aktivieren gesetzt ist. Wählen Sie die IAM-Rolle, die Sie erstellt haben, damit Amazon RDS für Sie mit Amazon CloudWatch Logs kommunizieren kann, oder wählen Sie Standard, damit RDS eine Rolle für Sie erstellt. `rds-monitoring-role` Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).
Granularität	Nur verfügbar, wenn Erweiterte Überwachung auf Erweiterte Überwachung aktivieren gesetzt ist. Mit ihr können Sie die Zeitspanne zwischen den Erfassungen der Kennzahlen des DB-Clusters in Sekunden festlegen.

Option	Vorgehensweise
Kleinere Versions-Upgrades automatisch aktivieren	Wählen Sie Kleinere Versions-Upgrades automatisch aktivieren aus, wenn Ihr Aurora-DB-Cluster automatisch Upgrades der DB-Engine-Unterversion erhalten soll, sobald diese verfügbar sind. Die Einstellung Automatisches Unterversion-Upgrade gilt nur für Aurora-PostgreSQL- und Aurora-MySQL-DB-Cluster. Bei Clustern von Aurora MySQL 2.x werden mit dieser Einstellung die Cluster auf eine maximale Version von 2.07.2 aktualisiert. Weitere Informationen über Engine-Updates für Aurora PostgreSQL finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL. Weitere Informationen über Engine-Updates für Aurora MySQL finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.

6. Wählen Sie Add reader (Reader hinzufügen) aus, um das Aurora Replica zu erstellen.

AWS CLI

Um eine Aurora Replica in Ihrem DB-Cluster zu erstellen, führen Sie den create-db-instance AWS CLI Befehl aus. Beziehen Sie den Namen des DB-Clusters als Option `--db-cluster-identifier` mit ein. Optional können Sie mithilfe des Parameters `--availability-zone` eine Availability Zone für die Aurora-Replica festlegen, wie in den folgenden Beispielen dargestellt.

Mit dem folgenden Befehl beispielsweise wird eine neue MySQL 5.7-kompatible Aurora-Replik mit dem Namen erstell `sample-instance-us-west-2a`.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Für Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

Mit dem folgenden Befehl wird eine neue MySQL 5.7-kompatible Aurora-Replica mit dem Namen `sample-instance-us-west-2a` erstellt.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Für Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

Der folgende Befehl erstellt ein neues PostgreSQL-kompatibles Aurora-Replikat mit dem Namen `sample-instance-us-west-2a`.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large \
    --availability-zone us-west-2a
```

Für Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large ^
    --availability-zone us-west-2a
```

RDS-API

Um eine Aurora Replica in Ihrem DB-Cluster zu erstellen, rufen Sie den DBInstance Vorgang Create auf. Beziehen Sie den Namen des DB-Clusters als Parameter `DBClusterIdentifier` mit ein. Optional können Sie mithilfe des Parameters `AvailabilityZone` eine Availability Zone für die Aurora-Replica angeben.

Informationen zu Amazon EC2 Auto Scaling Amazon Aurora mit Aurora-Repliken finden Sie in den folgenden Abschnitten.

Themen

- Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten
- Hinzufügen einer Auto-Scaling-Richtlinie zu einem Amazon-Aurora-DB-Cluster
- Bearbeiten einer Auto-Scaling-Richtlinie für einen Amazon-Aurora-DB-Cluster
- Löschen einer Auto-Scaling-Richtlinie aus Ihrem Amazon-Aurora-DB-Cluster

Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten

Um Ihre Konnektivitäts- und Workload-Anforderungen zu erfüllen, passt Aurora Auto Scaling dynamisch die Anzahl der Aurora-Replicas (Reader-DB-Instances) an, die für einen Aurora-DB-Cluster bereitgestellt werden. Aurora Auto Scaling ist für Aurora MySQL und Aurora PostgreSQL verfügbar. Mithilfe von Aurora Auto Scaling kann Ihr Aurora-DB-Cluster plötzliche Konnektivitäts- oder Workloaderhöhungen bewältigen. Wenn die Konnektivität oder Workload abnimmt, entfernt Aurora Auto Scaling unnötige Aurora-Replicas, sodass Sie nicht für nicht genutzte bereitgestellte DB-Instances zahlen müssen.

Sie definieren und wenden eine Skalierungsrichtlinie auf einen Aurora DB-Cluster an. Die Skalierungsrichtlinie definiert die minimale und maximale Anzahl von Aurora-Replicas, die Aurora Auto Scaling verwalten kann. Basierend auf der Richtlinie passt Aurora Auto Scaling die Anzahl der Aurora Replicas als Reaktion auf die tatsächlichen Workloads, die anhand von CloudWatch Amazon-Metriken und Zielwerten bestimmt werden, nach oben oder unten an.

> **Note**
>
> Aurora-Auto-Scaling gilt nicht für die Workload auf der Writer-DB-Instance. Aurora-Auto-Scaling dient nur als Unterstützung bei der Workload auf den Reader-Instances.

Sie können die verwenden AWS-Managementkonsole , um eine Skalierungsrichtlinie anzuwenden, die auf einer vordefinierten Metrik basiert. Alternativ können Sie entweder die AWS CLI oder die Aurora Auto Scaling Scaling-API verwenden, um eine Skalierungsrichtlinie anzuwenden, die auf einer vordefinierten oder benutzerdefinierten Metrik basiert.

Themen

- Bevor Sie beginnen
- Aurora-Auto Scaling-Richtlinien
- DB-Instance IDs und Tagging
- Aurora Auto Scaling und Performance Insights

Bevor Sie beginnen

Bevor Sie Aurora Auto Scaling mit einem Aurora-DB-Cluster verwenden können, müssen Sie zunächst einen Aurora-DB-Cluster mit einer primären (Writer-)DB-Instance erstellen. Weitere

Informationen zum Erstellen eines Aurora-DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Aurora Auto Scaling skaliert einen DB-Cluster nur, wenn sich der DB-Cluster im verfügbaren Status befindet.

Wenn Aurora Auto Scaling eine neue Aurora-Replica hinzufügt, gehört die neue Aurora-Replica derselben DB-Instance-Klasse an, die auch von der primären Instance verwendet wird. Weitere Informationen zu DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora. Die Hochstufungsebene für neue Aurora Replicas ist auf die niedrigste Priorität eingestellt, die standardmäßig 15 ist. Dies bedeutet, dass während eines Failovers eine Replica mit einer höheren Priorität, wie beispielsweise eine manuell erstellte, zuerst hochgestuft wird. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.

Aurora Auto Scaling entfernt nur die Aurora-Replicas, die es selbst erstellt hat.

Um von Aurora Auto Scaling zu profitieren, müssen Ihre Anwendungen Verbindungen zu neuen Aurora-Replicas unterstützen. Hierfür empfehlen wir die Verwendung des Aurora-Reader-Endpunkts. Sie können einen Treiber wie den AWS JDBC-Treiber verwenden. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

> **Note**
>
> Aurora Global Databases unterstützen derzeit kein Aurora Auto Scaling für sekundäre DB-Cluster.

Aurora-Auto Scaling-Richtlinien

Aurora Auto Scaling verwendet eine Skalierungsrichtlinie, um die Anzahl der Aurora-Replicas in einem Aurora-DB-Cluster anzupassen. Aurora-Auto Scaling besteht aus folgenden Komponenten:

- Eine serviceverknüpfte Rolle
- Eine Zielmetrik
- Minimale und maximale Kapazität
- Eine Ruhephase

Themen

- Serviceverknüpfte Rolle
- Zielmetrik
- Minimale und maximale Kapazität
- Ruhephase
- Aktivieren oder Deaktivieren von Scale-In-Aktivitäten
- Hinzufügen, Bearbeiten oder Löschen von Auto-Scaling-Richtlinien

Serviceverknüpfte Rolle

Aurora Auto Scaling verwendet die serviceverknüpfte Rolle `AWSServiceRoleForApplicationAutoScaling_RDSCluster`. Weitere Informationen finden Sie unter Serviceverknüpfte Rollen für Application Auto Scaling im Benutzerhandbuch zu Application Auto Scaling.

Zielmetrik

Bei dieser Art von Richtlinien wird eine vordefinierte oder benutzerdefinierte Metrik und ein Zielwert für die Metrik in einer Konfiguration der Skalierungsrichtlinien für die Zielverfolgung festgelegt. Aurora Auto Scaling erstellt und verwaltet CloudWatch Alarme, die die Skalierungsrichtlinie auslösen, und berechnet die Skalierungsanpassung auf der Grundlage der Metrik und des Zielwerts. Die Skalierungsrichtlinie fügt Aurora-Repliken hinzu oder entfernt sie, wenn erforderlich, um die Metrik auf dem angegebenen Zielwert oder in der Nähe davon zu halten. Abgesehen davon, dass eine Skalierungsrichtlinie für die Ziel-Nachverfolgung die Metrik nahe an dem Zielwert hält, passt sie sich auch an die Schwankungen in der Metrik aufgrund einer sich ändernden Workload an. Eine solche Richtlinie minimiert auch schnelle Schwankungen in der Anzahl der verfügbaren Aurora-Repliken für Ihren DB-Cluster.

Nehmen wir zum Beispiel eine Skalierungsrichtlinie, die die vordefinierte durchschnittliche CPU-Auslastung verwendet. Eine solche Richtlinie kann die CPU-Auslastung bei einem bestimmten Prozentsatz der Auslastung halten, z. B. bei 40 Prozent.

> **Note**
>
> Für jeden Aurora DB-Cluster können Sie nur eine Auto-Scaling-Richtlinie für jede Zielmetrik erstellen.

Wenn Sie Aurora-Auto-Scaling konfigurieren, wird der Zielmetrikwert als Durchschnitt aller Reader-Instances im Cluster berechnet. Diese Berechnung wird wie folgt durchgeführt:

- Beinhaltet alle Reader-Instances im Aurora-Cluster, unabhängig davon, ob sie von Auto Scaling verwaltet oder manuell hinzugefügt werden.
- Beinhaltet Instances, die benutzerdefinierten Endpunkten zugeordnet sind. Benutzerdefinierte Endpunkte haben keinen Einfluss auf die Berechnung von Zielmetriken.
- Beinhaltet nicht die Writer-Instance des Clusters.

Die Metriken werden CloudWatch anhand der folgenden Dimensionen abgeleitet:

- `DBClusterIdentifier`
- `Role=READER`

Betrachten Sie zum Beispiel einen Aurora-MySQL-Cluster mit dem folgenden Setup:

- manuelle Instances (nicht von Auto Scaling gesteuert):
  - Writer mit 50 % CPU-Auslastung
  - Reader 1 (benutzerdefinierter Endpunkt: `custom-reader-1`) mit einer CPU-Auslastung von 90 %
  - Reader 2 (benutzerdefinierter Endpunkt: `custom-reader-2`) mit einer CPU-Auslastung von 90 %
- Auto-Scaling-Instance:
  - Reader 3 (mit Auto Scaling hinzugefügt) mit 10 % CPU-Auslastung

In diesem Szenario wird die Zielmetrik für die Auto-Scaling-Richtlinie wie folgt berechnet:

```
Target metric = (CPU utilization of reader 1 + reader 2 + reader 3) / total number of
 readers

Target metric = (90 + 90 + 10) / 3 = 63.33%
```

Die Auto-Scaling-Richtlinie verwendet diesen Wert, um zu bewerten, ob auf der Grundlage des definierten Schwellenwerts ab- oder aufskaliert werden soll.

Berücksichtigen Sie dabei Folgendes:

- Obwohl benutzerdefinierte Endpunkte festlegen, wie der Datenverkehr an bestimmte Reader weitergeleitet wird, schließen sie Reader nicht aus der Metrikberechnung aus.
- Manuelle Instances sind immer in den Berechnungen der Zielmetriken enthalten.
- Um unerwartetes Skalierungsverhalten zu vermeiden, stellen Sie sicher, dass die Auto-Scaling-Konfiguration alle Reader-Instances im Cluster berücksichtigt.
- Wenn ein Cluster keine Reader hat, wird die Metrik nicht berechnet und die Auto-Scaling-Richtlinienalarme bleiben inaktiv. Damit die Auto-Scaling-Richtlinie effektiv funktioniert, muss zu jeder Zeit mindestens ein Reader vorhanden sein.

Minimale und maximale Kapazität

Sie können die maximale Anzahl von Aurora-Replicas angeben, die von Application Auto Scaling verwaltet werden sollen. Dieser Wert muss auf 0 – 15 gesetzt werden und gleich oder größer als der für die Mindestanzahl von Aurora-Replicas angegebene Wert sein.

Sie können auch die minimale Anzahl von Aurora-Replicas angeben, die von Application Auto Scaling verwaltet werden sollen. Dieser Wert muss auf 0 – 15 gesetzt werden und gleich oder kleiner als der für die maximal Anzahl von Aurora-Replicas angegebene Wert sein.

Es muss mindestens eine Reader-DB-Instance vorhanden sein, damit Auto Scaling funktioniert. Wenn der DB-Cluster keine Reader-Instance hat und Sie die Mindestkapazität auf 0 setzen, funktioniert Aurora Auto Scaling nicht.

> **Note**
>
> Die minimale und maximale Kapazität wird für einen Aurora DB-Cluster festgelegt. Die angegebenen Werte gelten für alle mit diesem Aurora DB-Cluster verknüpften Richtlinien.

Ruhephase

Sie können die Reaktionsfähigkeit einer Skalierungsrichtlinie für die Zielverfolgung anpassen, indem Sie Ruhephasen hinzufügen, nachdem eine Skalierung Ihres Aurora DB-Clusters erfolgt ist. Eine Ruhephase blockiert nachfolgende Scale-in- oder Scale-out-Anforderungen bis zum Ablauf der Frist. Diese Blockierungen verlangsamen das Löschen von Aurora-Replicas in Ihrem Aurora-DB-Cluster für Scale-In-Anforderungen und das Erstellen von Aurora-Replicas für Scale-Out-Anforderungen.

Sie können die folgenden Ruhephasen angeben:

- Eine Scale-In-Aktivität reduziert die Anzahl der Aurora-Replicas in Ihrem Aurora-DB-Cluster. Eine Scale-In-Ruhephase gibt die Zeitspanne in Sekunden an, nach der eine Scale-In-Aktivität abgeschlossen sein muss, bevor eine weitere Scale-In-Aktivität gestartet werden kann.
- Eine Scale-Out-Aktivität erhöht die Anzahl der Aurora-Replicas in Ihrem Aurora-DB-Cluster. Eine Scale-Out-Ruhephase gibt die Zeitspanne in Sekunden an, nach der eine Scale-Out-Aktivität abgeschlossen sein muss, bevor eine weitere Scale-Out-Aktivität gestartet werden kann.

> **Note**
>
> Eine Ruhephase zum Aufskalieren wird ignoriert, wenn eine nachfolgende Anforderung zum Aufskalieren für eine größere Anzahl von Aurora-Replikaten gilt als die erste Anforderung.

Wenn Sie keine Ruhephase für das Ab- bzw. Aufskalieren festlegen, wird jeweils der Standardwert von 300 Sekunden verwendet.

Aktivieren oder Deaktivieren von Scale-In-Aktivitäten

Sie können die Scale-In-Aktivitäten für eine Richtlinie aktivieren oder deaktivieren. Die Aktivierung von Scale-In-Aktivitäten ermöglicht es der Skalierungsrichtlinie, Aurora-Repliken zu löschen. Wenn die Scale-In-Aktivitäten aktiviert sind, gilt für die Scale-In-Aktivitäten die Scale-In-Ruhezeit in der Skalierungsrichtlinie. Das Deaktivieren von Scale-In-Aktivitäten verhindert das Löschen von Aurora-Repliken durch die Skalierungsrichtlinie.

> **Note**
>
> Scale-Out-Aktivitäten sind immer aktiviert, so dass die Skalierungsrichtlinie bei Bedarf Aurora-Replicas erstellen kann.

Hinzufügen, Bearbeiten oder Löschen von Auto-Scaling-Richtlinien

Sie können Auto Scaling-Richtlinien mithilfe der, oder Application Auto Scaling-API hinzufügen AWS-Managementkonsole AWS CLI, bearbeiten oder löschen. Weitere Informationen zum Hinzufügen, Bearbeiten oder Löschen von Auto Scaling-Richtlinien finden Sie in den folgenden Abschnitten.

- Hinzufügen einer Auto-Scaling-Richtlinie zu einem Amazon-Aurora-DB-Cluster
- Bearbeiten einer Auto-Scaling-Richtlinie für einen Amazon-Aurora-DB-Cluster

- Löschen einer Auto-Scaling-Richtlinie aus Ihrem Amazon-Aurora-DB-Cluster

DB-Instance IDs und Tagging

Wenn ein Replikat von Aurora Auto Scaling hinzugefügt wird, wird seine DB-Instance-ID von `application-autoscaling-` vorangestellt, z.B. `application-autoscaling-61aabbcc-4e2f-4c65-b620-ab7421abc123`.

Das folgende Tag wird automatisch zur DB-Instance hinzugefügt. Sie können es auf der Registerkarte Tags der Detailseite der DB-Instance anzeigen.

Tag	Value
application-autoscaling:resourceId	cluster:mynewcluster-cluster

Weitere Informationen zu Amazon-RDS-Ressourcen-Tags finden Sie unter Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen.

Aurora Auto Scaling und Performance Insights

Sie können Performance Insights verwenden, um Replikate zu überwachen, die von Aurora Auto Scaling hinzugefügt wurden, genau wie bei jeder Reader-DB-Instance von Aurora.

Weitere Informationen zum Verwenden von Performance Insights zur Überwachung von Aurora-DB-Clustern finden Sie unter Überwachung mit Performance Insights auf .

Hinzufügen einer Auto-Scaling-Richtlinie zu einem Amazon-Aurora-DB-Cluster

Sie können eine Skalierungsrichtlinie über die AWS-Managementkonsole, die AWS CLI oder die Application-Auto-Scaling-API hinzufügen.

> **Note**
>
> Ein Beispiel, bei dem eine Skalierungsrichtlinie mit CloudFormation hinzugefügt wird, finden Sie unter Declaring a scaling policy for an Aurora DB cluster im AWS CloudFormation-Benutzerhandbuch.

Konsole

Sie können eine Skalierungsrichtlinie über die AWS-Managementkonsole zu einem Aurora-DB-Cluster hinzufügen.

So fügen Sie eine automatische Skalierungsrichtlinie zu einem Aurora DB-Cluster hinzu:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Aurora DB-Cluster aus, zu dem Sie die Richtlinie hinzufügen möchten.
4. Wählen Sie die Registerkarte Logs & events (Protokolle und Ereignisse).
5. Wählen Sie im Abschnitt Auto scaling policies (Automatische Skalierungsrichtlinien) die Option Add (Hinzufügen) aus.

   Das Dialogfeld Add Auto Scaling policy (Auto Scaling-Richtlinie hinzufügen) wird aufgerufen.
6. Geben Sie unter Policy Name (Richtlinienname) den Namen der Richtlinie ein.
7. Wählen Sie für die Zielmetrik eine der nachstehenden Optionen aus:

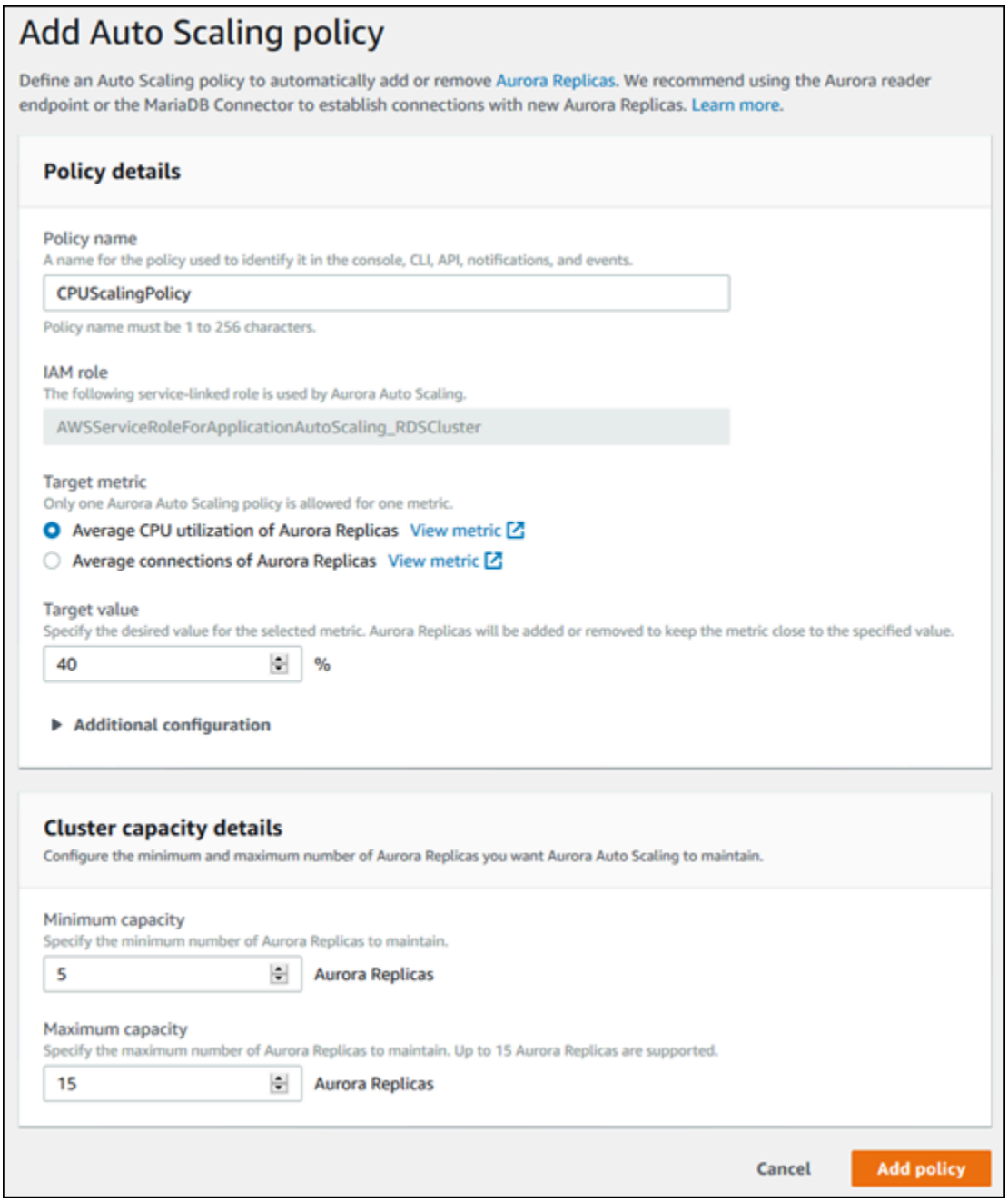
   - Average CPU utilization of Aurora Replicas (Durchschnittliche CPU-Nutzung von Aurora-Replicas), um eine Richtlinie zu erstellen, die auf der durchschnittlichen CPU-Auslastung basiert.
   - Average connections of Aurora Replicas (Durchschnittliche Verbindungen von Aurora-Repliken), um eine Richtlinie zu erstellen, die auf der durchschnittlichen Anzahl der Verbindungen zu den Aurora-Replicas basiert.
8. Geben Sie für den Zielwert eine der nachstehenden Optionen an:

   - Wenn Sie im vorherigen Schritt Average CPU utilization of Aurora Replicas (Durchschnittliche CPU-Nutzung von Aurora-Replicas) ausgewählt haben, geben Sie den Prozentsatz der CPU-Auslastung ein, den Sie auf Aurora-Replicas aufrechterhalten möchten.
   - Wenn Sie im vorherigen Schritt Average connections of Aurora Replicas (Durchschnittliche Verbindungen von Aurora-Replicas) ausgewählt haben, geben Sie die Anzahl der Verbindungen ein, die Sie aufrechterhalten möchten.

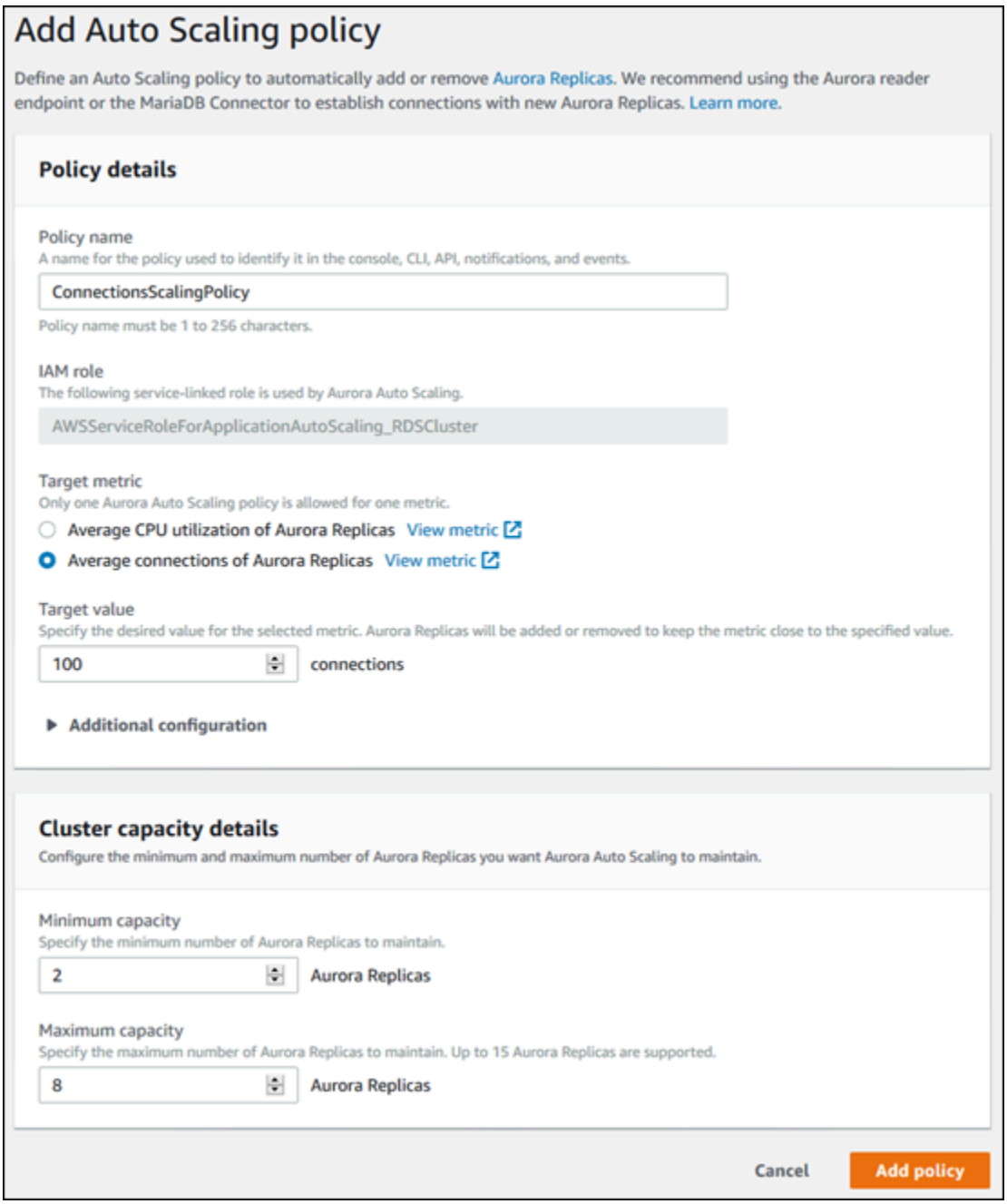
   Aurora-Replicas werden hinzugefügt oder entfernt, um die Metrik in der Nähe des angegebenen Wertes zu halten.

9. (Optional) Erweitern Sie Zusätzliche Konfigurationen, um eine Ruhephase zum Ab- oder Aufskalieren zu erstellen.
10. Geben Sie für Minimale Kapazität die minimale Anzahl von Aurora-Replicas ein, die die Aurora Auto Scaling-Richtlinie beibehalten soll.
11. Geben Sie für Maximale Kapazität die maximale Anzahl von Aurora-Replicas ein, die die Aurora Auto Scaling-Richtlinie beibehalten soll.
12. Wählen Sie Richtlinie hinzufügen aus.

Im folgenden Dialogfeld wird eine Auto Scaling-Richtlinie erstellt, die auf einer durchschnittlichen CPU-Auslastung von 40 Prozent basiert. Die Richtlinie legt ein Minimum von 5 Aurora-Replicas und ein Maximum von 15 Aurora-Replicas fest.



Das folgende Dialogfeld erstellt eine automatische Skalierungsrichtlinie, die auf einer durchschnittlichen Verbindungszahl von 100 basiert. Die Richtlinie legt ein Minimum von zwei Aurora-Replicas und ein Maximum von acht Aurora-Replicas fest.



AWS CLI oder API für Application Auto Scaling

Sie können eine Skalierungsrichtlinie anwenden, die entweder auf einer vordefinierten oder einer benutzerdefinierten Metrik basiert. Hierfür können Sie die AWS CLI oder die Application-Auto-Scaling-API verwenden. Der erste Schritt ist die Registrierung Ihres Aurora-DB-Clusters mit Application Auto Scaling.

Registrieren eines Aurora-DB-Clusters

Bevor Sie Aurora Auto Scaling mit einem Aurora-DB-Cluster verwenden können, müssen Sie zunächst Ihren Aurora-DB-Cluster mit Application Auto Scaling registrieren. Damit legen Sie die Skalierungsdimension und die Grenzen fest, die auf diesen Cluster angewendet werden sollen. Application Auto Scaling skaliert den Aurora-DB-Cluster dynamisch entlang der skalierbaren `rds:cluster:ReadReplicaCount`-Dimension, die die Anzahl der Aurora-Replikate darstellt.

Um Ihren Aurora-DB-Cluster zu registrieren, können Sie entweder die AWS CLI oder die Application-Auto-Scaling-API verwenden.

AWS CLI

Verwenden Sie zum Registrieren Ihres Aurora-DB-Clusters den AWS CLI-Befehl `register-scalable-target` mit folgenden Parametern:

- `--service-namespace` – Legen Sie diesen Wert auf fest `rds`.
- `--resource-id` – Die Ressourcenkennung für den Aurora-DB-Cluster. Für diesen Parameter lautet der Ressourcentyp `cluster` und die eindeutige Kennung ist der Name des Aurora-DB-Clusters, beispielsweise `cluster:myscalablecluster`.
- `--scalable-dimension` – Legen Sie diesen Wert auf fest `rds:cluster:ReadReplicaCount`.
- `--min-capacity` – Die minimale Anzahl an Reader-DB-Instances, die von Application Auto Scaling verwaltet werden sollen. Informationen zur Beziehung zwischen `--min-capacity`, `--max-capacity` und der Anzahl der DB-Instances in Ihrem Cluster finden Sie unter Minimale und maximale Kapazität.
- `--max-capacity` – Die maximale Anzahl an Reader-DB-Instances, die von Application Auto Scaling verwaltet werden sollen. Informationen zur Beziehung zwischen `--min-capacity`, `--max-capacity` und der Anzahl der DB-Instances in Ihrem Cluster finden Sie unter Minimale und maximale Kapazität.

Example

Im folgenden Beispiel registrieren Sie einen Aurora-DB-Cluster mit dem Namen `myscalablecluster`. Die Registrierung zeigt an, dass der DB-Cluster dynamisch skaliert werden soll, um zwischen einer und acht Aurora-Replicas zu haben.

Für Linux, macOS oder Unix:

```
aws application-autoscaling register-scalable-target \
    --service-namespace rds \
    --resource-id cluster:myscalablecluster \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --min-capacity 1 \
    --max-capacity 8 \
```

Für Windows:

```
aws application-autoscaling register-scalable-target ^
    --service-namespace rds ^
    --resource-id cluster:myscalablecluster ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --min-capacity 1 ^
    --max-capacity 8 ^
```

API für Application Auto Scaling

Um Ihren Aurora-DB-Cluster mit Auto Scaling von Anwendungen zu registrieren, verwenden Sie die Application Auto Scaling-API-Operation `RegisterScalableTarget` mit den folgenden Parametern:

- `ServiceNamespace` – Legen Sie diesen Wert auf fest `rds`.
- `ResourceID` – Die Ressourcenkennung für den Aurora-DB-Cluster. Für diesen Parameter lautet der Ressourcentyp `cluster` und die eindeutige Kennung ist der Name des Aurora-DB-Clusters, beispielsweise `cluster:myscalablecluster`.
- `ScalableDimension` – Legen Sie diesen Wert auf fest `rds:cluster:ReadReplicaCount`.
- `MinCapacity` – Die minimale Anzahl an Reader-DB-Instances, die von Application Auto Scaling verwaltet werden sollen. Informationen zur Beziehung zwischen `MinCapacity`, `MaxCapacity` und der Anzahl der DB-Instances in Ihrem Cluster finden Sie unter Minimale und maximale Kapazität.
- `MaxCapacity` – Die maximale Anzahl an Reader-DB-Instances, die von Application Auto Scaling verwaltet werden sollen. Informationen zur Beziehung zwischen `MinCapacity`, `MaxCapacity` und der Anzahl der DB-Instances in Ihrem Cluster finden Sie unter Minimale und maximale Kapazität.

Example

Im folgenden Beispiel registrieren Sie einen Aurora-DB-Cluster mit dem Namen `myscalablecluster` mit der Application Auto Scaling API. Diese Registrierung zeigt an, dass der DB-Cluster dynamisch skaliert werden soll, um zwischen eins bis acht Aurora-Repliken zu haben.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

Definieren einer Skalierungsrichtlinie für einen Aurora-DB-Cluster

Eine Konfiguration der Skalierungsrichtlinie für die Zielverfolgung wird durch einen JSON-Block repräsentiert, in dem die Metriken und Zielwerte definiert sind. Sie können die Konfiguration einer Skalierungsrichtlinie als JSON-Block in einer Textdatei speichern. Sie verwenden diese Textdatei, wenn Sie die AWS CLI oder die Application-Auto-Scaling-API aufrufen. Weitere Informationen zur Syntax der Richtlinienkonfiguration finden Sie unter `TargetTrackingScalingPolicyConfiguration` in der API-Referenz für Application Auto Scaling.

Die folgenden Optionen stehen zur Verfügung, um eine Konfiguration der Skalierungsrichtlinien für die Zielverfolgung zu definieren.

Themen

- Verwenden einer vordefinierten Metrik
- Verwenden einer benutzerdefinierten Metrik

- Verwenden von Ruhephasen
- Deaktivieren der Scale-In-Aktivität

Verwenden einer vordefinierten Metrik

Durch die Verwendung vordefinierter Metriken können Sie schnell eine Skalierungsrichtlinie für die Zielverfolgung für einen Aurora-DB-Cluster definieren, die sowohl mit der Zielverfolgung als auch mit der dynamischen Skalierung in Aurora Auto Scaling gut funktioniert.

Derzeit unterstützt Aurora die folgenden vordefinierten Metriken in Aurora Auto Scaling:

- RDSReaderAverageCPUUtilization – Der Mittelwert der Metrik `CPUUtilization` in CloudWatch über alle Aurora-Replicas im Aurora-DB-Cluster.
- RDSReaderAverageDatabaseConnections – Der Mittelwert der Metrik `DatabaseConnections` in CloudWatch über alle Aurora-Replicas im Aurora-DB-Cluster.

Weitere Informationen über die Metriken `CPUUtilization` und `DatabaseConnections` finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora.

Um eine vordefinierte Metrik in Ihrer Skalierungsrichtlinie zu verwenden, erstellen Sie eine Zielverfolgungskonfiguration für Ihre Skalierungsrichtlinie. Diese Konfiguration muss eine `PredefinedMetricSpecification` für die vordefinierte Metrik und einen `TargetValue` für den Zielwert dieser Metrik enthalten.

Example

Das folgende Beispiel beschreibt eine typische Richtlinienkonfiguration für die Skalierung der Zielverfolgung für einen Aurora DB-Cluster. In dieser Konfiguration wird die vordefinierte Metrik `RDSReaderAverageCPUUtilization` verwendet, um den Aurora-DB-Cluster basierend auf einer durchschnittlichen CPU-Auslastung von 40 Prozent über alle Aurora-Replicas hinweg anzupassen.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

Verwenden einer benutzerdefinierten Metrik

Durch die Verwendung von benutzerdefinierten Metriken können Sie eine Skalierungsrichtlinie für die Zielverfolgung definieren, die Ihren individuellen Anforderungen entspricht. Sie können eine benutzerdefinierte Metrik basierend auf einer beliebigen Aurora-Metrik definieren, die sich proportional zur Skalierung ändert.

Nicht alle Aurora-Metriken funktionieren für die Zielverfolgung. Die Metrik muss eine gültige Auslastungsmetrik sein und beschreiben, wie ausgelastet eine Instance ist. Der Wert der Metrik muss sich proportional zur Anzahl der Aurora-Replicas im Aurora-DB-Cluster erhöhen oder verringern. Diese proportionale Erhöhung oder Verminderung ist notwendig, um die metrischen Daten zur proportionalen Skalierung oder in der Anzahl der Aurora-Repliken zu verwenden.

Example

Das folgende Beispiel beschreibt die Konfiguration einer Zielverfolgung für eine Skalierungsrichtlinie. In dieser Konfiguration passt eine benutzerdefinierte Metrik einen Aurora-DB-Cluster basierend auf einer durchschnittlichen CPU-Auslastung von 50 Prozent über alle Aurora-Replicas in einem Aurora-DB-Cluster namens a `my-db-cluster`.

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier","Value": "my-db-cluster"},
            {"Name": "Role","Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

Verwenden von Ruhephasen

Sie können einen Wert in Sekunden für `ScaleOutCooldown` angeben, um eine Ruhephase für die Hochskalierung Ihres Aurora-DB-Clusters hinzuzufügen. Ähnlich können Sie einen Wert in Sekunden für `ScaleInCooldown` angeben, um eine Ruhephase für die Herunterskalierung Ihres Aurora-DB-Clusters hinzuzufügen. Weitere Informationen über `ScaleInCooldown` und `ScaleOutCooldown`

finden Sie unter `TargetTrackingScalingPolicyConfiguration` in der API-Referenz für Application Auto Scaling.

Example

Das folgende Beispiel beschreibt die Konfiguration einer Zielverfolgung für eine Skalierungsrichtlinie. In dieser Konfiguration wird die vordefinierte Metrik `RDSReaderAverageCPUUtilization` verwendet, um einen Aurora DB-Cluster anzupassen, basierend auf einer durchschnittlichen CPU-Auslastung von 40 Prozent über alle Aurora -Repliken hinweg in diesem Aurora DB-Cluster. Die Konfiguration sieht eine Scale-In-Ruhephase von 10 Minuten und eine Scale-Out-Ruhephase von 5 Minuten vor.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

Deaktivieren der Scale-In-Aktivität

Sie können durch die Konfiguration der Skalierungsrichtlinie für die Zielverfolgung verhindern, dass in Ihrem Aurora DB-Cluster ein Scale-In erfolgt, indem Sie die Scale-In-Aktivität deaktivieren. Das Deaktivieren der Scale-In-Aktivität verhindert das Löschen von Aurora-Repliken durch die Skalierungsrichtlinie, und erlaubt der Skalierungsrichtlinie dennoch, Repliken nach Bedarf zu erstellen.

Sie können einen booleschen Wert für `DisableScaleIn` angeben, um die Scale-In-Aktivität für Ihren Aurora-DB-Cluster zu aktivieren oder zu deaktivieren. Weitere Informationen über `DisableScaleIn` finden Sie unter `TargetTrackingScalingPolicyConfiguration` in der API-Referenz für Application Auto Scaling.

Example

Das folgende Beispiel beschreibt die Konfiguration einer Zielverfolgung für eine Skalierungsrichtlinie. In dieser Konfiguration passt die vordefinierte Metrik `RDSReaderAverageCPUUtilization` einen Aurora-DB-Cluster basierend auf einer durchschnittlichen CPU-Auslastung von 40 Prozent über alle

Aurora-Replicas in diesem Aurora-DB-Cluster hinweg an. Die Konfiguration deaktiviert die Scale-In-Aktivität für die Skalierungsrichtlinie.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "DisableScaleIn": true
}
```

Anwenden einer Skalierungsrichtlinie für einen Aurora-DB-Cluster

Nach der Registrierung Ihres Aurora-DB-Clusters mit Application Auto Scaling und der Definition einer Skalierungsrichtlinie wenden Sie die Skalierungsrichtlinie auf den registrierten Aurora-DB-Cluster an. Um eine Skalierungsrichtlinie auf einen Aurora-DB-Cluster anzuwenden, können Sie die AWS CLI oder die Application-Auto-Scaling-API verwenden.

AWS CLI

Um eine Skalierungsrichtlinie auf Ihren Aurora-DB-Cluster anzuwenden, verwenden Sie den AWS CLI-Befehl `put-scaling-policy` mit den folgenden Parametern:

- `--policy-name` – Der Name der Skalierungsrichtlinie.
- `--policy-type` – Legen Sie diesen Wert auf fest `TargetTrackingScaling`.
- `--resource-id` – Die Ressourcenkennung für den Aurora-DB-Cluster. Für diesen Parameter lautet der Ressourcentyp `cluster` und die eindeutige Kennung ist der Name des Aurora-DB-Clusters, beispielsweise `cluster:myscalablecluster`.
- `--service-namespace` – Legen Sie diesen Wert auf fest `rds`.
- `--scalable-dimension` – Legen Sie diesen Wert auf fest `rds:cluster:ReadReplicaCount`.
- `--target-tracking-scaling-policy-configuration` – Die Konfiguration der Skalierungsrichtlinie für die Zielverfolgung, die für den Aurora-DB-Cluster verwendet werden soll.

Example

Im folgenden Beispiel wenden Sie eine Zielverfolgungs-Skalierungsrichtlinie namens `myscalablepolicy` auf einen Aurora-DB-Cluster namens `myscalablecluster` mit Application

Auto Scaling an. Dazu verwenden Sie eine Richtlinienkonfiguration, die in einer Datei mit dem Namen gespeichert is `config.json`.

Für Linux, macOS oder Unix:

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

Für Windows:

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --target-tracking-scaling-policy-configuration file://config.json
```

API für Application Auto Scaling

Um eine Skalierungsrichtlinie auf Ihren Aurora-DB-Cluster über die Application Auto Scaling API anzuwenden, verwenden Sie die Application Auto Scaling-API-Operation `PutScalingPolicy` mit den folgenden Parametern:

- `PolicyName` – Der Name der Skalierungsrichtlinie.
- `ServiceNamespace` – Legen Sie diesen Wert auf fest `rds`.
- `ResourceID` – Die Ressourcenkennung für den Aurora-DB-Cluster. Für diesen Parameter lautet der Ressourcentyp `cluster` und die eindeutige Kennung ist der Name des Aurora-DB-Clusters, beispielsweise `cluster:myscalablecluster`.
- `ScalableDimension` – Legen Sie diesen Wert auf fest `rds:cluster:ReadReplicaCount`.
- `PolicyType` – Legen Sie diesen Wert auf fest `TargetTrackingScaling`.
- `TargetTrackingScalingPolicyConfiguration` – Die Konfiguration der Skalierungsrichtlinie für die Zielverfolgung, die für den Aurora-DB-Cluster verwendet werden soll.

Example

Im folgenden Beispiel wenden Sie eine Zielverfolgungs-Skalierungsrichtlinie namens `myscalablepolicy` auf einen Aurora-DB-Cluster namens `myscalablecluster` mit Auto Scaling von Anwendungen an. Sie verwenden die Richtlinienkonfiguration `RDSReaderAverageCPUUtilization` basierend auf einer vordefinierten Metrik.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
    }
}
```

Bearbeiten einer Auto-Scaling-Richtlinie für einen Amazon-Aurora-DB-Cluster

Sie können eine Skalierungsrichtlinie über die AWS-Managementkonsole, die AWS CLI oder die Application-Auto-Scaling-API bearbeiten.

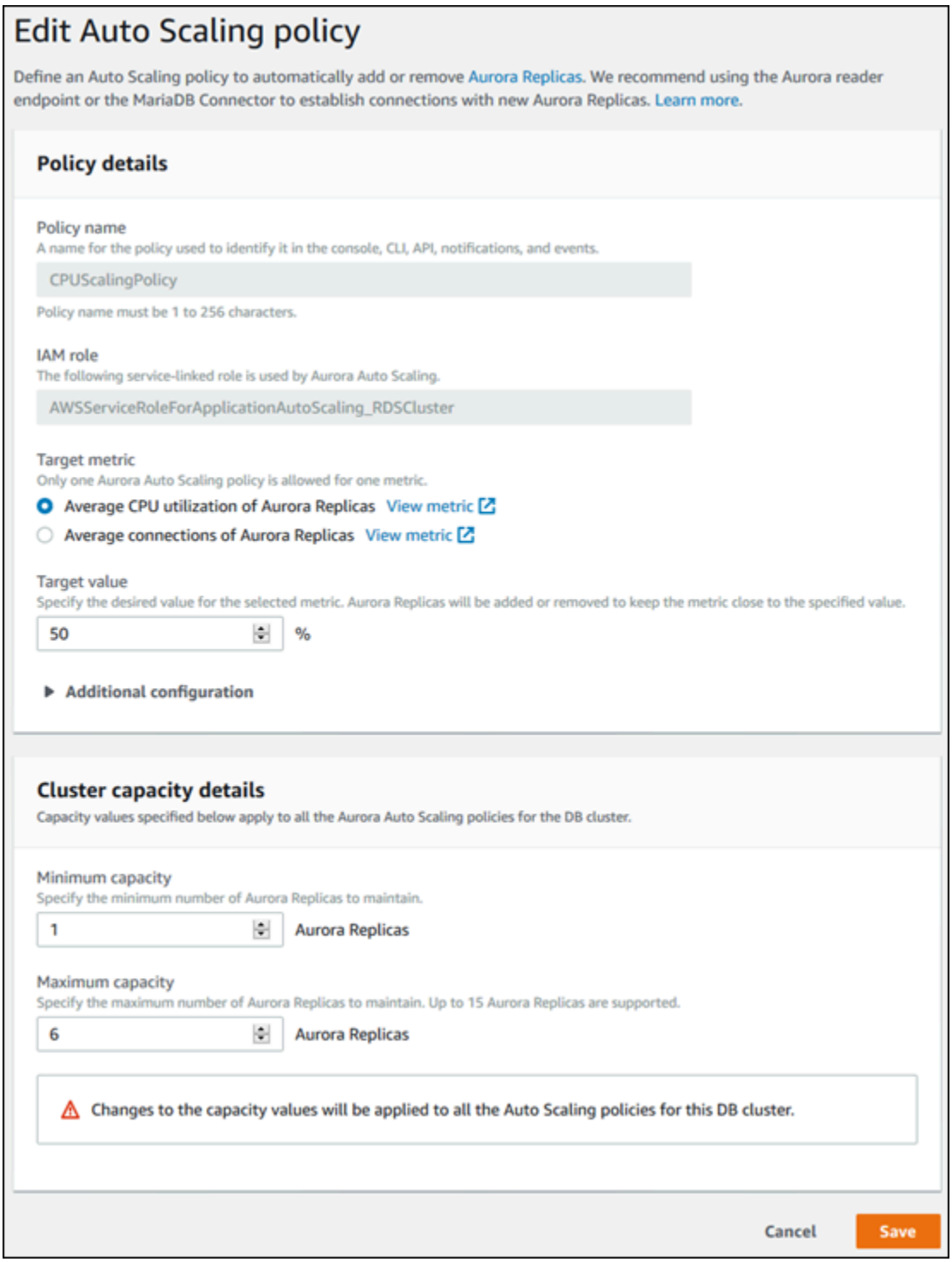
Konsole

Sie können eine Skalierungsrichtlinie über die AWS-Managementkonsole bearbeiten.

So bearbeiten Sie eine automatische Skalierungsrichtlinie für einen Aurora DB-Cluster:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Aurora DB-Cluster aus, dessen automatische Skalierungsrichtlinie Sie bearbeiten möchten.
4. Wählen Sie die Registerkarte Logs & events (Protokolle und Ereignisse).
5. Wählen Sie im Abschnitt Auto scaling policies (Automatische Skalierungsrichtlinien) die betreffende automatische Skalierungsrichtlinie und dann Edit (Bearbeiten) aus.
6. Nehmen Sie die Änderungen an der Richtlinie vor.
7. Wählen Sie Save aus.

Im Folgenden sehen Sie ein Beispiel für das Dialogfeld Edit Auto Scaling policy (Auto Scaling-Richtlinie bearbeiten).



AWS CLI oder API für Application Auto Scaling

Über die AWS CLI oder die Application-Auto-Scaling-API können Sie eine Skalierungsrichtlinie auf die gleiche Weise bearbeiten, wie Sie sie anwenden:

- Wenn Sie die AWS CLI verwenden, geben Sie den Namen der zu bearbeitenden Richtlinie im Parameter `--policy-name` an. Legen Sie neue Werte für die Parameter fest, die Sie ändern möchten.
- Wenn Sie die Application Auto Scaling API verwenden, geben Sie den Namen der zu bearbeitenden Richtlinie im Parameter `PolicyName` an. Legen Sie neue Werte für die Parameter fest, die Sie ändern möchten.

Weitere Informationen finden Sie unter Anwenden einer Skalierungsrichtlinie für einen Aurora-DB-Cluster.

Löschen einer Auto-Scaling-Richtlinie aus Ihrem Amazon-Aurora-DB-Cluster

Sie können eine Skalierungsrichtlinie über die AWS-Managementkonsole, die AWS CLI oder die Application-Auto-Scaling-API löschen.

Konsole

Sie können eine Skalierungsrichtlinie über die AWS-Managementkonsole löschen.

So löschen Sie eine automatische Skalierungsrichtlinie für einen Aurora DB-Cluster:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Aurora DB-Cluster aus, dessen automatische Skalierungsrichtlinie Sie löschen möchten.
4. Wählen Sie die Registerkarte Logs & events (Protokolle und Ereignisse).
5. Wählen Sie im Abschnitt Auto Scaling-Richtlinien die automatische Skalierungsrichtlinie und dann Löschen aus.

AWS CLI

Um eine Skalierungsrichtlinie aus Ihrem Aurora-DB-Cluster zu löschen, verwenden Sie den AWS CLI-Befehl `delete-scaling-policy` mit den folgenden Parametern:

- `--policy-name` – Der Name der Skalierungsrichtlinie.

- `--resource-id` – Die Ressourcenkennung für den Aurora-DB-Cluster. Für diesen Parameter lautet der Ressourcentyp `cluster` und die eindeutige Kennung ist der Name des Aurora-DB-Clusters, beispielsweise `cluster:myscalablecluster`.
- `--service-namespace` – Legen Sie diesen Wert auf fest `rds`.
- `--scalable-dimension` – Legen Sie diesen Wert auf fest `rds:cluster:ReadReplicaCount`.

Example

Im folgenden Beispiel löschen Sie eine Zielverfolgungs-Skalierungsrichtlinie namens `myscalablepolicy` aus einem Aurora-DB-Cluster namens `myscalablecluster`.

Für Linux, macOS oder Unix:

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
```

Für Windows:

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
```

API für Application Auto Scaling

Um eine Skalierungsrichtlinie aus Ihrem Aurora-DB-Cluster zu löschen, verwenden Sie die Application Auto Scaling-API-Operation `DeleteScalingPolicy` mit den folgenden Parametern:

- `PolicyName` – Der Name der Skalierungsrichtlinie.
- `ServiceNamespace` – Legen Sie diesen Wert auf fest `rds`.

- `ResourceID` – Die Ressourcenkennung für den Aurora-DB-Cluster. Für diesen Parameter lautet der Ressourcentyp `cluster` und die eindeutige Kennung ist der Name des Aurora-DB-Clusters, beispielsweise `cluster:myscalablecluster`.
- `ScalableDimension` – Legen Sie diesen Wert auf fest `rds:cluster:ReadReplicaCount`.

Example

Im folgenden Beispiel löschen Sie eine Zielverfolgungs-Skalierungsrichtlinie namens `myscalablepolicy` über die Application Auto Scaling API aus einem Aurora-DB-Cluster namens `myscalablecluster`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

Verwalten von Performance und Skalierung für einen Aurora-DB-Cluster

Für die Verwaltung von Performance und Skalierung von Aurora DB-Clustern und DB-Instances stehen Ihnen folgende Möglichkeiten zur Verfügung:

Themen

- Speicherskalierung
- Skalierung von Instances
- Skalierung von Lesevorgängen
- Verwalten von Verbindungen
- Verwalten von Abfrageausführungsplänen

Speicherskalierung

Der Aurora-Speicher wird automatisch mit den Daten in Ihrem Cluster-Volume skaliert. Wenn die Datenmenge wächst, wird Ihr Cluster-Volume-Speicher je nach Version der DB-Engine erweitert. Weitere Informationen zu den maximalen Volume-Größen von Aurora-Clustern für jede Engine-Version finden Sie unter Amazon Aurora-Größenbeschränkungen. Informationen dazu, welche Datentypen im Cluster-Volume enthalten sind, finden Sie unter Amazon-Aurora-Speicher. Weitere Informationen zur maximalen Größe für eine bestimmte Version finden Sie unter Amazon Aurora-Größenbeschränkungen.

Die Größe Ihres Cluster-Volumes wird stündlich überprüft, um die Kosten für Ihren Speicherplatz zu bestimmen. Informationen zu den Preisen finden Sie in der Aurora-Preisliste.

Obwohl ein Aurora-Cluster-Volume in der Größe auf viele Tebibyte skaliert werden kann, wird nur der Speicherplatz berechnet, den Sie auf dem Volume verwenden. Der Mechanismus zur Bestimmung des fakturierten Speicherplatzes hängt von der Version Ihres Aurora-Clusters ab.

- Wenn Aurora-Daten aus dem Cluster-Volume entfernt werden, verringert sich der gesamte fakturierte Speicherplatz um einen entsprechenden Betrag. Dieses dynamische Größenanpassungsverhalten tritt auf, wenn zugrunde liegende Tabellenbereiche gelöscht oder neu organisiert werden, damit sie weniger Speicherplatz benötigen. So können Sie Speicherkosten senken, indem Sie Tabellen und Datenbanken löschen, die Sie nicht mehr benötigen. Die dynamische Größenanpassung gilt für bestimmte Aurora-Versionen. Im Folgenden werden die

Aurora-Versionen aufgeführt, bei denen die Größe des Cluster-Volumes beim Entfernen von Daten dynamisch geändert wird:

Datenbank-Engine	Versionen mit dynamischer Größenanpassung
Aurora MySQL	• Version 3 (mit MySQL 8.0 kompatibel): alle unterstützten Versionen • Version 2 (mit MySQL 5.7 kompatibel): 2.11 und höher
Aurora PostgreSQL	Alle unterstützten Versionen
Aurora Serverless v2	Alle unterstützten Versionen
Aurora Serverless v1	Nicht verfügbar

- Bei Aurora-Versionen, die niedriger als in der vorherigen Liste angegeben sind, kann das Cluster-Volume Speicherplatz wiederverwenden, der beim Entfernen von Daten freigegeben wird, aber das Volume selbst wird nie kleiner.

Die dynamische Größenanpassung gilt für Operationen, bei denen Tabellenbereiche innerhalb des Cluster-Volumes physisch entfernt oder in der Größe geändert werden. Daher gilt sie für SQL-Anweisungen wie `DROP TABLE`, `DROP DATABASE`, `TRUNCATE TABLE` und `ALTER TABLE ... DROP PARTITION`. Sie gilt nicht für das Löschen von Zeilen mit der `DELETE`-Anweisung. Wenn Sie eine große Anzahl von Zeilen aus einer Tabelle löschen, können Sie anschließend die Aurora MySQL `OPTIMIZE TABLE`-Anweisung ausführen oder die Aurora PostgreSQL `pg_repack`-Erweiterung verwenden, um die Tabelle zu reorganisieren und die Größe des Cluster-Volumes dynamisch zu ändern.

Für Aurora MySQL gelten die folgenden Überlegungen:

- Nachdem Sie Ihren DB-Cluster auf eine DB-Engine-Version aktualisiert haben, die die dynamische Größenanpassung unterstützt, und wenn die Funktion in dieser speziellen AWS-Region aktiviert ist, kann jeglicher Speicherplatz, der später durch bestimmte SQL-Anweisungen freigegeben wird, wie z. B. `DROP TABLE`, zurückgewonnen werden.

Wenn die Funktion in einer bestimmten AWS-Region explizit deaktiviert wird, ist der Speicherplatz möglicherweise nur wiederverwendbar – und nicht zurückgewinnbar –, selbst in Versionen, die die dynamische Größenanpassung unterstützen.

Die Funktion wurde zwischen November 2020 und März 2022 für bestimmte DB-Engine-Versionen (1.23.0–1.23.4, 2.09.0–2.09.3 und 2.10.0) aktiviert und ist in allen nachfolgenden Versionen standardmäßig aktiviert.

- Eine Tabelle wird intern in einem oder mehreren zusammenhängenden Fragmenten unterschiedlicher Größe gespeichert. Während der Ausführung von `TRUNCATE TABLE`-Operationen ist der Speicherplatz, der dem ersten Fragment entspricht, wiederverwendbar und kann nicht zurückgewonnen werden. Andere Fragmente können zurückgewonnen werden. Während `DROP TABLE`-Operationen kann Speicherplatz, der dem gesamten Tablespace entspricht, zurückgewonnen werden.
- Der Parameter `innodb_file_per_table` wirkt sich auf die Organisation des Tabellenspeichers aus. Wenn Tabellen Teil des System-Tablespace sind, wird durch das Löschen der Tabelle die Größe des System-Tablespace nicht verringert. Stellen Sie daher sicher, dass Sie `innodb_file_per_table` für DB-Cluster von Aurora MySQL auf 1 festlegen, um die dynamische Größenanpassung optimal zu nutzen.
- In Version 2.11 und höher wird der temporäre InnoDB-Tablespace gelöscht und beim Neustart neu erstellt. Dadurch wird der vom temporären Tabellenbereich belegte Speicherplatz für das System freigegeben und die Größe des Cluster-Volumes anschließend geändert. Sie sollten Ihren DB-Cluster auf Version 2.11 oder höher aktualisieren, um die dynamische Größenanpassungsfunktion voll auszunutzen.

> **Note**
>
> Mit der dynamischen Größenanpassungsfunktion wird Speicherplatz nicht sofort zurückgewonnen, wenn Tabellen in Tablespaces gelöscht werden, sondern schrittweise mit einer Geschwindigkeit von etwa 10 TB pro Tag. Speicherplatz im System-Tablespace wird nicht zurückgewonnen, da der System-Tablespace niemals entfernt wird. Nicht zurückgewonnener freier Speicherplatz in einem Tabellenraum wird wiederverwendet, wenn ein Vorgang Speicherplatz in diesem Tabellenraum benötigt. Die dynamische Größenanpassungsfunktion kann Speicherplatz nur dann zurückgewinnen, wenn sich der Cluster in einem verfügbaren Zustand befindet.

Sie können überprüfen, wie viel Speicherplatz ein Cluster verwendet, indem Sie die `VolumeBytesUsed`-Metrik in CloudWatch überwachen. Weitere Informationen zu Speicherkosten finden Sie unter Informationen zur Abrechnung des Aurora-Datenspeichers.

- In AWS-Managementkonsole können Sie diese Abbildung in einem Diagramm anzeigen, indem Sie die `Monitoring`-Registerkarte auf der Detailseite für den Cluster anzeigen.
- Mit AWS CLI können Sie einen Befehl ausführen, der dem folgenden Linux-Beispiel ähnelt. Ersetzen Sie Ihre eigenen Werte durch die Start- und Endzeit und den Namen des Clusters.

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" \
  --statistics Average Maximum Minimum \
  --dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

Die Ausgabe dieses Befehls sieht etwa so aus:

```
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T21:25:00+00:00",
            "Average": 182871982080.0,
            "Minimum": 182871982080.0,
            "Maximum": 182871982080.0,
            "Unit": "Bytes"
        }
    ]
}
```

Die folgenden Beispiele zeigen, wie Sie die Speichernutzung eines Aurora-Clusters im Laufe der Zeit mithilfe von AWS CLI-Befehlen auf einem Linux-System verfolgen können. Die Parameter `--start-time` und `--end-time` definieren das Gesamtzeitintervall als einen Tag. Der Parameter `--period` fordert die Messungen in einstündigen Intervallen an. Es ist sinnlos, einen kleinen `--period`-Wert zu wählen, da die Metriken in Intervallen und nicht kontinuierlich erfasst werden. Außerdem werden Aurora-Speicheroperationen manchmal für einige Zeit im Hintergrund fortgesetzt, nachdem die entsprechende SQL-Anweisung beendet wurde.

Das erste Beispiel gibt die Ausgabe im Standard-JSON-Format zurück. Die Datenpunkte werden in beliebiger Reihenfolge zurückgegeben, nicht nach Zeitstempel sortiert. Sie können diese JSON-Daten in ein Diagrammwerkzeug importieren, um sie zu sortieren und zu visualisieren.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T19:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T00:40:00+00:00",
            "Maximum": 198573719552.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T05:40:00+00:00",
            "Maximum": 206827454464.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-04T17:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
... output omitted ...
```

In diesem Beispiel werden dieselben Daten zurückgegeben wie vorhin. Der Parameter `--output` stellt die Daten im kompakten Nur-Text-Format dar. Der Befehl `aws cloudwatch` übergibt seine Ausgabe an den Befehl `sort`. Der Parameter `-k` des Befehls `sort` sortiert die Ausgabe nach dem dritten Feld, dem Zeitstempel im UTC-Format (Universal Coordinated Time).

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id \
```

```
  --output text | sort -k 3
VolumeBytesUsed
DATAPOINTS  182872522752.0  2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS  187667791872.0  2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS  190981029888.0  2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS  195587244032.0  2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS  201048915968.0  2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS  205368492032.0  2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T05:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T13:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T16:41:00+00:00 Bytes
```

Die sortierte Ausgabe zeigt an, wie viel Speicher zu Beginn und Ende des Überwachungszeitraums genutzt wurde. Sie können auch die Punkte in diesem Zeitraum finden, wenn Aurora mehr Speicher für den Cluster zugewiesen hat. Im folgenden Beispiel werden Linux-Befehle verwendet, um die `VolumeBytesUsed`-Start- und Endwerte als Gigabyte (GB) und als Gibibyte (GiB) neu zu formatieren. Gigabyte stehen für Einheiten, die in Potenzen von 10 gemessen werden und werden häufig in Diskussionen über Speicher für Rotationsfestplatten verwendet. Gibibytes stellen Einheiten dar, die in Zweierpotenzen gemessen werden. Aurora-Speichermessungen und -grenzwerte werden typischerweise in 2er-Potenzeinheiten wie Gibibyte und Tebibyte angegeben.

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
```

```
Start: 182 GB, End: 206 GB
```

Die `VolumeBytesUsed`-Metrik gibt an,für wie viel Speicher im Cluster Gebühren anfallen. Daher ist es am besten, diesen Wert nach Möglichkeit zu minimieren. Diese Metrik enthält jedoch keinen Speicher, den Aurora intern im Cluster verwendet und nicht berechnet. Wenn sich Ihr Cluster dem Speicherlimit nähert und möglicherweise nicht genügend Speicherplatz zur Verfügung steht, ist es hilfreich, die `AuroraVolumeBytesLeftTotal`-Metrik zu überwachen und zu maximieren. Im folgenden Beispiel wird eine ähnliche Berechnung ausgeführt wie oben, jedoch für `AuroraVolumeBytesLeftTotal` statt `VolumeBytesUsed`.

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_old_cluster_id \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      140530528288768.0       2023-02-23T19:25:00+00:00       Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

Bei einem Cluster mit Aurora-MySQL-Version 2.09 oder höher oder Aurora PostgreSQL erhöht sich die von `VolumeBytesUsed` gemeldete Größe des freien Speichers, wenn Daten hinzugefügt werden, und verringert sich, wenn Daten entfernt werden. Im folgenden Beispiel wird gezeigt, wie dies geschieht. Dieser Bericht zeigt die maximale und minimale Speichergröße für einen Cluster in 15-Minuten-Intervallen an, wenn Tabellen mit temporären Daten erstellt und gelöscht werden. Der Bericht listet den Maximalwert vor dem Minimalwert auf. Um zu verstehen, wie sich die Speichernutzung innerhalb des 15-minütigen Intervalls verändert hat, interpretieren Sie daher die Zahlen von rechts nach links.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
 Name=DBClusterIdentifier,Value=my_new_cluster_id
  --output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
```

```
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

Das folgende Beispiel zeigt, wie bei Clustern mit kompatiblen Versionen von Aurora MySQL oder Aurora PostgreSQL die von `AuroraVolumeBytesLeftTotal` gemeldete Größe des freien Speichers die höhere 256 TiB-Größenbeschränkung widerspiegelt. Weitere Informationen zu kompatiblen Versionen finden Sie unter Amazon Aurora-Größenbeschränkungen.

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
 "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBClusterIdentifier,Value=pq-57 \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
256 TiB remaining for this cluster
```

Skalierung von Instances

Sie können Ihr Aurora-DB-Cluster skalieren, indem Sie die DB-Instance-Klasse für jede DB-Instance im DB-Cluster ändern. Aurora unterstützt mehrere DB-Instance-Klassen, die für Aurora optimiert sind, abhängig von der Kompatibilität des Datenbankmoduls.

Datenbank-Engine	Skalierung von Instances
Amazon Aurora MySQL	Siehe Skalierung von Aurora MySQL-DB-Instances
Amazon Aurora PostgreSQL	Siehe Skalierung von Aurora PostgreSQL-DB-Instances

Skalierung von Lesevorgängen

Sie können eine Skalierung von Lesevorgängen für Ihren Aurora-DB-Cluster erreichen, indem Sie bis zu 15 Replicas in Ihrem DB-Cluster erstellen. Jede Aurora-Replica gibt dieselben Daten aus dem Cluster-Volume mit minimaler Replica-Verzögerung zurück – im Normalfall weniger als 100 Millisekunden, nachdem die primäre Instance eine Aktualisierung geschrieben hat. Wenn sich Ihr Datenverkehr mit Lesevorgänge erhöht, können Sie zusätzliche Aurora-Replicas erstellen und sie direkt verbinden, um die Auslastung von Schreibvorgängen für Ihr DB-Cluster zu verteilen. Aurora-Replicas müssen sich nicht in derselben DB-Instance-Klasse wie die primäre Instance befinden.

Informationen zum Hinzufügen von Aurora-Replicas zu einem DB-Cluster finden Sie unter Hinzufügen von Aurora-Replicas zu einem DB-Cluster.

Verwalten von Verbindungen

Die maximale Anzahl der zulässigen Verbindungen mit einer Aurora-DB-Instance wird durch den Parameter `max_connections` in der Instance-Ebenen-Parametergruppe für die DB-Instance festgelegt. Der Standardwert dieses Parameters hängt von der DB-Instance-Klasse ab, die für die Kompatibilität von DB-Instance und Datenbank-Engine verwendet wird.

Datenbank-Engine	Vorgabewert max_connections
Amazon Aurora MySQL	Siehe Maximale Verbindungen zu einer Aurora MySQL-DB-Instance
Amazon Aurora PostgreSQL	Siehe Maximale Verbindungen zu einer Aurora PostgreSQL-DB-Instance

> **Tip**
>
> Wenn Ihre Anwendungen häufig Verbindungen öffnen und schließen oder langlebige Verbindungen in großer Zahl offen lassen, empfehlen wir Ihnen, Amazon-RDS-Proxy zu verwenden. RDS-Proxy ist ein vollständig verwalteter, hochverfügbarer Datenbank-Proxy, der Datenbankverbindungen sicher und effizient per Verbindungspooling freigibt. Weitere Informationen zu RDS-Proxy finden Sie unter Amazon-RDS-Proxy für Aurora.

Verwalten von Abfrageausführungsplänen

Mit der Abfrageplanverwaltung für Aurora PostgreSQL haben Sie die Kontrolle darüber, welche Pläne der Optimierer ausführt. Weitere Informationen finden Sie unter Verwalten von Abfrageausführungsplänen für Aurora PostgreSQL.

Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster

Mithilfe der Aurora-Klonfunktion können Sie einen neuen Cluster erstellen, der anfänglich dieselben Datenseiten wie das Original enthält, aber ein separates und unabhängiges Volume darstellt. Der Prozess ist so konzipiert, dass er schnell und kostengünstig ist. Der neue Cluster mit dem zugehörigen Datenvolume wird als clone (Klon) bezeichnet. Das Erstellen eines Klons ist schneller und platzsparender als das physische Kopieren der Daten mit anderen Techniken, wie z. B. das Wiederherstellen eines Snapshots.

Themen

- Übersicht über das Aurora-Klonen
- Einschränkungen für das Aurora-Klonen
- Funktionsweise des Klonens von Aurora
- Erstellen eines Amazon-Aurora-DB-Klons
- VPC-übergreifendes Klonen mit Amazon Aurora
- Kontoübergreifendes Klonen mit AWS RAM und Amazon Aurora

Übersicht über das Aurora-Klonen

Aurora verwendet ein Copy-on-Write-Protokoll, um einen Klon zu erstellen. Dieser Mechanismus verwendet minimalen zusätzlichen Speicherplatz, um einen ersten Klon zu erstellen. Wenn der Klon zum ersten Mal erstellt wird, behält Aurora eine einzelne Kopie der Daten, die vom Aurora-DB-Quellcluster und dem neuen (geklonten) Aurora-DB-Cluster verwendet werden. Zusätzlicher Speicher wird nur zugewiesen, wenn Änderungen an Daten (auf dem Aurora-Speicher-Volume) durch den Aurora-DB-Quellcluster oder den Aurora-DB-Clusterklon vorgenommen werden. Weitere Informationen zum Copy-on-Write-Protokoll finden Sie unter Funktionsweise des Klonens von Aurora.

Das Klonen von Aurora ist besonders nützlich, um Testumgebungen mit Ihren Produktionsdaten schnell einzurichten, ohne Datenbeschädigung zu riskieren. Sie können Klone für viele Arten von Anwendungen verwenden, z. B. für Folgende:

- Experimentieren Sie mit möglichen Änderungen (z. B. Schemaänderungen und Parametergruppenänderungen), um alle Auswirkungen zu bewerten.
- Führen Sie Workload-intensive Vorgänge aus, z. B. das Exportieren von Daten oder das Ausführen analytischer Abfragen auf dem Klon.

- Erstellen Sie eine Kopie Ihres Produktions-DB-Clusters zu Entwicklungs-, Test- oder anderen Zwecken.

Sie können mehr als einen Klon aus demselben Aurora-DB-Cluster erstellen. Sie können auch mehrere Klone aus einem anderen Klon erstellen.

Nachdem Sie einen Aurora-Klon erstellt haben, können Sie die Aurora-DB-Instances anders als den Aurora-DB-Quellcluster konfigurieren. Beispielsweise benötigen Sie möglicherweise keinen Klon für Entwicklungszwecke, um dieselben Hochverfügbarkeitsanforderungen zu erfüllen wie der Aurora-DB-Cluster für die Quellproduktion. In diesem Fall können Sie den Klon mit einer einzelnen Aurora-DB-Instance konfigurieren, anstatt mit mehreren DB-Instances, die vom Aurora-DB-Cluster verwendet werden.

Wenn Sie einen Klon mit einer anderen Bereitstellungskonfiguration als der Quelle erstellen, wird der Klon mit der neuesten Nebenversion der Aurora-DB-Engine der Quelle erstellt.

Wenn Sie Klone aus Ihren Aurora-DB-Clustern erstellen, werden die Klone in Ihrem AWS-Konto erstellt – dem gleichen Konto, dem der Aurora-DB-Quellcluster gehört. Sie können jedoch auch Aurora Serverless v2- und bereitgestellte Aurora-DB-Cluster und -Klone für andere AWS-Konten freigeben. Weitere Informationen finden Sie unter Kontoübergreifendes Klonen mit AWS RAM und Amazon Aurora.

Wenn Sie den Klon für Test-, Entwicklungs- oder andere Zwecke nicht mehr verwenden, können Sie ihn löschen.

Einschränkungen für das Aurora-Klonen

Das Klonen von Aurora hat derzeit die folgenden Einschränkungen:

- Sie können so viele Klone erstellen, wie Sie möchten, bis zur maximalen Anzahl von DB-Clustern, die in der AWS-Region zulässig sind.
- Sie können bis zu 15 Klone mit dem Copy-On-Write-Protokoll erstellen. Nachdem Sie jedoch 15 Klone erstellt haben, ist der nächste Klon, den Sie erstellen, eine vollständige Kopie. Das Full-Copy-Protokoll funktioniert wie eine zeitpunktbezogene Wiederherstellung.
- Sie können keinen Klon in einer anderen AWS-Region als der des Aurora-DB-Quell-Clusters erstellen.
- Sie können keinen Klon von einem Aurora-DB-Cluster ohne die parallele Abfragefunktion für einen Cluster erstellen, der parallele Abfragen verwendet. Um Daten in einen Cluster zu bringen, der

parallele Abfragen verwendet, erstellen Sie einen Snapshot des ursprünglichen Clusters und stellen Sie ihn in dem Cluster wieder her, der die parallele Abfragefunktion verwendet.

- Sie können keinen Klon aus einem Aurora DB-Cluster erstellen, der keine DB-Instances hat. Sie können nur Aurora-DB-Cluster klonen, die über mindestens eine DB-Instance verfügen.
- Sie können einen Klon in einer anderen Virtual Private Cloud (VPC) als der des Aurora-DB-Clusters erstellen. In diesem Fall müssen die Subnetze der VPCs denselben Availability Zones zugeordnet sein.
- Sie können einen von Aurora bereitgestellten Klon aus einem bereitgestellten Aurora-DB-Cluster erstellen.
- Cluster mit Aurora Serverless v2-Instances folgen den gleichen Regeln wie bereitgestellte Cluster.
- Für Aurora Serverless v1:
  - Sie können einen bereitgestellten Klon aus einem Aurora Serverless v1-DB-Cluster erstellen.
  - Sie können einen Aurora Serverless v1-Klon aus einem Aurora Serverless v1- oder bereitgestellten DB-Cluster erstellen.
  - Sie können keinen Aurora Serverless v1-Klon aus einem unverschlüsselten bereitgestellten Aurora-DB-Cluster erstellen.
  - Das kontoübergreifende Klonen unterstützt derzeit das Klonen von Aurora Serverless v1-DB-Clustern nicht. Weitere Informationen finden Sie unter Einschränkungen für kontoübergreifendes Klonen.
  - Ein geklonter Aurora Serverless v1-DB-Cluster hat das gleiche Verhalten und die gleichen Einschränkungen wie jeder Aurora Serverless v1-DB-Cluster. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Serverless v1.
  - Aurora Serverless v1-DB-Cluster sind immer verschlüsselt. Wenn Sie einen Aurora Serverless v1-DB-Cluster in einen bereitgestellten Aurora-DB-Cluster klonen, wird der bereitgestellte Aurora-DB-Cluster verschlüsselt. Sie können den Verschlüsselungsschlüssel auswählen, aber Sie können die Verschlüsselung nicht deaktivieren. Zum Klonen aus einem bereitgestellten Aurora-DB-Cluster in Aurora Serverless v1 müssen Sie mit einem verschlüsselten bereitgestellten Aurora-DB-Cluster beginnen.

Funktionsweise des Klonens von Aurora

Das Klonen von Aurora funktioniert auf der Speicherschicht eines Aurora-DB-Clusters. Es verwendet ein Copy-on-Write-Protokoll, das sowohl schnell als auch platzsparend in Bezug auf die

zugrunde liegenden langlebigen Medien ist, die das Aurora-Speichervolumen unterstützen. Weitere Informationen zu Aurora-Cluster-Volumes finden Sie unterÜbersicht über Amazon-Aurora-Speicher.

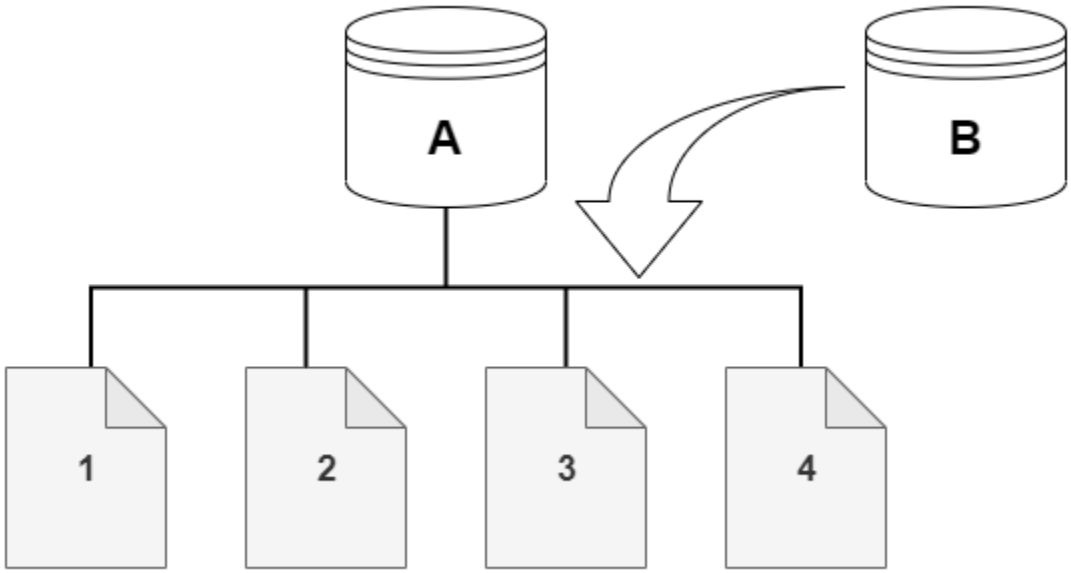
Themen

- Copy-On-Write-Protokoll verstehen
- Löschen eines Quell-Cluster-Volumes

Copy-On-Write-Protokoll verstehen

Ein Aurora-DB-Cluster speichert Daten in Seiten im zugrunde liegenden Aurora-Speicher-Volume.

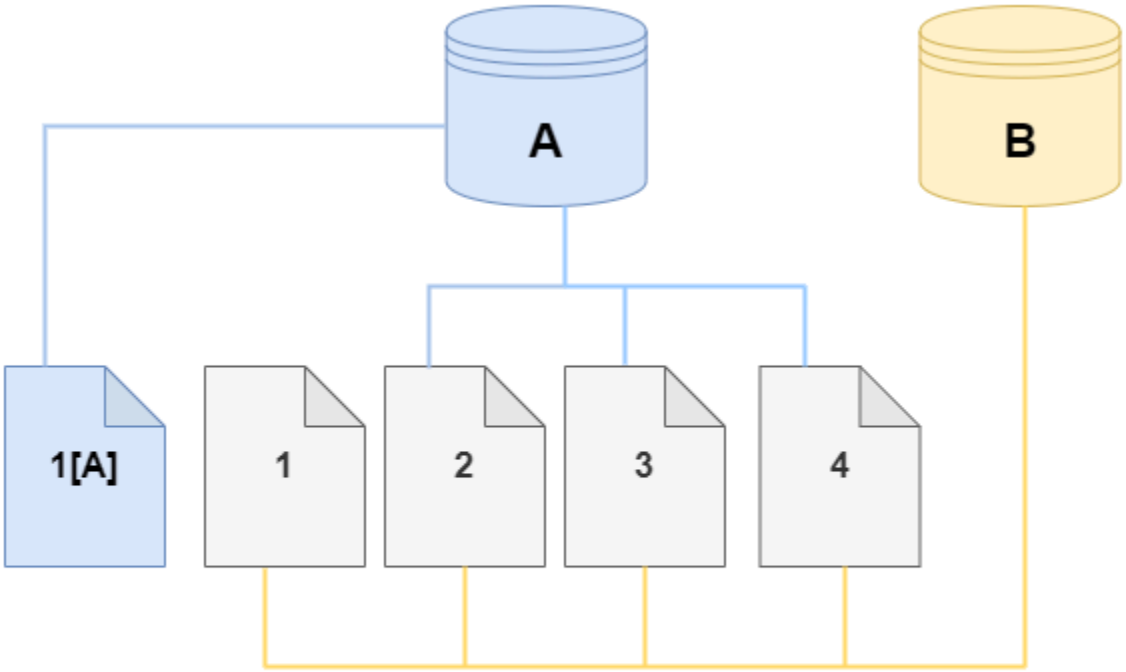
Im folgenden Diagramm finden Sie beispielsweise einen Aurora-DB-Cluster (A) mit vier Datenseiten 1, 2, 3 und 4. Stellen Sie sich vor, dass ein Klon, Klon B, aus dem Aurora-DB-Cluster erstellt wird. Wenn der Klon erstellt wird, werden keine Daten kopiert. Stattdessen verweist der Klon auf denselben Seitensatz wie der Aurora-DB-Quellcluster.



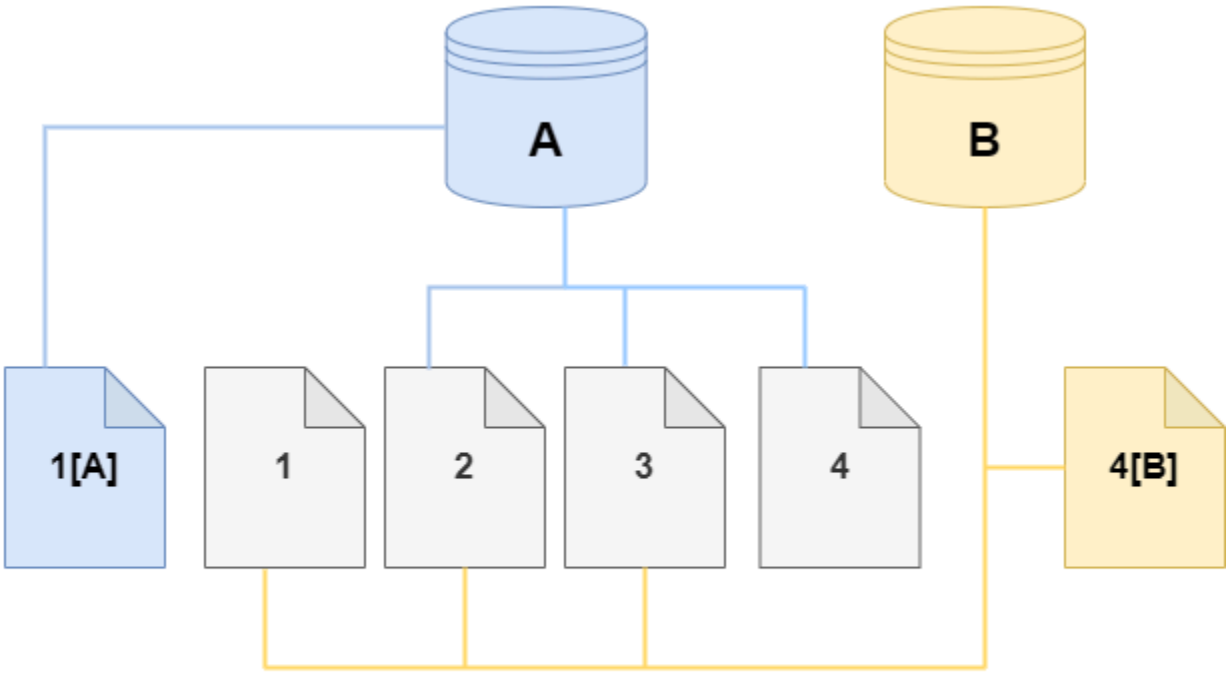
Wenn der Klon erstellt wird, ist normalerweise kein zusätzlicher Speicher erforderlich. Das Copy-on-Write-Protokoll verwendet dasselbe Segment auf dem physischen Speichermedium wie das Quellsegment. Zusätzlicher Speicher ist nur erforderlich, wenn die Kapazität des Quellsegments für das gesamte Klonsegment nicht ausreicht. Wenn dies der Fall ist, wird das Quellsegment auf ein anderes physisches Gerät kopiert.

In den folgenden Diagrammen finden Sie ein Beispiel für das Copy-on-Write-Protokoll in Aktion mit demselben Cluster A und dessen Klon, B, wie oben gezeigt. Nehmen wir an, Sie nehmen eine

Änderung an Ihrem Aurora-DB-Cluster (A) vor, was zu einer Änderung der Daten auf Seite 1 führt. Anstatt auf die ursprüngliche Seite 1 zu schreiben, erstellt Aurora eine neue Seite 1 [A]. Das Aurora-DB-Cluster-Volume für Cluster (A) verweist nun auf Seite 1 [A], 2, 3 und 4, während der Klon (B) weiterhin auf die ursprünglichen Seiten verweist.



Auf dem Klon wird eine Änderung an Seite 4 auf dem Speichervolume vorgenommen. Anstatt auf die ursprüngliche Seite 4 zu schreiben, erstellt Aurora eine neue Seite, 4 [B]. Der Klon verweist nun auf die Seiten 1, 2, 3 und auf Seite 4[B], während der Cluster (A) weiterhin auf 1[A], 2, 3 und 4 verweist.



Da im Laufe der Zeit mehr Änderungen sowohl im Aurora-DB-Cluster-Quellvolume als auch im Klon auftreten, wird mehr Speicherplatz benötigt, um die Änderungen zu erfassen und zu speichern.

Löschen eines Quell-Cluster-Volumes

Anfänglich verwendet das Klon-Volume dieselben Datenseiten wie das ursprüngliche Volume, aus dem der Klon erstellt wurde. Solange das ursprüngliche Volume existiert, gilt das Klon-Volume nur als Eigentümer der Seiten, die der Klon erstellt oder geändert hat. Die `VolumeBytesUsed`-Metrik für das Klon-Volume ist also zunächst gering und steigt nur, wenn die Daten zwischen dem ursprünglichen Cluster und dem Klon voneinander abweichen. Für Seiten, die beim Quell-Volume und dem Klon identisch sind, gelten die Speichergebühren nur für den ursprünglichen Cluster. Für weitere Informationen über die `VolumeBytesUsed`-Metrik siehe Metriken auf Cluster-Ebene für Amazon Aurora.

Wenn Sie ein Quell-Cluster-Volume löschen, dem ein oder mehrere Klone zugeordnet sind, werden die Daten in den Cluster-Volumes der Klone nicht geändert. Aurora behält die Seiten bei, die zuvor im Besitz des Quell-Cluster-Volumes waren. Aurora verteilt die Speicherabrechnung für die Seiten, die

dem gelöschten Cluster gehörten, neu. Nehmen wir zum Beispiel an, ein ursprünglicher Cluster hatte zwei Klone und dann wurde der ursprüngliche Cluster gelöscht. Die Hälfte der Datenseiten, die dem ursprünglichen Cluster gehörten, würde jetzt einem Klon gehören. Die andere Hälfte der Seiten würde dem anderen Klon gehören.

Wenn Sie den ursprünglichen Cluster löschen und weitere Klone erstellen oder löschen, verteilt Aurora weiterhin den Besitz der Datenseiten auf alle Klone, die dieselben Seiten gemeinsam nutzen. Daher stellen Sie möglicherweise fest, dass sich der Wert der `VolumeBytesUsed`-Metrik für das Cluster-Volume eines Klons ändert. Der Metrikwert kann sinken, wenn mehr Klone erstellt werden und der Seitenbesitz auf mehr Cluster verteilt wird. Der Metrikwert kann auch steigen, wenn Klone gelöscht werden und der Seitenbesitz einer kleineren Anzahl von Clustern zugewiesen wird. Informationen dazu, wie sich Schreiboperationen auf Datenseiten auf Klon-Volumes auswirken, finden Sie unter Copy-On-Write-Protokoll verstehen.

Wenn der ursprüngliche Cluster und die Klone demselben AWS-Konto gehören, fallen alle Speichergebühren für diese Cluster bei diesem AWS-Konto an. Wenn es sich bei einigen Clustern um kontenübergreifende Klone handelt, kann das Löschen des ursprünglichen Clusters zu zusätzlichen Speichergebühren für die AWS-Konten führen, denen die kontenübergreifenden Klone gehören.

Nehmen wir beispielsweise an, dass ein Cluster-Volume 1 000 verwendete Datenseiten hat, bevor Sie Klone erstellen. Wenn Sie diesen Cluster klonen, hat das Klon-Volume zunächst keine verwendeten Seiten. Wenn der Klon Änderungen an 100 Datenseiten vornimmt, werden nur diese 100 Seiten auf dem Klon-Volume gespeichert und als verwendet markiert. Die anderen 900 unveränderten Seiten des übergeordneten Volumes werden von beiden Clustern gemeinsam genutzt. In diesem Fall fallen für den übergeordneten Cluster Speichergebühren für 1 000 Seiten und für das Klon-Volume für 100 Seiten an.

Wenn Sie das Quell-Volume löschen, umfassen die Speichergebühren für den Klon die 100 Seiten, die er geändert hat, plus die 900 gemeinsam genutzten Seiten aus dem ursprünglichen Volume, also insgesamt 1 000 Seiten.

Erstellen eines Amazon-Aurora-DB-Klons

Sie können einen Klon im selben AWS-Konto wie den Aurora-DB-Quellcluster erstellen. Dazu können Sie das AWS-Managementkonsole oder das AWS CLI und die folgenden Verfahren verwenden.

Um einem anderen AWS-Konto zu erlauben, einen Klon zu erstellen oder einen Klon mit einem anderen AWS-Konto zu teilen, verwenden Sie die Verfahren in Kontoübergreifendes Klonen mit AWS RAM und Amazon Aurora.

Konsole

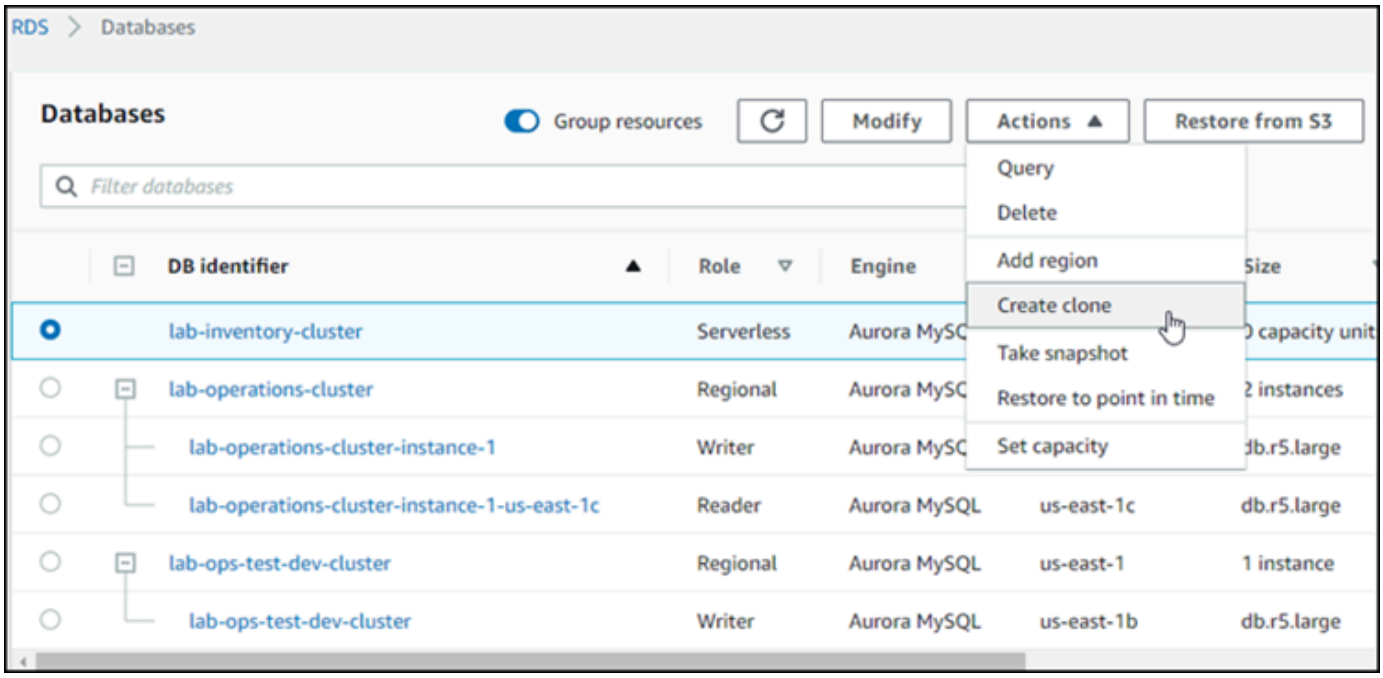
Das folgende Verfahren beschreibt das Klonen eines Aurora-DB-Clusters mittels der AWS-Managementkonsole.

Erstellen eines Klons unter Verwendung der AWS-Managementkonsole-Ergebnisse in einem Aurora-DB-Cluster mit einer Aurora-DB-Instance.

Diese Anweisungen gelten für DB-Cluster, die sich im Besitz desselben AWS-Kontos befinden, das den Klon erstellt. Wenn der DB-Cluster einem anderenAWS-Konto gehört, lesen Sie stattdessen Kontoübergreifendes Klonen mit AWS RAM und Amazon Aurora.

So erstellen Sie mit AWS einen Klon eines DB-Clusters, der Ihrem AWS-Managementkonsole-Konto gehört:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie Ihren Aurora-DB-Cluster aus der Liste aus, und wählen Sie für Aktionen die Option Klon erstellen aus.



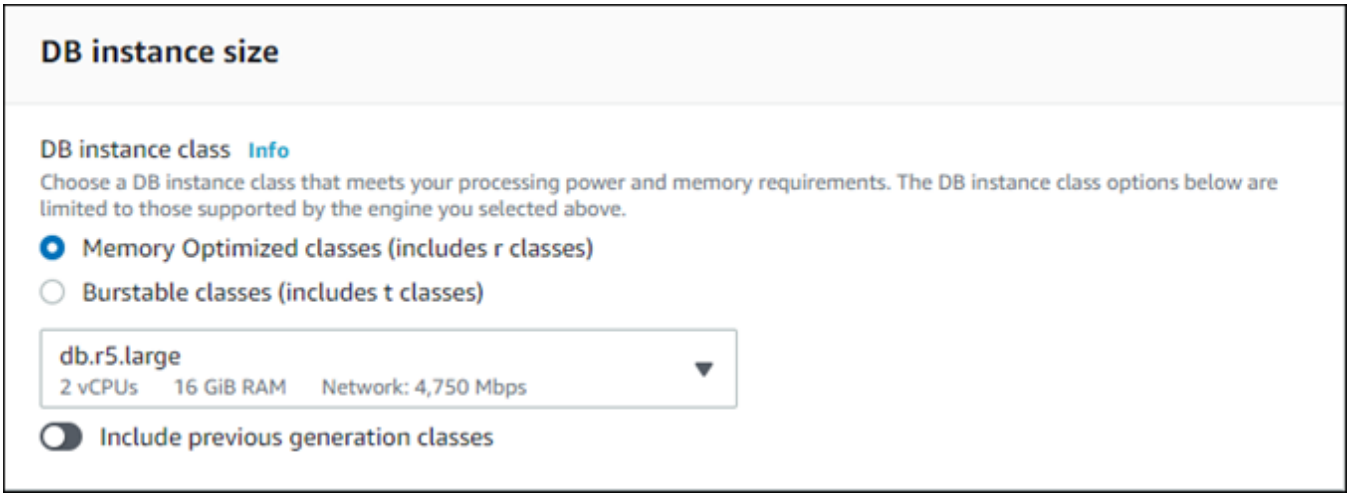
Die Seite „Klon erstellen“ wird geöffnet, auf der Sie Einstellungen, Konnektivität und andere Optionen für den Klon des Aurora-DB-Clusters konfigurieren können.

4. Geben Sie als DB-Instance-ID den Namen ein, den Sie Ihrem geklonten Aurora-DB-Cluster geben möchten.
5. Wählen Sie für Aurora Serverless v1-DB-Cluster als Kapazitätstyp die Option Bereitgestellt oder Serverless aus.

   Sie können Serverless nur auswählen, wenn der Aurora-DB-Quellcluster ein Aurora Serverless v1-DB-Cluster oder ein bereitgestellter Aurora-DB-Cluster ist, der verschlüsselt ist.
6. Wählen Sie für Aurora Serverless v2- oder bereitgestellte DB-Cluster entweder Aurora I/O-Optimized oder Aurora Standard für Konfiguration des Cluster-Speichers aus.

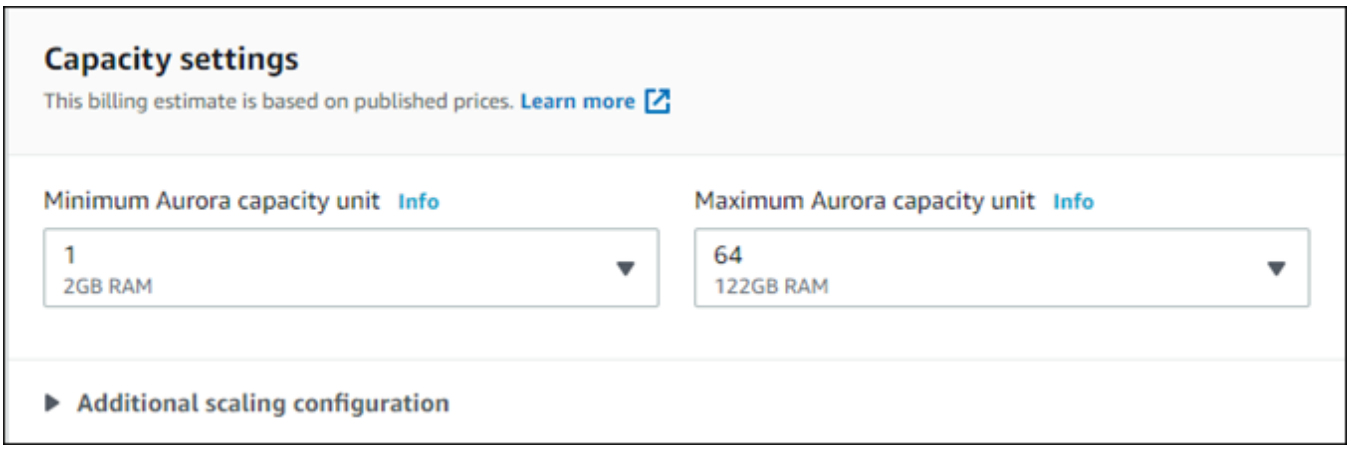
   Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.
7. Wählen Sie die Größe der DB-Instance oder die DB-Cluster-Kapazität aus:

   - Wählen Sie für einen bereitgestellten Klon eine DB-Instance-Klasse aus.



     Sie können die bereitgestellte Einstellung akzeptieren oder eine andere DB-Instance-Klasse für Ihren Klon verwenden.

   - Wählen Sie für einen Aurora Serverless v1- oder Aurora Serverless v2-Klon die Kapazitätseinstellungen aus.



Sie können die bereitgestellten Einstellungen übernehmen oder für Ihren Klon ändern.

8. Wählen Sie nach Bedarf weitere Einstellungen für Ihren Klon aus. Weitere Informationen zu Aurora DB-Cluster- und -Instance-Einstellungen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.
9. Wählen Sie Klon erstellen aus.

Wenn der Klon erstellt wird, wird er mit Ihren anderen Aurora-DB-Clustern im Abschnitt Datenbanken der Konsole aufgelistet und zeigt seinen aktuellen Status an. Ihr Klon ist einsatzbereit, wenn sein Status Verfügbar ist.

AWS CLI

Wenn Sie die AWS CLI zum Klonen Ihres Aurora-DB-Clusters verwenden, sind separate Schritte zum Erstellen des Klon-Clusters und zum Hinzufügen einer oder mehrerer DB-Instances erforderlich.

Der von Ihnen verwendete AWS CLI-Befehl `restore-db-cluster-to-point-in-time` führt zu einem Aurora-DB-Cluster mit denselben Speicherdaten wie der ursprüngliche Cluster, jedoch ohne Aurora-DB-Instances. Sie erstellen die DB-Instances separat, nachdem der Klon verfügbar ist. Sie können die Anzahl der DB-Instances und ihrer Instance-Klassen wählen, um dem Klon mehr oder weniger Rechenkapazität als der ursprüngliche Cluster zuzuweisen. Der Prozess umfasst folgende Schritte:

1. Erstellen Sie den Klon mit dem CLI-Befehl restore-db-cluster-to-point-in-time.
2. Erstellen Sie die Writer-DB-Instance für den Klon mit dem CLI-Befehl create-db-instance.
3. (Optional) Führen Sie zusätzliche create-db-instance-CLI-Befehle aus, um dem Klon-Cluster eine oder mehrere Reader-Instances hinzuzufügen. Die Verwendung von Reader-Instances trägt dazu bei, die Hochverfügbarkeit und die Leseskalierbarkeit des Klons zu verbessern. Sie können

diesen Schritt überspringen, wenn Sie den Klon nur für Entwicklungs- und Testzwecke verwenden möchten.

Themen

- Erstellen des Klons
- Überprüfen des Status und Abrufen von Klon-Details
- Erstellen einer Aurora-DB-Instance für Ihren Klon
- Zu verwendende Parameter für das Klonen

Erstellen des Klons

Verwenden Sie den CLI-Befehl `restore-db-cluster-to-point-in-time`, um den ersten Klon-Cluster zu erstellen.

So erstellen Sie einen Klon aus einem Aurora-DB-Quellcluster

- Verwenden Sie den `restore-db-cluster-to-point-in-time`-CLI-Befehl. Geben Sie Werte für die folgenden Parameter an. In diesem typischen Fall verwendet der Klon denselben Engine-Modus wie der ursprüngliche Cluster, entweder bereitgestellt oder Aurora Serverless v1.
  - `--db-cluster-identifier` – Wählen Sie einen aussagekräftigen Namen für Ihren Klon. Sie benennen den Klon, wenn Sie den CLI-Befehl restore-db-cluster-to-point-in-time verwenden. Anschließend übergeben Sie den Namen des Klons im CLI-Befehl create-db-instance.
  - `--restore-type` – Verwenden Sie `copy-on-write`, um einen Klon des Quell-DB-Clusters zu erstellen. Ohne diesen Parameter stellt `restore-db-cluster-to-point-in-time` den Aurora-DB-Cluster wieder her, anstatt einen Klon zu erstellen.
  - `--source-db-cluster-identifier` – Verwenden Sie den Namen des Aurora-DB-Quell-Clusters, den Sie klonen möchten.
  - `--use-latest-restorable-time` – Dieser Wert verweist auf die neuesten wiederherstellbaren Volume-Daten für den Quell-DB-Cluster. Verwenden Sie ihn, um Klone zu erstellen.

Im folgenden Beispiel wird ein Klon namens `my-clone` aus einem Cluster namens `my-source-cluster` erstellt.

Linux, macOS oder Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Für Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

Der Befehl gibt das JSON-Objekt zurück, das Details des Klons enthält. Stellen Sie sicher, dass Ihr geklonter DB-Cluster verfügbar ist, bevor Sie versuchen, die DB-Instance für Ihren Klon zu erstellen. Weitere Informationen finden Sie unter Überprüfen des Status und Abrufen von Klon-Details.

Angenommen, Sie haben einen Cluster mit dem Namen `tpch100g`, den Sie klonen möchten. Im folgenden Linux-Beispiel werden ein geklonter Cluster namens `tpch100g-clone`, eine Aurora Serverless v2-Writer-Instance mit dem Namen `tpch100g-clone-instance` sowie eine bereitgestellte Reader-Instance namens `tpch100g-clone-instance-2` für den neuen Cluster erstellt.

Einige Parameter wie `--master-username` und `--master-user-password` müssen nicht angegeben werden. Aurora ermittelt diese automatisch aus dem ursprünglichen Cluster. Sie müssen die zu verwendende DB-Engine angeben. Daher testet das Beispiel den neuen Cluster, um den richtigen Wert für den Parameter `--engine` zu ermitteln.

Dieses Beispiel beinhaltet auch die `--serverless-v2-scaling-configuration`-Option beim Erstellen des Klon-Clusters. Auf diese Weise können Sie dem Klon Aurora Serverless v2-Instances hinzufügen, auch wenn der ursprüngliche Cluster Aurora Serverless v2 nicht verwendet hat.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16 \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

```
$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
    --query '*[].[Engine]' \
    --output text
aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.serverless \
  --engine aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance-2 \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.r6g.2xlarge \
  --engine aurora-mysql
```

So erstellen Sie einen Klon mit einem anderen Engine-Modus als der Aurora-DB-Cluster

- Dieses Verfahren gilt nur für ältere Engine-Versionen, die Aurora Serverless v1 unterstützen. Angenommen, Sie haben einen Aurora Serverless v1-Cluster und möchten einen Klon erstellen, bei dem es sich um einen bereitgestellten Cluster handelt. Verwenden Sie in diesem Fall den CLI-Befehl `restore-db-cluster-to-point-in-time` und geben Sie ähnliche Parameterwerte wie im vorherigen Beispiel sowie diese zusätzlichen Parameter an:

  - `--engine-mode` – Verwenden Sie diesen Parameter nur, um Klone zu erstellen, die einen anderen Engine-Modus als der Aurora-DB-Quellcluster verwenden. Dieser Parameter gilt nur für ältere Engine-Versionen, die Aurora Serverless v1 unterstützen. Wählen Sie den mit `--engine-mode` zu übergebenden Wert wie folgt aus:

    - Verwenden Sie `--engine-mode provisioned`, um einen bereitgestellten Aurora-DB-Clusterklon aus einem Aurora Serverless-DB-Cluster zu erstellen.

      > **Note**
      >
      > Wenn Sie beabsichtigen, Aurora Serverless v2 mit einem Cluster zu verwenden, der aus Aurora Serverless v1 geklont wurde, geben Sie den Engine-Modus für den

Klon dennoch als `provisioned` an. Anschließend führen Sie weitere Upgrade- und Migrationsschritte durch.

- Verwenden Sie `--engine-mode serverless`, um einen Aurora Serverless v1-Klon aus einem bereitgestellten Aurora-DB-Cluster zu erstellen. Wenn Sie den `serverless`-Engine-Modus angeben, können Sie auch `--scaling-configuration` wählen.

- `--scaling-configuration` – (Optional) Verwenden Sie dies mit `--engine-mode serverless`, um die minimale und maximale Kapazität für einen Aurora Serverless v1-Klon zu konfigurieren. Wenn Sie diesen Parameter nicht verwenden, erstellt Aurora einen Aurora Serverless v1-Klon mit den Aurora Serverless v1-Standardkapazitätswerten für die DB-Engine.

Im folgenden Beispiel wird ein bereitgestellter Klon namens `my-clone` aus einem Aurora Serverless v1-DB-Cluster namens `my-source-cluster` erstellt.

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --engine-mode provisioned \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Für Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --engine-mode provisioned ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

Diese Befehle geben das JSON-Objekt zurück, das Details des Klons enthält, den Sie zum Erstellen der DB-Instance benötigen. Dies ist erst möglich, wenn der Status des Klons (der leere Aurora-DB-Cluster) den Status Verfügbar hat.

> **Note**
>
> Der AWS-CLI-Befehl restore-db-cluster-to-point-in-time stellt nur das DB-Cluster wieder her, nicht die DB-Instances für dieses DB-Cluster. Um DB-Instances für den wiederhergestellten DB-Cluster zu erstellen, führen Sie den Befehl create-db-instance aus. Mit diesem Befehl geben Sie die ID des wiederhergestellten DB-Clusters als `--db-cluster-identifier`-Parameter an. Sie können DB-Instances erst dann erstellen, wenn der Befehl `restore-db-cluster-to-point-in-time` abgeschlossen ist und der DB-Cluster verfügbar ist. Angenommen, Sie beginnen mit einem Aurora Serverless v1-Cluster und beabsichtigen, ihn zu einem Aurora Serverless v2-Cluster zu migrieren. Als ersten Schritt der Migration erstellen Sie einen bereitgestellten Klon des Aurora Serverless v1-Clusters. Das vollständige Verfahren, einschließlich aller erforderlichen Versionsupgrades, finden Sie unter Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2.

Überprüfen des Status und Abrufen von Klon-Details

Mit dem folgenden Befehl können Sie den Status Ihres neu erstellten Klon-Clusters überprüfen.

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]'
 --output text
```

Oder Sie können den Status und die anderen Werte abrufen, die Sie zum Erstellen der DB-Instance für Ihren Klon benötigen, indem Sie die folgende AWS CLI-Abfrage verwenden.

Für Linux, macOS oder Unix:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

Für Windows:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

Diese Abfrage gibt eine Ausgabe ähnlich der folgenden zurück.

```
[
  {
        "Status": "available",
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.04.1",
        "EngineMode": "provisioned"
    }
]
```

Erstellen einer Aurora-DB-Instance für Ihren Klon

Verwenden Sie den CLI-Befehl create-db-instance, um die DB-Instance für Ihren Aurora Serverless v2- oder bereitgestellten Klon zu erstellen. Sie erstellen keine DB-Instance für einen Aurora Serverless v1-Klon.

Die DB-Instance erbt die `--master-username`- und `--master-user-password`-Eigenschaft vom Quell-DB-Cluster.

Im folgenden Beispiel wird eine DB-Instance für einen bereitgestellten Klon erstellt.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.r6g.2xlarge \
    --engine aurora-mysql
```

Für Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.r6g.2xlarge ^
    --engine aurora-mysql
```

Im folgenden Beispiel wird eine Aurora Serverless v2-DB-Instance für einen Klon erstellt, der eine Engine-Version verwendet, die Aurora Serverless v2 unterstützt.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
```

```
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.serverless \
  --engine aurora-postgresql
```

Für Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.serverless ^
    --engine aurora-mysql
```

Zu verwendende Parameter für das Klonen

Die folgende Tabelle fasst die verschiedenen Parameter zusammen, die mit `restore-db-cluster-to-point-in-time` zum Klonen von Aurora-DB-Clustern verwendet werden.

Parameter	Beschreibung
`--source-db-cluster-identifier`	Verwenden Sie den Namen des Aurora-DB-Quell-Clusters, den Sie klonen möchten.
`--db-cluster-identifier`	Wählen Sie einen aussagekräftigen Namen für Ihren Klon, wenn Sie ihn mit dem `restore-db-cluster-to-point-in-time` -Befehl erstellen. Dann übergeben Sie diesen Namen an den `create-db-instance` -Befehl.
`--restore-type`	Geben Sie `copy-on-write` als `--restore-type` an, um einen Klon des Quell-DB-Clusters zu erstellen, anstatt den Quell-Aurora-DB-Cluster wiederherzustellen.
`--use-latest-restorable-time`	Dieser Wert verweist auf die neuesten wiederherstellbaren Volume-Daten für den Quell-DB-Cluster. Verwenden Sie ihn, um Klone zu erstellen.
`--serverless-v2-scaling-configuration`	(Neuere Versionen, die Aurora Serverless v2 unterstützen) Verwenden Sie diesen Parameter, um die minimale und maximale Kapazität für einen Aurora Serverless v2-Klon zu konfigurieren. Wenn Sie diesen

Parameter	Beschreibung
	Parameter nicht angeben, können Sie keine Aurora Serverless v2-Instan ces im Klon-Cluster erstellen, bis Sie den Cluster so ändern, dass dieses Attribut hinzugefügt wird.
`--engine-mode`	(Nur ältere Versionen, die Aurora Serverless v1 unterstützen) Verwenden Sie diesen Parameter, um Klone zu erstellen, deren Typ von dem des Aurora-DB-Quellcluster abweicht. Verwenden Sie einen der folgenden Werte: • Verwenden Sie `provisioned` , um einen bereitgestellten Klon aus einem Aurora Serverless v1-DB-Cluster zu erstellen. • Verwenden Sie `serverless` , um einen Aurora Serverless v1-Klon aus einem bereitgestellten oder Aurora Serverless v2-DB-Cluster zu erstellen. Wenn Sie den `serverless` -Engine-Modus angeben, können Sie auch `--scaling-configuration` wählen.
`--scaling-configuration`	(Nur ältere Versionen, die Aurora Serverless v1 unterstützen) Verwenden Sie diesen Parameter, um die minimale und maximale Kapazität für einen Aurora Serverless v1-Klon zu konfigurieren. Wenn Sie diesen Parameter nicht angeben, erstellt Aurora den Klon mit den Standardk apazitätswerten für die DB-Engine.

Informationen zu VPC- und kontoübergreifendem Klonen finden Sie in den folgenden Abschnitten.

Themen

- VPC-übergreifendes Klonen mit Amazon Aurora
- Kontoübergreifendes Klonen mit AWS RAM und Amazon Aurora

VPC-übergreifendes Klonen mit Amazon Aurora

Angenommen, Sie möchten für den ursprünglichen Cluster und den Klon unterschiedliche Netzwerkzugriffskontrollen festlegen. Sie können beispielsweise durch Klonen eine Kopie eines Aurora-Produktionsclusters in einer anderen VPC für Entwicklung und Tests erstellen. Oder Sie

können einen Klon als Teil einer Migration von öffentlichen Subnetzen zu privaten Subnetzen erstellen, um die Datenbanksicherheit zu erhöhen.

In den folgenden Abschnitten wird gezeigt, wie die Netzwerkkonfiguration für den Klon so eingerichtet wird, dass der ursprüngliche Cluster und der Klon beide auf dieselben Aurora-Speicherknoten zugreifen können, auch von unterschiedlichen Subnetzen oder unterschiedlichen VPCs aus. Durch die vorherige Überprüfung der Netzwerkressourcen können Fehler beim Klonen vermieden werden, die möglicherweise schwer zu diagnostizieren sind.

Wenn Sie nicht wissen, wie Aurora mit VPCs, Subnetzen und DB-Subnetzgruppen interagiert, informieren Sie sich zunächst unter Amazon VPC und Amazon Aurora. Sie können die Tutorials in diesem Abschnitt durcharbeiten, um diese Art von Ressourcen in der AWS-Konsole zu erstellen und zu verstehen, wie sie zusammenpassen.

Da die Schritte das Umschalten zwischen den RDS- und EC2-Diensten beinhalten, verwenden die Beispiele AWS-CLI-Befehle, damit Sie besser verstehen können, wie Sie solche Operationen automatisieren und die Ausgabe speichern.

Themen

- Bevor Sie beginnen
- Sammeln von Informationen über die Netzwerkumgebung
- Erstellen von Netzwerkressourcen für den Klon
- Erstellen eines Aurora-Klons mit neuen Netzwerkeinstellungen
- Verschieben eines Clusters aus öffentlichen Subnetzen in private
- Umfassendes Beispiel für das Erstellen eines VPC-Klons

Bevor Sie beginnen

Stellen Sie vor dem Einrichten eines VPC-übergreifenden Klons sicher, dass Sie über die folgenden Ressourcen verfügen:

- ein Aurora-DB-Cluster, der als Quelle für das Klonen verwendet werden soll. Wenn Sie zum ersten Mal einen Aurora-DB-Cluster erstellen, lesen Sie die Tutorials unter Erste Schritte mit Amazon Aurora, um einen Cluster mit der MySQL- oder der PostgreSQL-Datenbank-Engine einzurichten.
- eine zweite VPC, wenn Sie beabsichtigen, einen VPC-übergreifenden Klon zu erstellen. Wenn Sie keine VPC haben, die Sie für den Klon verwenden können, finden Sie weitere Informationen

unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4) oder Tutorial: Erstellen einer VPC zur Verwendung mit einer DB–einem DB-Cluster (Dual-Stack-Modus).

Sammeln von Informationen über die Netzwerkumgebung

Beim VPC-übergreifenden Klonen kann sich die Netzwerkumgebung zwischen dem ursprünglichen Cluster und seinem Klon erheblich unterscheiden. Bevor Sie den Klon erstellen, sollten Sie Informationen über die VPC, die Subnetze, die DB-Subnetzgruppe und die AZs sammeln und aufzeichnen, die im ursprünglichen Cluster verwendet wurden. Auf diese Weise können Sie die Wahrscheinlichkeit von Problemen minimieren. Wenn ein Netzwerkproblem auftritt, müssen Sie die Aktivitäten zur Problembehandlung nicht unterbrechen, um nach Diagnoseinformationen zu suchen. Die folgenden Abschnitte zeigen CLI-Beispiele zum Sammeln dieser Art von Informationen. Sie können die Details in einem beliebigen Format speichern, auf das Sie bei der Erstellung des Klons und bei der Problembehandlung schnell zugreifen können.

- Schritt 1: Überprüfen der Availability Zones des ursprünglichen Clusters
- Schritt 2: Überprüfen der DB-Subnetzgruppe des ursprünglichen Clusters
- Schritt 3: Überprüfen der Subnetze des ursprünglichen Clusters
- Schritt 4: Überprüfen Sie der Availability Zones der DB-Instances im ursprünglichen Cluster
- Schritt 5: Überprüfen der VPCs, die für den Klon verwendet werden können

Schritt 1: Überprüfen der Availability Zones des ursprünglichen Clusters

Bevor Sie den Klon erstellen, überprüfen Sie, welche AZs der ursprüngliche Cluster für seinen Speicher verwendet. Wie unter Amazon-Aurora-Speicher erklärt, ist der Speicher für jeden Aurora-Cluster genau drei AZs zugeordnet. Da Amazon-Aurora-DB-Cluster die Trennung zwischen Rechenleistung und Speicher nutzt, gilt diese Regel unabhängig davon, wie viele Instances in dem Cluster enthalten sind.

Führen Sie beispielsweise einen CLI-Befehl wie den folgenden aus und ersetzen Sie *my_cluster* durch den Namen Ihres Clusters. Im folgenden Beispiel wird eine alphabetisch nach dem AZ-Namen sortierte Liste erstellt.

```
aws rds describe-db-clusters \
  --db-cluster-identifier my_cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' \
  --output text
```

Das folgende Beispiel zeigt eine Beispielausgabe für den vorhergehenden `describe-db-clusters`-Befehl. Es zeigt, dass der Speicher für den Aurora-Cluster immer drei AZs verwendet.

```
us-east-1c
us-east-1d
us-east-1e
```

Um einen Klon in einer Netzwerkumgebung zu erstellen, die nicht über alle Ressourcen verfügt, um eine Verbindung zu diesen AZs herzustellen, müssen Sie Subnetze erstellen, die mindestens zwei dieser AZs zugeordnet sind. Danach erstellen Sie eine DB-Subnetzgruppe, die diese zwei oder drei Subnetze enthält. In den nachstehenden Beispielen wird die Vorgehensweise dazu veranschaulicht.

Schritt 2: Überprüfen der DB-Subnetzgruppe des ursprünglichen Clusters

Wenn Sie dieselbe Anzahl von Subnetzen für den Klon wie im ursprünglichen Cluster verwenden möchten, können Sie die Anzahl der Subnetze aus der DB-Subnetzgruppe des ursprünglichen Clusters abrufen. Eine Aurora-DB-Subnetzgruppe enthält mindestens zwei Subnetze, die jeweils einer anderen AZ zugeordnet sind. Notieren Sie sich, welchen AZs die Subnetze zugeordnet sind.

Das folgende Beispiel zeigt, wie Sie die DB-Subnetzgruppe des ursprünglichen Clusters finden und dann rückwärts zu den entsprechenden AZs arbeiten. Ersetzen Sie *`my_cluster`* im ersten Befehl durch den Namen Ihres Clusters. Geben Sie im zweiten Befehl für *`my_subnet`* den Namen der DB-Subnetzgruppe an.

```
aws rds describe-db-clusters --db-cluster-identifier my_cluster \
  --query '*[].DBSubnetGroup' --output text

aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetAvailabilityZone.Name]' --output text
```

Die Beispielausgabe könnte für einen Cluster mit einer DB-Subnetzgruppe, die zwei Subnetze enthält, in etwa wie folgt aussehen. In diesem Fall ist `two-subnets` ein Name, der bei der Erstellung der DB-Subnetzgruppe angegeben wurde.

```
two-subnets

us-east-1d
us-east-1c
```

Für einen Cluster, bei dem die DB-Subnetzgruppe drei Subnetze enthält, könnte die Ausgabe wie folgt aussehen.

```
three-subnets

us-east-1f
us-east-1d
us-east-1c
```

Schritt 3: Überprüfen der Subnetze des ursprünglichen Clusters

Wenn Sie weitere Informationen zu den Subnetzen im ursprünglichen Cluster benötigen, führen Sie AWS-CLI-Befehle aus, die den folgenden ähneln. Sie können die Subnetzattribute wie IP-Adressbereiche, Besitzer usw. untersuchen. Auf diese Weise können Sie bestimmen, ob Sie verschiedene Subnetze in derselben VPC verwenden oder Subnetze mit ähnlichen Eigenschaften in einer anderen VPC erstellen möchten.

Suchen Sie die Subnetz-IDs aller Subnetze, die in Ihrer VPC verfügbar sind.

```
aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc \
  --query '*[].[SubnetId]' --output text
```

Suchen Sie die genauen Subnetze, die in Ihrer DB-Subnetzgruppe verwendet werden.

```
aws rds describe-db-subnet-groups --db-subnet-group-name my_subnet_group \
  --query '*[].Subnets[].[SubnetIdentifier]' --output text
```

Geben Sie dann die Subnetze, die Sie untersuchen möchten, in einer Liste an, wie im folgenden Befehl. Geben Sie die Namen Ihrer Subnetze für *my_subnet_1* usw. ein.

```
aws ec2 describe-subnets \
  --subnet-ids '["my_subnet_1","my_subnet2","my_subnet3"]'
```

Das folgende Beispiel zeigt die teilweise Ausgabe eines solchen `describe-subnets`-Befehls. Die Ausgabe zeigt einige der wichtigen Attribute, die Sie für jedes Subnetz sehen können, z. B. die zugehörige AZ und die VPC, zu der es gehört.

```
{
    'Subnets': [
```

```
        {
            'AvailabilityZone': 'us-east-1d',
            'AvailableIpAddressCount': 54,
            'CidrBlock': '10.0.0.64/26',
            'State': 'available',
            'SubnetId': 'subnet-000a0bca00e0b0000',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
        },
        {
            'AvailabilityZone': 'us-east-1c',
            'AvailableIpAddressCount': 55,
            'CidrBlock': '10.0.0.0/26',
            'State': 'available',
            'SubnetId': 'subnet-4b4dbfe4d4a4fd4c4',
            'VpcId': 'vpc-3f3c3fc3333b3ffb3',
            ...
```

Schritt 4: Überprüfen Sie der Availability Zones der DB-Instances im ursprünglichen Cluster

Sie können dieses Verfahren verwenden, um nachzuvollziehen, welche AZs für die DB-Instances im ursprünglichen Cluster verwendet werden. Auf diese Weise können Sie exakt dieselben AZs für die DB-Instances im Klon einrichten. Sie können auch mehr oder weniger DB-Instances im Klon verwenden, je nachdem, ob der Klon für Produktion, Entwicklung und Tests verwendet wird usw.

Führen Sie für jede Instance im ursprünglichen Cluster einen Befehl wie den folgenden aus. Stellen Sie sicher, dass die Instance den Erstellungsvorgang abgeschlossen hat und sich im Status `Available` befindet. Geben Sie die Instance-ID für *`my_instance`* ein.

```
aws rds describe-db-instances --db-instance-identifier my_instance \
  --query '*[].AvailabilityZone' --output text
```

Das folgende Beispiel zeigt die Ausgabe nach dem Ausführen des `describe-db-instances`-Befehls. Der Aurora-Cluster hat vier Datenbank-Instances. Daher führen wir den Befehl viermal aus und geben dabei jedes Mal eine andere DB-Instance-ID an. Die Ausgabe zeigt, wie diese DB-Instances auf maximal drei AZs verteilt sind.

```
us-east-1a
us-east-1c
us-east-1d
us-east-1a
```

Wenn der Klon erstellt wurde und Sie ihm DB-Instances hinzufügen, können Sie dieselben AZ-Namen in den `create-db-instance`-Befehlen angeben. Sie können dies tun, um DB-Instances im neuen Cluster einzurichten, die für genau dieselben AZs wie im ursprünglichen Cluster konfiguriert sind.

Schritt 5: Überprüfen der VPCs, die für den Klon verwendet werden können

Wenn Sie beabsichtigen, den Klon in einer anderen VPC als der ursprünglichen zu erstellen, können Sie eine Liste der für Ihr Konto verfügbaren VPC-IDs abrufen. Sie können diesen Schritt auch ausführen, wenn Sie zusätzliche Subnetze in derselben VPC wie der ursprüngliche Cluster erstellen müssen. Wenn Sie den Befehl zum Erstellen eines Subnetzes ausführen, geben Sie die VPC-ID als Parameter an.

Führen Sie den folgenden CLI-Befehl aus, um alle VPCs für Ihr Konto aufzuführen:

```
aws ec2 describe-vpcs --query '*[].[VpcId]' --output text
```

Das folgende Beispiel zeigt eine Beispielausgabe für den vorhergehenden `describe-vpcs`-Befehl. Die Ausgabe zeigt, dass das aktuelle AWS-Konto vier VPCs enthält, die als Quelle oder Ziel für das VPC-übergreifende Klonen verwendet werden können.

```
vpc-fd111111
vpc-2222e2cd2a222f22e
vpc-33333333a33333d33
vpc-4ae4d4de4a4444dad
```

Sie können dieselbe VPC als Ziel für den Klon oder eine andere VPC verwenden. Wenn sich der ursprüngliche Cluster und der Klon in derselben VPC befinden, können Sie dieselbe DB-Subnetzgruppe für den Klon wiederverwenden. Sie können auch eine andere DB-Subnetzgruppe erstellen. Beispielsweise könnte die neue DB-Subnetzgruppe private Subnetze verwenden, während die DB-Subnetzgruppe des ursprünglichen Clusters öffentliche Subnetze verwenden könnte. Wenn Sie den Klon in einer anderen VPC erstellen, stellen Sie sicher, dass in der neuen VPC genügend Subnetze vorhanden sind und dass die Subnetze den richtigen AZs aus dem ursprünglichen Cluster zugeordnet sind.

Erstellen von Netzwerkressourcen für den Klon

Wenn Sie beim Sammeln der Netzwerkinformationen festgestellt haben, dass zusätzliche Netzwerkressourcen für den Klon benötigt werden, können Sie diese Ressourcen erstellen, bevor

Sie versuchen, den Klon einzurichten. Beispielsweise müssen Sie möglicherweise mehr Subnetze, bestimmten AZs zugeordnete Subnetze oder eine neue DB-Subnetzgruppe erstellen.

- Schritt 1: Erstellen der Subnetze für den Klon
- Schritt 2: Erstellen der DB-Subnetzgruppe für den Klon

Schritt 1: Erstellen der Subnetze für den Klon

Wenn Sie neue Subnetze für den Klon erstellen müssen, führen Sie einen Befehl wie den folgenden aus. Möglicherweise müssen Sie dies tun, wenn Sie den Klon in einer anderen VPC erstellen oder wenn Sie eine andere Netzwerkänderung vornehmen und z. B. private Subnetze anstelle von öffentlichen Subnetzen verwenden.

AWS generiert automatisch die ID des Subnetzes. Geben Sie den Namen der VPC des Klons für *my_vpc* ein. Wählen Sie den Adressbereich für die `--cidr-block`-Option so aus, dass mindestens 16 IP-Adressen in dem Bereich zugelassen werden. Sie können alle anderen Eigenschaften einbeziehen, die Sie angeben möchten. Führen Sie den Befehl `aws ec2 create-subnet help` aus, um alle Auswahlmöglichkeiten zu sehen.

```
aws ec2 create-subnet --vpc-id my_vpc \
  --availability-zone AZ_name --cidr-block IP_range
```

Das folgende Beispiel zeigt einige wichtige Attribute eines neu erstellten Subnetzes.

```
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1b',
        'AvailableIpAddressCount': 59,
        'CidrBlock': '10.0.0.64/26',
        'State': 'available',
        'SubnetId': 'subnet-44b4a44f4e44db444',
        'VpcId': 'vpc-555fc5df555e555dc',
        ...
        }
}
```

Schritt 2: Erstellen der DB-Subnetzgruppe für den Klon

Wenn Sie den Klon in einer anderen VPC oder eine andere Gruppe von Subnetzen innerhalb derselben VPC erstellen, erstellen Sie eine neue DB-Subnetzgruppe und geben sie bei der Erstellung des Klons an.

Stellen Sie sicher, dass Ihnen alle folgenden Details bekannt sind. Sie können all diese Informationen in der Ausgabe der vorherigen Beispiele finden.

1. VPC des ursprünglichen Clusters. Detaillierte Anweisungen finden Sie unter Schritt 3: Überprüfen der Subnetze des ursprünglichen Clusters.
2. VPC des Klons, wenn Sie ihn in einer anderen VPC erstellen. Detaillierte Anweisungen finden Sie unter Schritt 5: Überprüfen der VPCs, die für den Klon verwendet werden können.
3. drei AZs, die dem Aurora-Speicher für den ursprünglichen Cluster zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 1: Überprüfen der Availability Zones des ursprünglichen Clusters.
4. zwei oder drei AZs, die der DB-Subnetzgruppe für den ursprünglichen Cluster zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 2: Überprüfen der DB-Subnetzgruppe des ursprünglichen Clusters.
5. die Subnetz-IDs und zugehörigen AZs aller Subnetze in der VPC, die Sie für den Klon verwenden möchten. Verwenden Sie denselben `describe-subnets`-Befehl wie in Schritt 3: Überprüfen der Subnetze des ursprünglichen Clusters und ersetzen Sie dabei die VPC-ID der Ziel-VPC.

Prüfen Sie, wie viele AZs sowohl dem Speicher des ursprünglichen Clusters als auch den Subnetzen in der Ziel-VPC zugeordnet sind. Um den Klon erfolgreich zu erstellen, müssen sie zwei oder drei AZs gemeinsam haben. Wenn weniger als zwei gemeinsam verwendete AZs vorhanden sind, gehen Sie zurück zu Schritt 1: Erstellen der Subnetze für den Klon. Erstellen Sie ein, zwei oder drei neue Subnetze, die den AZs zugeordnet sind, die dem Speicher des ursprünglichen Clusters zugeordnet sind.

Wählen Sie Subnetze in der Ziel-VPC aus, die denselben AZs zugeordnet sind wie der Aurora-Speicher im ursprünglichen Cluster. Wählen Sie idealerweise drei AZs aus. Auf diese Weise haben Sie die größte Flexibilität, um die DB-Instances des Klons auf mehrere AZs zu verteilen und so eine hohe Verfügbarkeit der Rechenressourcen zu gewährleisten.

Führen Sie einen ähnlichen Befehl wie den folgenden aus, um die neue DB-Subnetzgruppe zu erstellen. Ersetzen Sie die IDs Ihrer Subnetze in der Liste. Wenn Sie die Subnetz-IDs mithilfe

von Umgebungsvariablen angeben, achten Sie darauf, die `--subnet-ids`-Parameterliste so in Anführungszeichen zu setzen, dass die IDs weiter in doppelten Anführungszeichen stehen.

```
aws rds create-db-subnet-group --db-subnet-group-name my_subnet_group \
  --subnet-ids '["my_subnet_1","my_subnet_2","my_subnet3"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets for clone'
```

Das folgende Beispiel zeigt teilweise die Ausgabe des Befehls `create-db-subnet-group`.

```
{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'my_subnet_group',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets for clone',
        'VpcId': 'vpc-555fc5df555e555dc',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'my_subnet_1',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1c'
                },
                'SubnetStatus': 'Active'
            },
            {
                'SubnetIdentifier': 'my_subnet_2',
                'SubnetAvailabilityZone': {
                    'Name': 'us-east-1d'
                },
                'SubnetStatus': 'Active'
            }
            ...
        ],
        'SupportedNetworkTypes': [
            'IPV4'
        ]
    }
}
```

Zu diesem Zeitpunkt haben Sie den Klon noch nicht erstellt. Sie haben alle relevanten VPC- und Subnetzressourcen erstellt, sodass Sie beim Erstellen des Klons die entsprechenden Parameter für die Befehle `restore-db-cluster-to-point-in-time` und `create-db-instance` angeben können.

Erstellen eines Aurora-Klons mit neuen Netzwerkeinstellungen

Sobald Sie sichergestellt haben, dass die richtige Konfiguration von VPCs, Subnetzen, AZs und Subnetzgruppen für den neuen Cluster vorhanden ist, können Sie den eigentlichen Klonvorgang durchführen. In den folgenden CLI-Beispielen werden die Optionen wie `--db-subnet-group-name`, `--availability-zone` und `--vpc-security-group-ids` hervorgehoben, die Sie in den Befehlen zum Einrichten des Klons und seiner DB-Instances angeben.

- Schritt 1: Angeben der DB-Subnetzgruppe für den Klon
- Schritt 2: Angeben der Netzwerkeinstellungen für Instances im Klon
- Schritt 3: Herstellen der Konnektivität zwischen einem Clientsystem und einem Klon

Schritt 1: Angeben der DB-Subnetzgruppe für den Klon

Wenn Sie den Klon erstellen, können Sie die richtigen VPC-, Subnetz- und AZ-Einstellungen konfigurieren, indem Sie eine DB-Subnetzgruppe angeben. Verwenden Sie die Befehle in den vorherigen Beispielen, um alle Beziehungen und Zuordnungen zu überprüfen, die zur DB-Subnetzgruppe gehören.

Die folgenden Befehle veranschaulichen beispielsweise das Klonen eines ursprünglichen Clusters in einen Klon. Im ersten Beispiel ist der Quellcluster mit zwei Subnetzen und der Klon mit drei Subnetzen verknüpft. Das zweite Beispiel zeigt den umgekehrten Fall, nämlich das Klonen von einem Cluster mit drei Subnetzen zu einem Cluster mit zwei Subnetzen.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-3-subnets \
  --db-cluster-identifier cluster-cloned-to-2-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name two-subnets
```

Wenn Sie beabsichtigen, Aurora-Serverless-v2-Instances im Klon zu verwenden, fügen Sie bei der Erstellung des Klons eine `--serverless-v2-scaling-configuration`-Option hinzu, wie hier gezeigt. Auf diese Weise können Sie die `db.serverless`-Klasse verwenden, wenn Sie DB-Instances im Klon erstellen. Sie können den Klon auch später ändern, um dieses Attribut für die Skalierungskonfiguration hinzuzufügen. Die Kapazitätswerte in diesem Beispiel ermöglichen es jeder Serverless-v2-Instance im Cluster, zwischen 2 und 32 Aurora Capacity Units (ACUs) zu skalieren. Informationen zur Aurora-Serverless-v2-Funktion und zur Auswahl des Kapazitätsbereichs finden Sie unter Verwenden von Aurora Serverless v2.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier cluster-with-2-subnets \
  --db-cluster-identifier cluster-cloned-to-3-subnets \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name three-subnets \
  --serverless-v2-scaling-configuration 'MinCapacity=2,MaxCapacity=32'
```

Unabhängig von der Anzahl der von den DB-Instances verwendeten Subnetze ist der Aurora-Speicher für den Quell-Cluster und den Klon drei AZs zugeordnet. Das folgende Beispiel listet die AZs auf, die sowohl dem ursprünglichen Cluster als auch dem Klon zugeordnet sind, und zwar für beide `restore-db-cluster-to-point-in-time`-Befehle in den vorherigen Beispielen.

```
aws rds describe-db-clusters --db-cluster-identifier cluster-with-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f

aws rds describe-db-clusters --db-cluster-identifier cluster-with-2-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d

aws rds describe-db-clusters --db-cluster-identifier cluster-cloned-to-3-subnets \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1a
us-east-1c
us-east-1d
```

Da sich mindestens zwei der AZs zwischen jedem Paar aus Original- und Kloncluster überschneiden, können beide Cluster auf denselben zugrunde liegenden Aurora-Speicher zugreifen.

Schritt 2: Angeben der Netzwerkeinstellungen für Instances im Klon

Wenn Sie DB-Instances im Klon erstellen, erben diese standardmäßig die DB-Subnetzgruppe vom Cluster selbst. Auf diese Weise weist Aurora jede Instance automatisch einem bestimmten Subnetz zu und erstellt sie in der AZ, die dem Subnetz zugeordnet ist. Diese Wahl ist besonders für Entwicklungs- und Testsysteme praktisch, da Sie beim Hinzufügen neuer Instances zum Klon nicht die Subnetz-IDs oder AZs im Auge behalten müssen.

Als Alternative können Sie die AZ angeben, wenn Sie eine Aurora-DB-Instance für den Klon erstellen. Die AZ, die Sie angeben, muss aus der Gruppe von AZs stammen, die dem Klon zugeordnet sind. Wenn die DB-Subnetzgruppe, die Sie für den Klon verwenden, nur zwei Subnetze enthält, können Sie nur aus den AZs auswählen, die diesen beiden Subnetzen zugeordnet sind. Diese Wahl bietet Flexibilität und Resilienz für Systeme mit hoher Verfügbarkeit, da Sie sicherstellen können, dass sich die Writer-Instance und die Standby-Reader-Instance in unterschiedlichen AZs befinden. Oder wenn Sie dem Cluster weitere Reader hinzufügen, können Sie sicherstellen, dass diese auf drei AZs verteilt sind. Auf diese Weise haben Sie selbst im seltenen Fall eines AZ-Ausfalls immer noch eine Writer-Instance und eine weitere Reader-Instance in zwei anderen AZs.

Das folgende Beispiel fügt einem geklonten Aurora-PostgreSQL-Cluster, der eine benutzerdefinierte DB-Subnetzgruppe verwendet, eine bereitgestellte DB-Instance hinzu.

```
aws rds create-db-instance --db-cluster-identifier my_aurora_postgresql_clone \
  --db-instance-identifier my_postgres_instance \
  --db-subnet-group-name my_new_subnet \
  --engine aurora-postgresql \
  --db-instance-class db.t4g.medium
```

Das folgende Beispiel zeigt die teilweise Ausgabe eines solchen Befehls.

```
{
  'DBInstanceIdentifier': 'my_postgres_instance',
  'DBClusterIdentifier': 'my_aurora_postgresql_clone',
  'DBInstanceClass': 'db.t4g.medium',
  'DBInstanceStatus': 'creating'
  ...
}
```

Das folgende Beispiel fügt eine Aurora-Serverless-v2-DB-Instance zu einem Aurora-MySQL-Klon hinzu, der eine benutzerdefinierte DB-Subnetzgruppe verwendet. Um Serverless-v2-

Instances verwenden zu können, stellen Sie sicher, dass Sie die `--serverless-v2-scaling-configuration`-Option für den `restore-db-cluster-to-point-in-time`-Befehl angeben, wie in den vorherigen Beispielen gezeigt.

```
aws rds create-db-instance --db-cluster-identifier my_aurora_mysql_clone \
  --db-instance-identifier my_mysql_instance \
  --db-subnet-group-name my_other_new_subnet \
  --engine aurora-mysql \
  --db-instance-class db.serverless
```

Das folgende Beispiel zeigt die teilweise Ausgabe eines solchen Befehls.

```
{
  'DBInstanceIdentifier': 'my_mysql_instance',
  'DBClusterIdentifier': 'my_aurora_mysql_clone',
  'DBInstanceClass': 'db.serverless',
  'DBInstanceStatus': 'creating'
  ...
}
```

Schritt 3: Herstellen der Konnektivität zwischen einem Clientsystem und einem Klon

Wenn Sie bereits von einem Clientsystem aus eine Verbindung zu einem Aurora-Cluster herstellen, sollten Sie dieselbe Art von Konnektivität für einen neuen Klon zulassen. Beispielsweise können Sie über eine Amazon-Cloud9-Instance oder EC2-Instance eine Verbindung mit dem ursprünglichen Cluster herstellen. Um Verbindungen von denselben oder neuen Clientsystemen, die Sie in der Ziel-VPC erstellen, zuzulassen, richten Sie äquivalente DB-Subnetzgruppen und VPC-Sicherheitsgruppen wie in der VPC ein. Geben Sie dann die Subnetzgruppe und die Sicherheitsgruppen an, wenn Sie den Klon erstellen.

In den folgenden Beispielen wird ein Aurora-Serverless-v2-Klon eingerichtet. Diese Konfiguration basiert auf der Kombination von `--engine-mode provisioned` und `--serverless-v2-scaling-configuration` bei der Erstellung des DB-Clusters und dann `--db-instance-class db.serverless` bei der Erstellung der einzelnen DB-Instances im Cluster. Der Engine-Modus `provisioned` ist die Standardeinstellung, sodass Sie diese Option auch weglassen können, wenn Sie dies bevorzugen.

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier serverless-sql-postgres\
  --db-cluster-identifier serverless-sql-postgres-clone \
```

```
  --db-subnet-group-name 'default-vpc-1234' \
  --vpc-security-group-ids 'sg-4567' \
  --serverless-v2-scaling-configuration 'MinCapacity=0.5,MaxCapacity=16' \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

Geben Sie dann beim Erstellen der DB-Instances im Klon dieselbe `--db-subnet-group-name`-Option an. Optional können Sie die `--availability-zone`-Option einschließen und eine der AZs angeben, die den Subnetzen in dieser Subnetzgruppe zugeordnet sind. Diese AZ muss auch eine der AZs sein, die dem ursprünglichen Cluster zugeordnet sind.

```
aws rds create-db-instance \
  --db-cluster-identifier serverless-sql-postgres-clone \
  --db-instance-identifier serverless-sql-postgres-clone-instance \
  --db-instance-class db.serverless \
  --db-subnet-group-name 'default-vpc-987zyx654' \
  --availability-zone 'us-east-1c' \
  --engine aurora-postgresql
```

Verschieben eines Clusters aus öffentlichen Subnetzen in private

Mithilfe des Klonens können Sie einen Cluster zwischen öffentlichen und privaten Subnetzen migrieren. Sie können dies tun, wenn Sie Ihrer Anwendung zusätzliche Sicherheitsebenen hinzufügen, bevor Sie sie in der Produktion einsetzen. In diesem Beispiel sollten Sie die privaten Subnetze und das NAT-Gateway bereits eingerichtet haben, bevor Sie den Klonvorgang mit Aurora starten.

Für die Schritte, an denen Aurora beteiligt ist, können Sie dieselben allgemeinen Schritte wie in den vorherigen Beispielen für Sammeln von Informationen über die Netzwerkumgebung und Erstellen eines Aurora-Klons mit neuen Netzwerkeinstellungen befolgen. Der Hauptunterschied besteht darin, dass Sie, selbst wenn Sie über öffentliche Subnetze verfügen, die allen AZs des ursprünglichen Clusters zugeordnet sind, jetzt überprüfen müssen, ob Sie über genügend private Subnetze für einen Aurora-Cluster verfügen. Außerdem müssen diese Subnetze allen AZs zugeordnet sein, die für Aurora-Speicher im ursprünglichen Cluster verwendet werden. Ähnlich wie bei anderen Anwendungsfällen des Klonens können Sie die DB-Subnetzgruppe für den Klon entweder mit drei oder zwei privaten Subnetzen erstellen, die den erforderlichen AZs zugeordnet sind. Wenn Sie jedoch zwei private Subnetze in der DB-Subnetzgruppe verwenden, benötigen Sie ein drittes privates Subnetz, das der dritten AZ zugeordnet ist, die für Aurora-Speicher im ursprünglichen Cluster verwendet wird.

Anhand dieser Checkliste können Sie überprüfen, ob alle Voraussetzungen für diese Art von Klonvorgang erfüllt sind.

- Notieren Sie die drei AZs, die dem ursprünglichen Cluster zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 1: Überprüfen der Availability Zones des ursprünglichen Clusters.
- Notieren Sie die drei oder zwei AZs, die den öffentlichen Subnetzen in der DB-Subnetzgruppe für den ursprünglichen Cluster zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 3: Überprüfen der Subnetze des ursprünglichen Clusters.
- Erstellen Sie private Subnetze, die allen drei AZs zugeordnet sind, die dem ursprünglichen Cluster zugeordnet sind. Führen Sie außerdem alle anderen Schritte zur Netzwerkeinrichtung durch, z. B. die Einrichtung eines NAT-Gateways, um mit den privaten Subnetzen kommunizieren zu können. Eine ausführliche Anleitung finden Sie unter Erstellen eines Subnetzes im Benutzerhandbuch für Amazon Virtual Private Cloud.
- Erstellen Sie eine neue DB-Subnetzgruppe, die drei oder zwei der privaten Subnetze enthält, die den AZs aus dem ersten Punkt zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 2: Erstellen der DB-Subnetzgruppe für den Klon.

Wenn alle Voraussetzungen erfüllt sind, können Sie die Datenbankaktivität auf dem ursprünglichen Cluster pausieren, während Sie den Klon erstellen, und Ihre Anwendung auf die Verwendung des Klons umstellen. Nachdem der Klon erstellt wurde und Sie sich vergewissert haben, dass eine Verbindung dazu hergestellt werden kann, Sie Ihren Anwendungscode ausführen können usw., können Sie die Verwendung des ursprünglichen Clusters einstellen.

Umfassendes Beispiel für das Erstellen eines VPC-Klons

Beim Erstellen eines Klons in einer anderen VPC als dem Original werden dieselben allgemeinen Schritte wie in den vorherigen Beispielen ausgeführt. Da die VPC-ID eine Eigenschaft der Subnetze ist, geben Sie die VPC-ID nicht als Parameter an, wenn Sie einen der RDS-CLI-Befehle ausführen. Der Hauptunterschied besteht darin, dass Sie mit größerer Wahrscheinlichkeit neue Subnetze, neue Subnetze, die bestimmten AZs zugeordnet sind, eine VPC-Sicherheitsgruppe und eine neue DB-Subnetzgruppe erstellen müssen. Dies gilt insbesondere, wenn dies der erste Aurora-Cluster ist, den Sie in dieser VPC erstellen.

Anhand dieser Checkliste können Sie überprüfen, ob alle Voraussetzungen für diese Art von Klonvorgang erfüllt sind.

- Notieren Sie die drei AZs, die dem ursprünglichen Cluster zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 1: Überprüfen der Availability Zones des ursprünglichen Clusters.
- Notieren Sie die drei oder zwei AZs, die den Subnetzen in der DB-Subnetzgruppe für den ursprünglichen Cluster zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 2: Überprüfen der DB-Subnetzgruppe des ursprünglichen Clusters.
- Erstellen Sie Subnetze, die allen drei AZs zugeordnet sind, die dem ursprünglichen Cluster zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 1: Erstellen der Subnetze für den Klon.
- Führen Sie weitere Schritte für die Netzwerkeinrichtung, z. B. die Einrichtung einer VPC-Sicherheitsgruppe, für Clientsysteme, Anwendungsserver usw. aus, um mit den DB-Instances im Klon kommunizieren zu können. Detaillierte Anweisungen finden Sie unter Zugriffskontrolle mit Sicherheitsgruppen.
- Erstellen Sie eine neue DB-Subnetzgruppe, die drei oder zwei der Subnetze enthält, die den AZs aus dem ersten Punkt zugeordnet sind. Detaillierte Anweisungen finden Sie unter Schritt 2: Erstellen der DB-Subnetzgruppe für den Klon.

Wenn alle Voraussetzungen erfüllt sind, können Sie die Datenbankaktivität auf dem ursprünglichen Cluster pausieren, während Sie den Klon erstellen, und Ihre Anwendung auf die Verwendung des Klons umstellen. Nachdem der Klon erstellt wurde und Sie sich vergewissert haben, dass eine Verbindung dazu hergestellt werden kann, Sie Ihren Anwendungscode ausführen können usw., können Sie entscheiden, ob sowohl der ursprüngliche Cluster als auch die Klone ausgeführt werden sollen, oder die Ausführung des ursprünglichen Clusters einstellen.

Die folgenden Linux-Beispiele zeigen die Abfolge der AWS-CLI-Operationen zum Klonen eines Aurora-DB-Clusters von einer VPC in eine andere. Einige Felder, die für die Beispiele nicht relevant sind, werden in der Befehlsausgabe nicht angezeigt.

Zunächst überprüfen wir die IDs der Quell- und Ziel-VPCs. Der beschreibende Name, den Sie einer VPC bei ihrer Erstellung zuweisen, wird in den VPC-Metadaten als Tag dargestellt.

```
$ aws ec2 describe-vpcs --query '*[].[VpcId,Tags]'
[
    [
        'vpc-0f0c0fc0000b0ffb0',
        [
            {
                'Key': 'Name',
```

```
                'Value': 'clone-vpc-source'
            }
        ]
    ],
    [
        'vpc-9e99d9f99a999bd99',
        [
            {
                'Key': 'Name',
                'Value': 'clone-vpc-dest'
            }
        ]
    ]
]
```

Der ursprüngliche Cluster ist bereits in der Quell-VPC vorhanden. Um den Klon mit demselben Satz von AZs für den Aurora-Speicher einzurichten, überprüfen wir die AZs, die vom ursprünglichen Cluster verwendet werden.

```
$ aws rds describe-db-clusters --db-cluster-identifier original-cluster \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

Wir stellen sicher, dass Subnetze vorhanden sind, die den vom ursprünglichen Cluster verwendeten AZs entsprechen: `us-east-1c`, `us-east-1d` und `us-east-1f`.

```
$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
  --availability-zone us-east-1c --cidr-block 10.0.0.128/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1c',
        'SubnetId': 'subnet-3333a33be3ef3e333',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1d --cidr-block 10.0.0.160/28
{
```

```
    'Subnet': {
        'AvailabilityZone': 'us-east-1d',
        'SubnetId': 'subnet-4eeb444cd44b4d444',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}

$ aws ec2 create-subnet --vpc-id vpc-9e99d9f99a999bd99 \
--availability-zone us-east-1f --cidr-block 10.0.0.224/28
{
    'Subnet': {
        'AvailabilityZone': 'us-east-1f',
        'SubnetId': 'subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
    }
}
```

Dieses Beispiel bestätigt, dass es Subnetze gibt, die den erforderlichen AZs in der Ziel-VPC zugeordnet sind.

```
aws ec2 describe-subnets --query 'sort_by(*[] | [?VpcId == `vpc-9e99d9f99a999bd99`] |
[].{SubnetId:SubnetId,VpcId:VpcId,AvailabilityZone:AvailabilityZone},
 &AvailabilityZone)' --output table

-------------------------------------------------------------------------
|                            DescribeSubnets                            |
+------------------+----------------------------+-------------------------+
| AvailabilityZone |         SubnetId           |          VpcId          |
+------------------+----------------------------+-------------------------+
|  us-east-1a      |  subnet-000ff0e00000c0aea  |  vpc-9e99d9f99a999bd99  |
|  us-east-1b      |  subnet-1111d111111ca11b1  |  vpc-9e99d9f99a999bd99  |
|  us-east-1c      | subnet-3333a33be3ef3e333   |  vpc-9e99d9f99a999bd99  |
|  us-east-1d      | subnet-4eeb444cd44b4d444   |  vpc-9e99d9f99a999bd99  |
|  us-east-1f      | subnet-66eea6666fb66d66c   |  vpc-9e99d9f99a999bd99  |
+------------------+----------------------------+-------------------------+
```

Bevor Sie einen Aurora-DB-Cluster in der VPC erstellen, benötigen Sie eine DB-Subnetzgruppe mit Subnetzen, die den für Aurora-Speicher verwendeten AZs zugeordnet sind. Wenn Sie einen regulären Cluster erstellen, können Sie einen beliebigen Satz von drei AZs verwenden. Wenn Sie einen vorhandenen Cluster klonen, muss die Subnetzgruppe mit mindestens zwei der drei AZs übereinstimmen, die für Aurora-Speicher verwendet werden.

```
$ aws rds create-db-subnet-group \
  --db-subnet-group-name subnet-group-in-other-vpc \
  --subnet-ids
 '["subnet-3333a33be3ef3e333","subnet-4eeb444cd44b4d444","subnet-66eea6666fb66d66c"]' \
  --db-subnet-group-description 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c'

{
    'DBSubnetGroup': {
        'DBSubnetGroupName': 'subnet-group-in-other-vpc',
        'DBSubnetGroupDescription': 'DB subnet group with 3 subnets:
 subnet-3333a33be3ef3e333,subnet-4eeb444cd44b4d444,subnet-66eea6666fb66d66c',
        'VpcId': 'vpc-9e99d9f99a999bd99',
        'SubnetGroupStatus': 'Complete',
        'Subnets': [
            {
                'SubnetIdentifier': 'subnet-4eeb444cd44b4d444',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1d' }
            },
            {
                'SubnetIdentifier': 'subnet-3333a33be3ef3e333',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1c' }
            },
            {
                'SubnetIdentifier': 'subnet-66eea6666fb66d66c',
                'SubnetAvailabilityZone': { 'Name': 'us-east-1f' }
            }
        ]
    }
}
```

Jetzt sind die Subnetze und die DB-Subnetzgruppe eingerichtet. Das folgende Beispiel zeigt `restore-db-cluster-to-point-in-time`, womit der Cluster geklont wird. Die `--db-subnet-group-name`-Option ordnet den Klon dem richtigen Satz von Subnetzen zu, die wiederum dem richtigen Satz von AZs aus dem ursprünglichen Cluster zugeordnet sind.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier original-cluster \
  --db-cluster-identifier clone-in-other-vpc \
  --restore-type copy-on-write --use-latest-restorable-time \
  --db-subnet-group-name subnet-group-in-other-vpc
```

```
{
  'DBClusterIdentifier': 'clone-in-other-vpc',
  'DBSubnetGroup': 'subnet-group-in-other-vpc',
  'Engine': 'aurora-postgresql',
  'EngineVersion': '15.4',
  'Status': 'creating',
  'Endpoint': 'clone-in-other-vpc.cluster-c0abcdef.us-east-1.rds.amazonaws.com'
}
```

Im folgenden Beispiel wird bestätigt, dass der Aurora-Speicher im Klon denselben Satz von AZs wie im ursprünglichen Cluster verwendet.

```
$ aws rds describe-db-clusters --db-cluster-identifier clone-in-other-vpc \
  --query 'sort_by(*[].AvailabilityZones[].{Zone:@},&Zone)' --output text

us-east-1c
us-east-1d
us-east-1f
```

Nun können Sie DB-Instances für den Klon erstellen. Stellen Sie sicher, dass die jeder Instance zugeordnete VPC-Sicherheitsgruppe Verbindungen aus den IP-Adressbereichen zulässt, die Sie für die EC2-Instances, Anwendungsserver usw. in der Ziel-VPC verwenden.

Kontoübergreifendes Klonen mit AWS RAM und Amazon Aurora

Wenn Sie AWS Resource Access Manager (AWS RAM) mit Amazon Aurora verwenden, können Sie Aurora-DB-Cluster und -Clones, die zu Ihrem AWS Konto gehören, mit einem anderen AWS Konto oder einer anderen Organisation teilen. Ein solches kontoübergreifendes Klonen ist viel schneller als das Erstellen und Wiederherstellen eines Datenbank-Snapshots. Sie können einen Klon eines Ihrer Aurora-DB-Cluster erstellen und den Klon freigeben. Oder Sie können Ihren Aurora-DB-Cluster mit einem anderen AWS Konto teilen und den Kontoinhaber den Clone erstellen lassen. Welchen Ansatz Sie wählen, hängt von Ihrem Anwendungsfall ab.

Beispielsweise müssen Sie möglicherweise regelmäßig einen Klon Ihrer Finanzdatenbank für das interne Revisionsteam Ihres Unternehmens freigeben. In diesem Fall hat Ihr Prüfungsteam ein eigenes AWS -Konto für die von ihm verwendeten Anwendungen. Sie können dem AWS Konto des Prüfungsteams die Erlaubnis erteilen, auf Ihren Aurora-DB-Cluster zuzugreifen und ihn nach Bedarf zu klonen.

Auf der anderen Seite, wenn ein externer Anbieter Ihre Finanzdaten prüft, ziehen Sie es möglicherweise vor, den Klon selbst zu erstellen. Sie geben dann dem externen Anbieter nur Zugriff auf den Klon.

Sie können auch kontenübergreifendes Klonen verwenden, um viele der gleichen Anwendungsfälle für das Klonen innerhalb desselben AWS Kontos zu unterstützen, z. B. Entwicklung und Testen. Beispielsweise könnte Ihre Organisation unterschiedliche AWS Konten für Produktion, Entwicklung, Tests usw. verwenden. Weitere Informationen finden Sie unter Übersicht über das Aurora-Klonen.

Daher möchten Sie vielleicht einen Clone mit einem anderen Konto teilen oder einem anderen AWS Konto erlauben, Klone Ihrer Aurora-DB-Cluster zu erstellen. AWS Verwenden Sie in beiden Fällen zunächst, AWS RAM um ein Share-Objekt zu erstellen. Vollständige Informationen zur gemeinsamen Nutzung von AWS Ressourcen zwischen AWS Konten finden Sie im AWS RAM Benutzerhandbuch.

Zum Erstellen eines kontoübergreifenden Clones sind Aktionen von dem AWS Konto, dem der ursprüngliche Cluster gehört, und von dem AWS Konto, das den Clone erstellt, erforderlich. Zuerst ändert der ursprüngliche Clusterbesitzer den Cluster, um es einem oder mehreren anderen Konten zu ermöglichen, ihn zu klonen. Wenn sich eines der Konten in einer anderen AWS Organisation befindet, wird eine Einladung zum Teilen AWS generiert. Das andere Konto muss die Einladung annehmen, bevor Sie fortfahren können. Dann kann jedes autorisierte Konto den Cluster klonen. Während dieses Prozesses wird der Cluster durch seinen eindeutigen Amazon-Ressourcennamen (ARN) identifiziert.

Wie beim Klonen innerhalb desselben AWS Kontos wird zusätzlicher Speicherplatz nur verwendet, wenn die Quelle oder der Klon Änderungen an den Daten vornehmen. Zu diesem Zeitpunkt werden dann Gebühren für die Speicherung erhoben. Wenn der Quellen-Cluster gelöscht wird, werden Speicherkosten gleichmäßig auf die verbleibenden geklonten Cluster verteilt.

Themen

- Einschränkungen für kontoübergreifendes Klonen
- Erlauben Sie anderen AWS Konten, Ihren Cluster zu klonen
- Einen Cluster klonen, der einem anderen AWS Konto gehört

Einschränkungen für kontoübergreifendes Klonen

Kontoübergreifendes Aurora-Klonen hat folgende Einschränkungen:

- Sie können einen Aurora Serverless v1 Cluster nicht für mehrere AWS Konten klonen.

- Sie können keine Einladungen zu gemeinsam genutzten Ressourcen mit dem anzeigen oder annehmen AWS-Managementkonsole. Verwenden Sie die AWS CLI, die Amazon RDS-API oder die AWS RAM Konsole, um Einladungen zu gemeinsam genutzten Ressourcen anzuzeigen und anzunehmen.
- Sie können nur einen neuen Klon aus einer Ressource erstellen, die mit Ihrem geteilt wird AWS-Konto. Dies gilt unabhängig davon, ob es sich bei der freigegebenen Ressource um einen ursprünglichen Aurora-DB-Cluster oder um einen zuvor erstellten Klon handelt.
- Sie können nur einen neuen Klon aus einem Klon erstellen, der mit Ihrem AWS Konto geteilt wurde.
- Sie können keine Ressourcen (Klone oder Aurora-DB-Cluster) teilen, die mit Ihrem AWS Konto geteilt wurden.
- Sie können maximal 15 kontoübergreifende Klone aus einem einzelnen Aurora-DB-Cluster erstellen.
- Jeder der 15 kontoübergreifenden Klone muss einem anderen Konto gehören. AWS Das heißt, Sie können innerhalb eines Kontos nur einen kontenübergreifenden Klon eines Clusters erstellen. AWS
- Nachdem Sie einen Cluster geklont haben, werden der ursprüngliche Cluster und sein Klon als identisch angesehen, um Grenzwerte für kontoübergreifende Klone durchzusetzen. Sie können innerhalb desselben Kontos keine kontenübergreifenden Klone sowohl des ursprünglichen Clusters als auch des geklonten Clusters erstellen. AWS Die Gesamtzahl der kontoübergreifenden Klone für den ursprünglichen Cluster und einen seiner Klone darf 15 nicht überschreiten.
- Sie können einen Aurora-DB-Cluster nicht mit anderen AWS Konten teilen, es sei denn, der Cluster befindet sich in einem `ACTIVE` Status.
- Sie können einen Aurora-DB-Cluster, der mit anderen AWS Konten geteilt wurde, nicht umbenennen.
- Sie können keinen kontoübergreifenden Klon eines Clusters erstellen, der mit dem RDS-Standardschlüssel verschlüsselt ist.
- Sie können in einem AWS Konto keine unverschlüsselten Klone aus verschlüsselten Aurora-DB-Clustern erstellen, die von einem anderen AWS Konto gemeinsam genutzt wurden. Der Clustereigentümer muss die Erlaubnis zum Zugriff auf den Quellcluster AWS KMS key. Sie können jedoch auch einen anderen Schlüssel verwenden, wenn Sie den Klon erstellen.

Erlauben Sie anderen AWS Konten, Ihren Cluster zu klonen

Um anderen AWS Konten das Klonen eines Clusters zu ermöglichen, dessen Eigentümer Sie sind, legen AWS RAM Sie die Freigabeberechtigung fest. Dadurch wird auch eine Einladung an jedes der anderen Konten gesendet, die sich in einer anderen AWS Organisation befinden.

Informationen zur gemeinsamen Nutzung von Ressourcen, die Ihnen gehören, in der AWS RAM Konsole finden Sie im AWS RAM Benutzerhandbuch unter Gemeinsame Nutzung von Ressourcen, die Ihnen gehören.

Themen

- Erteilen Sie anderen AWS Konten die Erlaubnis, Ihren Cluster zu klonen
- Prüfen Sie, ob ein Cluster, den Sie besitzen, mit anderen AWS Konten geteilt wird

Erteilen Sie anderen AWS Konten die Erlaubnis, Ihren Cluster zu klonen

Wenn der Cluster, den Sie gemeinsam nutzen, verschlüsselt ist, teilen Sie auch die AWS KMS key für den Cluster. Sie können AWS Identity and Access Management (IAM-) Benutzern oder Rollen in einem AWS Konto erlauben, einen KMS-Schlüssel in einem anderen Konto zu verwenden.

Dazu fügen Sie zunächst das externe Konto (Root-Benutzer) zur Schlüsselrichtlinie des KMS-Schlüssels hinzu. AWS KMS Sie fügen der Schlüsselrichtlinie nicht die einzelnen Benutzer oder Rollen hinzu, sondern nur das externe Konto, das sie enthält. Sie können nur einen von Ihnen erstellten KMS-Schlüssel freigeben, nicht den Standardschlüssel des RDS-Dienstes. Informationen zur Zugriffskontrolle für KMS-Schlüssel finden Sie unter Authentifizierung und Zugriffskontrolle für AWS KMS.

Konsole

So gewähren Sie die Berechtigung zum Klonen Ihres Clusters

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den DB-Cluster aus, den Sie freigeben möchten, um seine Details-Seite anzuzeigen, und wählen Sie dann die Registerkarte Connectivity & security (Konnektivität und Sicherheit) aus.

4. Geben Sie im Abschnitt DB-Cluster mit anderen AWS Konten teilen die numerische Konto-ID für das AWS Konto ein, dem Sie das Klonen dieses Clusters erlauben möchten. Für ein Konto IDs in derselben Organisation können Sie mit der Eingabe in das Feld beginnen und dann eine Auswahl aus dem Menü treffen.

   > **Important**
   >
   > In manchen Fällen möchten Sie vielleicht, dass ein Konto, das sich nicht in derselben AWS -Organisation befindet wie Ihr Konto, einen Cluster klont. In diesen Fällen meldet die Konsole aus Sicherheitsgründen nicht, wer diese Konto-ID besitzt oder ob das Konto existiert.
   > Seien Sie vorsichtig bei der Eingabe von Kontonummern, die sich nicht in derselben AWS Organisation wie Ihr AWS Konto befinden. Überprüfen Sie sofort, ob Sie die Freigabe für das beabsichtigte Konto erteilt haben.

5. Überprüfen Sie auf der Bestätigungsseite, dass die von Ihnen angegebene Konto-ID richtig ist. Geben Sie im Bestätigungsfeld `share` ein, um dies zu bestätigen.

   Auf der Detailseite wird unter Konten, mit denen dieser DB-Cluster gemeinsam genutzt wird, ein Eintrag mit der angegebenen AWS Konto-ID angezeigt. Die Spalte Status zeigt ursprünglich den Status Pending (Ausstehend) an.

6. Wenden Sie sich an den Besitzer des anderen AWS Kontos, oder melden Sie sich bei diesem Konto an, wenn Sie beide besitzen. Weisen Sie den Besitzer des anderen Kontos an, die Freigabeeinladung anzunehmen und den DB-Cluster zu klonen, wie nachfolgend beschrieben.

AWS CLI

So gewähren Sie die Berechtigung zum Klonen Ihres Clusters

1. Tragen Sie die Daten für die erforderlichen Parameter zusammen. Sie benötigen den ARN für Ihren Cluster und die numerische ID für das andere AWS Konto.
2. Führen Sie den AWS RAM CLI-Befehl aus `create-resource-share`.

   Für Linux, macOS oder Unix:

```
aws ram create-resource-share --name descriptive_name \
  --region region \
  --resource-arns cluster_arn \
```

```
  --principals other_account_ids
```

Für Windows:

```
aws ram create-resource-share --name descriptive_name ^
  --region region ^
  --resource-arns cluster_arn ^
  --principals other_account_ids
```

Um mehrere Konten IDs für den `--principals` Parameter anzugeben, IDs trennen Sie diese durch Leerzeichen voneinander. Um anzugeben, ob sich das zulässige Konto außerhalb Ihrer AWS Organisation befinden IDs kann, geben Sie den `--no-allow-external-principals` Parameter `--allow-external-principals` oder für an`create-resource-share`.

AWS RAM API

So gewähren Sie die Berechtigung zum Klonen Ihres Clusters

1. Tragen Sie die Daten für die erforderlichen Parameter zusammen. Sie benötigen den ARN für Ihren Cluster und die numerische ID für das andere AWS Konto.
2. Rufen Sie den AWS RAM API-Vorgang CreateResourceShareauf und geben Sie die folgenden Werte an:
   - Geben Sie die Konto-ID für ein oder mehrere AWS Konten als `principals` Parameter an.
   - Geben Sie die ARN mindestens eines Aurora-DB-Clusters als `resourceArns`-Parameter an.
   - Geben Sie an, ob sich das zulässige Konto außerhalb Ihrer AWS Organisation befinden IDs kann, indem Sie einen booleschen Wert für den `allowExternalPrincipals` Parameter angeben.

Neuerstellen eines Clusters, der den RDS-Standardschlüssel verwendet

Wenn der verschlüsselte Cluster, den Sie freigeben möchten, den RDS-Standardschlüssel verwendet, stellen Sie sicher, dass der Cluster neu erstellt wird. Erstellen Sie dazu einen manuellen Snapshot Ihres DB-Clusters, verwenden Sie eine AWS KMS key, und stellen Sie den Cluster dann in einem neuen Cluster wieder her. Geben Sie dann den neuen Cluster frei. Führen Sie die folgenden Schritte aus, um diesen Vorgang durchzuführen.

So erstellen Sie einen verschlüsselten Cluster neu, der den RDS-Standardschlüssel verwendet

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots aus.
3. Wählen Sie Ihren Snapshot aus.
4. Wählen Sie unter Actions (Aktionen) die Option Copy Snapshot (Snapshot kopieren) und anschließend Enable encryption (Verschlüsselung aktivieren) aus.
5. Wählen Sie für AWS KMS key den neuen Verschlüsselungscode, den Sie verwenden möchten.
6. Stellen Sie den kopierten Snapshot wieder her. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot. Die neue DB-Instance verwendet Ihren neuen Verschlüsselungsschlüssel.
7. (Optional) Löschen Sie den alten DB-Cluster, wenn Sie ihn nicht mehr benötigen. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Löschen eines DB-Cluster-Snapshots. Bevor Sie dies tun, überprüfen Sie, ob Ihr neuer Cluster über alle notwendigen Daten verfügt und ob Ihre Anwendung erfolgreich darauf zugreifen kann.

Prüfen Sie, ob ein Cluster, den Sie besitzen, mit anderen AWS Konten geteilt wird

Sie können überprüfen, ob andere Benutzer die Berechtigung für die Freigabe eines Clusters haben. Dies kann hilfreich sein, um zu wissen, ob der Cluster sich dem Limit für die maximale Anzahl an kontoübergreifenden Klonen nähert.

Die Verfahren zur gemeinsamen Nutzung von Ressourcen mithilfe der AWS RAM Konsole finden Sie im AWS RAM Benutzerhandbuch unter Gemeinsame Nutzung von Ressourcen, die Ihnen gehören.

AWS CLI

Finden Sie heraus, ob ein Cluster, den Sie besitzen, mit anderen AWS Konten gemeinsam genutzt wird

- Rufen Sie den AWS RAM CLI-Befehl `list-principals`auf und verwenden Sie dabei Ihre Konto-ID als Ressourcenbesitzer und den ARN Ihres Clusters als Ressourcen-ARN. Sie können mit dem folgenden Befehl alle Freigaben anzeigen. Die Ergebnisse geben an, welche AWS Konten den Cluster klonen dürfen.

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
```

```
    --principals your_aws_id
```

AWS RAM API

Um herauszufinden, ob ein Cluster, den Sie besitzen, mit anderen AWS Konten geteilt wird

- Rufen Sie den AWS RAM API-Vorgang auf ListPrincipals. Verwenden Sie Ihre Konto-ID als Ressourcenbesitzer und den ARN Ihres Clusters als Ressourcen-ARN.

Einen Cluster klonen, der einem anderen AWS Konto gehört

Um einen Cluster zu klonen, der einem anderen AWS Konto gehört, holen Sie sich AWS RAM die Erlaubnis ein, den Klon zu erstellen. Wenn Sie die erforderliche Berechtigung haben, verwenden Sie das Standardverfahren zum Klonen eines Aurora-Clusters.

Sie können auch überprüfen, ob es sich bei einem Cluster, den Sie besitzen, um einen Klon eines Clusters handelt, der einem anderen AWS Konto gehört.

Informationen zu den Verfahren für die Arbeit mit Ressourcen, die anderen Benutzern gehören, AWS RAM finden Sie im AWS RAM Benutzerhandbuch unter Zugreifen auf Ressourcen, die mit Ihnen geteilt wurden.

Themen

- Einladungen zum Klonen von Clustern anzeigen, die anderen AWS Konten gehören
- Annahme von Einladungen zur gemeinsamen Nutzung von Clustern, die anderen AWS Konten gehören
- Einen Aurora-Cluster klonen, der einem anderen AWS Konto gehört
- Überprüfen, ob ein DB-Cluster ein kontoübergreifender Klon ist

Einladungen zum Klonen von Clustern anzeigen, die anderen AWS Konten gehören

Um mit Einladungen zum Klonen von Clustern zu arbeiten, die AWS Konten in anderen AWS Organisationen gehören AWS CLI, verwenden Sie die, die AWS RAM Konsole oder die AWS RAM API. Aktuell können Sie dieses Verfahren nicht mithilfe der Amazon-RDS-Konsole durchführen.

Die Verfahren zur Verwendung von Einladungen in der AWS RAM Konsole finden Sie im AWS RAM Benutzerhandbuch unter Zugreifen auf Ressourcen, die mit Ihnen geteilt wurden.

AWS CLI

Um Einladungen zum Klonen von Clustern anzuzeigen, die anderen AWS Konten gehören

1. Führen Sie den AWS RAM CLI-Befehl aus `get-resource-share-invitations`.

   ```
   aws ram get-resource-share-invitations --region region_name
   ```

   Die Ergebnisse des vorausgehenden Befehls zeigen alle Einladungen, Cluster zu klonen, an, einschließlich aller Einladungen, die Sie bereits angenommen oder abgelehnt haben.
2. (Optional) Filtern Sie die Liste, um nur die Einladungen anzuzeigen, die eine Aktion Ihrerseits erforderlich machen. Fügen Sie dafür den -Parameter hinz `--query 'resourceShareInvitations[?status==`PENDING`]'`.

AWS RAM API

Um Einladungen zum Klonen von Clustern anzuzeigen, die anderen AWS Konten gehören

1. Rufen Sie den AWS RAM API-Vorgang auf `GetResourceShareInvitations`. Diese Operation gibt all diese Einladungen zurück, einschließlich derer, die Sie bereits angenommen oder abgelehnt haben.
2. (Optional) Suchen Sie nur die Einladungen, die eine Aktion Ihrerseits erforderlich machen, indem Sie das Rückgabefeld `resourceShareAssociations` für den `status`-Wert `PENDING` aktivieren.

Annahme von Einladungen zur gemeinsamen Nutzung von Clustern, die anderen AWS Konten gehören

Sie können Einladungen zur gemeinsamen Nutzung von Clustern annehmen, die anderen AWS Konten gehören, die sich in anderen AWS Organisationen befinden. Um mit diesen Einladungen zu arbeiten, verwenden Sie den AWS CLI, den AWS RAM und RDS APIs oder die AWS RAM Konsole. Aktuell können Sie dieses Verfahren nicht mithilfe der RDS-Konsole durchführen.

Die Verfahren zur Verwendung von Einladungen in der AWS RAM Konsole finden Sie im AWS RAM Benutzerhandbuch unter Zugreifen auf mit Ihnen geteilte Ressourcen.

AWS CLI

So nehmen Sie eine Einladung zur gemeinsamen Nutzung eines Clusters von einem anderen AWS Konto aus an

1. Suchen Sie den Einladungs-ARN, indem Sie den AWS RAM CLI-Befehl ausführen `get-resource-share-invitations`, wie oben gezeigt.
2. Nehmen Sie die Einladung an, indem Sie den AWS RAM CLI-Befehl aufrufen `accept-resource-share-invitation`, wie im Folgenden gezeigt.

   Für Linux, macOS oder Unix:

   ```
   aws ram accept-resource-share-invitation \
     --resource-share-invitation-arn invitation_arn \
     --region region
   ```

   Für Windows:

   ```
   aws ram accept-resource-share-invitation ^
     --resource-share-invitation-arn invitation_arn ^
     --region region
   ```

AWS RAM und RDS-API

So nehmen Sie Einladung zur Freigabe eines Clusters, der jemand anderem gehört, an

1. Suchen Sie den Einladungs-ARN, indem Sie den AWS RAM API-Vorgang aufrufen `GetResourceShareInvitations`, wie oben gezeigt.
2. Übergeben Sie diesen ARN als `resourceShareInvitationArn` Parameter an den RDS-API-Vorgang AcceptResourceShareInvitation.

Einen Aurora-Cluster klonen, der einem anderen AWS Konto gehört

Nachdem Sie die Einladung von dem AWS Konto angenommen haben, dem der DB-Cluster gehört, wie oben gezeigt, können Sie den Cluster klonen.

Konsole

Um einen Aurora-Cluster zu klonen, der einem anderen AWS Konto gehört

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.

   Ganz oben in der Datenbankliste sollten Sie mindestens ein Element mit dem Role (Rolle)-Wert `Shared from account #account_id` sehen. Aus Sicherheitsgründen können Sie nur eingeschränkte Informationen über die ursprünglichen Cluster sehen. Die Eigenschaften, die Sie sehen können, sind solche wie etwa die Datenbank-Engine und -Version, die in Ihrem geklonten Cluster identisch sein müssen.
3. Wählen Sie den Cluster aus, den Sie vorhaben zu klonen.
4. Wählen Sie für Actions (Aktionen) Create clone (Klon erstellen) aus.
5. Gehen Sie entsprechend dem Verfahren unter Konsole vor, um die Einrichtung des geklonten Clusters durchzuführen.
6. Aktivieren Sie je nach Bedarf die Verschlüsselung für den geklonten Cluster. Wenn der Cluster, den Sie klonen, verschlüsselt ist, müssen Sie die Verschlüsselung für den geklonten Cluster aktivieren. Das AWS -Konto, das den Cluster für Sie freigegeben hat, muss auch den KMS-Schlüssel freigeben, der zur Verschlüsselung des Clusters verwendet wurde. Sie können denselben KMS-Schlüssel zur Verschlüsselung des Klons verwenden oder Ihren eigenen KMS-Schlüssel. Sie können keinen kontoübergreifenden Klon für einen Cluster erstellen, der mit dem Standard-KMS-Schlüssel verschlüsselt ist.

   Das Konto, dem der Verschlüsselungsschlüssel gehört, muss dem Zielkonto die Berechtigung zur Verwendung des Schlüssels mithilfe einer Schlüsselrichtlinie erteilen. Dieser Vorgang ist ähnlich dem Verfahren, mit dem verschlüsselte Snapshots freigegeben werden, mithilfe einer Schlüsselrichtlinie, die dem Zielkonto die Berechtigung zur Verwendung des Schlüssels gewährt.

AWS CLI

Um einen Aurora-Cluster zu klonen, der einem anderen AWS Konto gehört

1. Nehmen Sie die Einladung von dem AWS Konto an, dem der DB-Cluster gehört, wie oben gezeigt.

2. Klonen Sie den Cluster, indem Sie den vollständigen ARN des Quell-Clusters im `source-db-cluster-identifier`-Parameter des RDS-CLI-Befehls `restore-db-cluster-to-point-in-time` angeben, wie nachfolgend gezeigt.

   Wenn der als `source-db-cluster-identifier` übergebene ARN nicht freigegeben wurde, wird derselbe Fehler zurückgegeben, wie wenn der angegebene Cluster nicht vorhanden ist.

   Für Linux, macOS oder Unix:

   ```
   aws rds restore-db-cluster-to-point-in-time \
     --source-db-cluster-identifier=arn:aws:rds:arn_details \
     --db-cluster-identifier=new_cluster_id \
     --restore-type=copy-on-write \
     --use-latest-restorable-time
   ```

   Für Windows:

   ```
   aws rds restore-db-cluster-to-point-in-time ^
     --source-db-cluster-identifier=arn:aws:rds:arn_details ^
     --db-cluster-identifier=new_cluster_id ^
     --restore-type=copy-on-write ^
     --use-latest-restorable-time
   ```

3. Wenn der Cluster, den Sie klonen, verschlüsselt ist, verschlüsseln Sie Ihren geklonten Cluster, indem Sie einen `kms-key-id`-Parameter einfügen. Dieser `kms-key-id`-Wert kann derselbe sein, der zur Verschlüsselung des ursprünglichen DB-Clusters verwendet wurde, oder Ihr eigener KMS-Schlüssel. Ihr Konto muss über die Berechtigung zum Verwenden dieses Verschlüsselungsschlüssels verfügen.

   Für Linux, macOS oder Unix:

   ```
   aws rds restore-db-cluster-to-point-in-time \
     --source-db-cluster-identifier=arn:aws:rds:arn_details \
     --db-cluster-identifier=new_cluster_id \
     --restore-type=copy-on-write \
     --use-latest-restorable-time \
     --kms-key-id=arn:aws:kms:arn_details
   ```

   Für Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time ^
  --kms-key-id=arn:aws:kms:arn_details
```

Das Konto, dem der Verschlüsselungsschlüssel gehört, muss dem Zielkonto die Berechtigung zur Verwendung des Schlüssels mithilfe einer Schlüsselrichtlinie erteilen. Dieser Vorgang ist ähnlich dem Verfahren, mit dem verschlüsselte Snapshots freigegeben werden, mithilfe einer Schlüsselrichtlinie, die dem Zielkonto die Berechtigung zur Verwendung des Schlüssels gewährt. Nachfolgend sehen Sie ein Beispiel für eine Schlüsselrichtlinie.

JSON

```
{
    "Id": "key-policy-1",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
```

```
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

> **Note**
>
> Der Befehl restore-db-cluster-to- point-in-time AWS CLI stellt nur den DB-Cluster wieder her, nicht die DB-Instances für diesen DB-Cluster. Rufen Sie den create-db-instanceBefehl auf, um DB-Instances für den wiederhergestellten DB-Cluster zu erstellen. Geben Sie die ID des wiederhergestellten DB-Clusters in `--db-cluster-identifier` an.
> Sie können DB-Instances erst dann erstellen, wenn der Befehl `restore-db-cluster-to-point-in-time` abgeschlossen ist und der DB-Cluster verfügbar ist.

RDS-API

Um einen Aurora-Cluster zu klonen, der einem anderen AWS Konto gehört

1. Nehmen Sie die Einladung von dem AWS Konto an, dem der DB-Cluster gehört, wie oben gezeigt.

2. Klonen Sie den Cluster, indem Sie den vollständigen ARN des Quell-Clusters im `SourceDBClusterIdentifier`-Parameter der RDS-API-Operation `RestoreDBClusterToPointInTime` angeben.

   Wenn der als `SourceDBClusterIdentifier` übergebene ARN nicht freigegeben wurde, dann wird derselbe Fehler zurückgegeben, wie wenn der angegebene Cluster nicht vorhanden ist.

3. Wenn der Cluster, den Sie klonen, verschlüsselt ist, fügen Sie einen `KmsKeyId`-Parameter ein, um Ihren geklonten Cluster zu verschlüsseln. Dieser `kms-key-id`-Wert kann derselbe sein, der zur Verschlüsselung des ursprünglichen DB-Clusters verwendet wurde, oder Ihr eigener KMS-Schlüssel. Ihr Konto muss über die Berechtigung zum Verwenden dieses Verschlüsselungsschlüssels verfügen.

   Wenn Sie ein Volume klonen, muss das Zielkonto die Berechtigung haben, den Verschlüsselungsschlüssel zu verwenden, der zum Verschlüsseln des Quellclusters verwendet wird. Aurora verschlüsselt den neuen geklonten Cluster mit dem in `KmsKeyId` angegebenen Verschlüsselungsschlüssel.

   Das Konto, dem der Verschlüsselungsschlüssel gehört, muss dem Zielkonto die Berechtigung zur Verwendung des Schlüssels mithilfe einer Schlüsselrichtlinie erteilen. Dieser Vorgang ist ähnlich dem Verfahren, mit dem verschlüsselte Snapshots freigegeben werden, mithilfe einer Schlüsselrichtlinie, die dem Zielkonto die Berechtigung zur Verwendung des Schlüssels gewährt. Nachfolgend sehen Sie ein Beispiel für eine Schlüsselrichtlinie.

   JSON

```
{
    "Id": "key-policy-1",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
```

```
                "kms:CreateGrant",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:user/KeyUser",
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

> **Note**
>
> Der Vorgang „DBClusterToPointInTimeRDS-API wiederherstellen“ stellt nur den DB-Cluster wieder her, nicht die DB-Instances für diesen DB-Cluster. Um DB-Instances für den wiederhergestellten DB-Cluster zu erstellen, rufen Sie den Vorgang DBInstance RDS-API erstellen auf. Geben Sie die ID des wiederhergestellten DB-Clusters in `DBClusterIdentifier` an. Sie können DB-Instances erst erstellen, nachdem die

Operation `RestoreDBClusterToPointInTime` abgeschlossen ist und der DB-Cluster verfügbar ist.

Überprüfen, ob ein DB-Cluster ein kontoübergreifender Klon ist

Das `DBClusters`-Objekt gibt an, ob der jeweilige Cluster ein kontoübergreifender Klon ist. Sie können die Cluster, für die Sie die Berechtigungen zum Klonen haben, anzeigen, indem Sie die `include-shared`-Option beim Ausführen des RDS-CLI-Befehls `describe-db-clusters` verwenden. Sie können jedoch die meisten Konfigurationsdetails für solche Cluster nicht anzeigen.

AWS CLI

Überprüfen, ob ein DB-Cluster ein kontoübergreifender Klon ist

- Rufen Sie den RDS-CLI-Befehl auf `describe-db-clusters`.

  Das folgende Beispiel zeigt, wie tatsächliche oder potenzielle kontoübergreifende Klon-DB-Cluster in der `describe-db-clusters`-Ausgabe angezeigt werden. Bei vorhandenen Clustern, die Ihrem AWS Konto gehören, gibt das `CrossAccountClone` Feld an, ob es sich bei dem Cluster um einen Klon eines DB-Clusters handelt, der einem anderen AWS Konto gehört.

  In einigen Fällen kann ein Eintrag in dem `DBClusterArn` Feld eine andere AWS Kontonummer als Ihre haben. In diesem Fall steht dieser Eintrag für einen Cluster, der einem anderen AWS Konto gehört und den Sie klonen können. Solche Einträge haben wenige andere Felder als `DBClusterArn`. Wenn Sie einen geklonten Cluster erstellen, geben Sie dieselben `StorageEncrypted`-, `Engine`- und `EngineVersion`-Werte wie im Original-Cluster an.

```
$aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
```

```
            "EngineVersion": "8.0.mysql_aurora.3.02.0",
            "CrossAccountClone": true,
...
        },
        {
            "StorageEncrypted": false,
            "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0
    ]
}
```

RDS-API

Überprüfen, ob ein DB-Cluster ein kontoübergreifender Klon ist

- Rufen Sie die RDS-API-Operation Describe aufDBClusters.

  Bei vorhandenen Clustern, die Ihrem AWS Konto gehören, gibt das `CrossAccountClone` Feld an, ob es sich bei dem Cluster um einen Klon eines DB-Clusters handelt, der einem anderen AWS Konto gehört. Einträge mit einer anderen AWS Kontonummer im `DBClusterArn` Feld stehen für Cluster, die Sie klonen können und die anderen AWS Konten gehören. Diese Einträge haben wenige andere Felder als `DBClusterArn`. Wenn Sie einen geklonten Cluster erstellen, geben Sie dieselben `StorageEncrypted`-, `Engine`- und `EngineVersion`-Werte wie im Original-Cluster an.

  Das folgende Beispiel zeigt einen Rückgabewert, der sowohl tatsächliche als auch potenzielle geklonte Cluster zeigt.

```
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
```

```
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0"
      }
  ]
}
```

Integration von Aurora mit anderen AWS Diensten

Integrieren Sie Amazon Aurora mit anderen AWS Services, sodass Sie Ihren Aurora-DB-Cluster erweitern können, um zusätzliche Funktionen in der AWS Cloud zu nutzen.

Themen

- Integration von AWS Diensten mit Amazon Aurora MySQL
- Integration von AWS Diensten mit Amazon Aurora PostgreSQL

Integration von AWS Diensten mit Amazon Aurora MySQL

Amazon Aurora MySQL lässt sich in andere AWS Dienste integrieren, sodass Sie Ihren Aurora MySQL-DB-Cluster erweitern können, um zusätzliche Funktionen in der AWS Cloud zu nutzen. Ihr Aurora MySQL-DB-Cluster kann AWS Dienste verwenden, um Folgendes zu tun:

- Rufen Sie synchron oder asynchron eine AWS Lambda Funktion mit den nativen Funktionen auf oder. `lambda_sync` `lambda_async` Oder rufen Sie asynchron eine AWS Lambda -Funktion mit der Prozedur `mysql.lambda_async` auf.
- Laden Sie Daten aus Text- und XML-Dateien, die in einem Amazon S3-Bucket gespeichert sind, mithilfe des Befehls `LOAD DATA FROM S3` oder `LOAD XML FROM S3` in Ihren DB-Cluster.
- Speichern Sie Daten in die Textdateien, die in einem Amazon S3-Bucket aus Ihrem DB-Cluster gespeichert sind, mithilfe des Befehls `SELECT INTO OUTFILE S3`.
- Automatisches Hinzufügen oder Entfernen von Aurora-Replicas mit Auto Scaling von Anwendungen. Weitere Informationen finden Sie unter Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten.

Weitere Informationen zur Integration von Aurora MySQL mit anderen AWS Diensten finden Sie unterIntegrieren von Amazon Aurora MySQL in anderen AWS-Services.

Integration von AWS Diensten mit Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL lässt sich in andere AWS Services integrieren, sodass Sie Ihren Aurora PostgreSQL-DB-Cluster erweitern können, um zusätzliche Funktionen in der Cloud zu nutzen. AWS Ihr Aurora PostgreSQL-DB-Cluster kann AWS Dienste verwenden, um Folgendes zu tun:

- Performance bei Ihren relationalen Datenbank-Workloads mit Performance Insights schnell ermitteln, anzeigen und beurteilen.
- Automatisches Hinzufügen oder Entfernen von Aurora Replicas mit Aurora Amazon EC2 Auto Scaling. Weitere Informationen finden Sie unter Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten.

Weitere Informationen zur Integration von Aurora PostgreSQL mit anderen AWS Diensten finden Sie unter. Integrieren von Amazon Aurora PostgreSQL in anderen AWS-Services

Warten eines Amazon Aurora-DB-Clusters

Amazon RDS führt in regelmäßigen Abständen Wartungsarbeiten an Amazon-RDS-Ressourcen durch. In den folgenden Themen werden diese Wartungsaktionen und ihre Anwendung beschrieben.

Übersicht über Wartungsupdates für DB-Cluster

Die Wartung beinhaltet in den meisten Fällen Aktualisierungen der folgenden Ressourcen in Ihrem DB-Cluster:

- Zugrundeliegende Hardware
- Zugrundeliegendes Betriebssystem
- Datenbank-Engine-Version

Häufig werden Betriebssystemupdates wegen Sicherheitsproblemen herausgegeben. Wir empfehlen, sie so schnell wie möglich zu installieren. Weitere Informationen über unterstützte Betriebssystemupdates finden Sie unter the section called “Betriebssystemupdates”.

Themen

- Offline-Ressourcen während Wartungsupdates
- Aufgeschobene Änderungen an der DB-Instance und dem DB-Cluster
- Eventuelle Konsistenz für die API DescribePendingMaintenanceActions

Offline-Ressourcen während Wartungsupdates

Einige Wartungselemente erfordern, dass Amazon RDS Ihren DB-Cluster für kurze Zeit in den Offlinebetrieb versetzt. Zu den Wartungselementen, für die eine Ressource offline sein muss, gehört z. B. das Ausführen erforderlicher Patches für das Betriebssystem oder die Datenbank. Das erforderliche Patching wird automatisch und nur für Patches eingeplant, welche die Sicherheit und Instance-Zuverlässigkeit betreffen. Solche Patches treten selten auf, in der Regel einmal alle paar Monate. Es ist selten mehr als ein Bruchteil Ihres Wartungsfensters dafür erforderlich.

Aufgeschobene Änderungen an der DB-Instance und dem DB-Cluster

Aufgeschobene DB-Cluster- und DB-Instance-Änderungen, die nicht sofort zur Anwendung kommen sollen, werden während des Wartungszeitraums umgesetzt. Sie können beispielsweise wählen,

DB-Instance-Klassen oder Cluster- oder DB-Parametergruppen während des Wartungsfensters zu ändern. Solche Änderungen, die Sie mit der Einstellung für ausstehenden Neustart angeben, werden nicht in der Liste der ausstehenden Wartungsarbeiten angezeigt. Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Um die Änderungen zu sehen, die für das nächste Wartungsfenster noch ausstehen, verwenden Sie den describe-db-clusters AWS CLI Befehl und überprüfen Sie das `PendingModifiedValues` Feld.

Eventuelle Konsistenz für die API DescribePendingMaintenanceActions

Die `DescribePendingMaintenanceActions`-API von Amazon RDS folgt einem eventuellen Konsistenzmodell. Dies bedeutet, dass das Ergebnis des `DescribePendingMaintenanceActions`-Befehls möglicherweise nicht sofort für alle nachfolgenden RDS-Befehle, die Sie ausführen, sichtbar ist. Denken Sie daran, wenn Sie `DescribePendingMaintenanceActions` unmittelbar nach der Verwendung eines vorherigen API-Befehls verwenden.

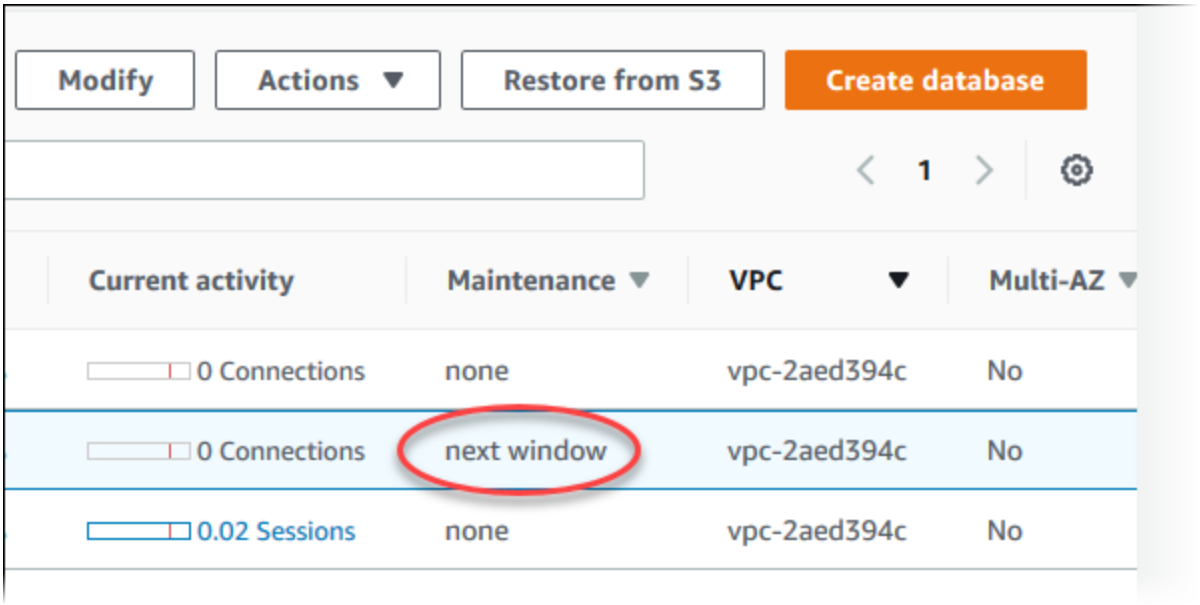
Eine letztendliche Konsistenz kann sich auf die Art und Weise auswirken, wie Sie Ihre Wartungsupdates verwaltet haben. Wenn Sie beispielsweise den `ApplyPendingMaintenanceActions`-Befehl ausführen, um die Version der Datenbank-Engine für eine einen DB-Cluster zu aktualisieren, wird dieser irgendwann für `DescribePendingMaintenanceActions` sichtbar sein. In diesem Szenario zeigt `DescribePendingMaintenanceActions` möglicherweise, dass die Wartungsaktion nicht angewendet wurde, obwohl dies der Fall ist.

Gehen Sie wie folgt vor, um die letztendliche Datenkonsistenz zu verwalten:

- Bestätigen Sie den Status Ihrer , bevor Sie einen Befehl ausführen, um ihn zu ändern. Führen Sie den entsprechenden Befehl `DescribePendingMaintenanceActions` mit einem exponentiellen Backoff-Algorithmus aus, um sicherzustellen, dass Sie genügend Zeit für die Ausbreitung des vorherigen Befehls im System einplanen. Führen Sie dazu den Befehl `DescribePendingMaintenanceActions` wiederholt aus, wobei Sie mit einer Wartezeit von einigen Sekunden beginnen und die Wartezeit schrittweise auf bis zu fünf Minuten erhöhen.
- Fügen Sie Wartezeiten zwischen aufeinanderfolgenden Befehlen hinzu, auch wenn der Befehl `DescribePendingMaintenanceActions` eine genaue Antwort zurückgibt. Wenden Sie einen exponentiellen Backoff-Algorithmus an, der mit einer Wartezeit von einigen Sekunden beginnt, und erhöhen Sie die Wartezeit schrittweise auf etwa fünf Minuten.

Anzeigen ausstehender Wartungs-Updates

Prüfen Sie mithilfe der RDS-Konsole, der oder der RDS-API, ob ein Wartungsupdate für Ihren verfügbar ist. AWS CLI Wenn ein Update verfügbar ist, wird dies in der Amazon-RDS-Konsole in der Spalte Wartung für die den DB-Cluster angegeben, wie in der folgenden Abbildung gezeigt.



Wenn für eienn DB-Cluster keine Aktualisierung verfügbar ist, lautet ihr bzw. sein Spaltenwert none (keine).

Wenn eine Wartungsaktualisierung für einen DB-Cluster verfügbar ist, sind die folgenden Spaltenwerte möglich:

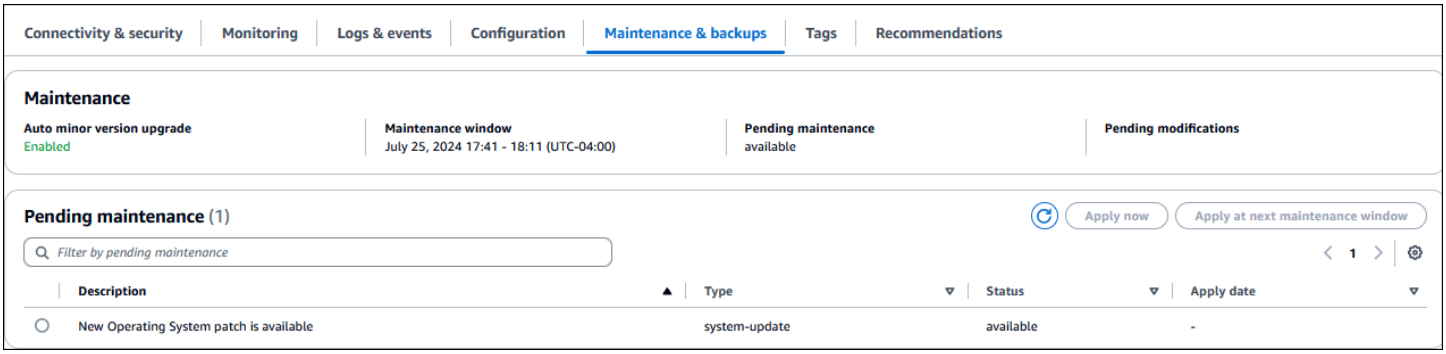
- required (erforderlich) – Die Wartungsaktion wird auf die Ressource angewendet und kann nicht unbegrenzt aufgeschoben werden.
- available (verfügbar) – Die Wartungsaktion ist verfügbar, wird aber nicht automatisch auf die Ressource angewandt. Sie können sie manuell anwenden.
- next window (nächstes Fenster) – Die Wartungsmaßnahme wird im nächsten Wartungsfenster auf die Ressource angewandt.
- Läuft – Die Wartungsmaßnahme wird derzeit auf die Ressource angewandt.

Wenn ein Update verfügbar ist, können Sie eine der folgenden Aktionen durchführen:

- Wenn der Wartungswert nächstes Fenster ist, schieben Sie die Wartungselemente durch Auswahl von Upgrade aufschieben in Aktionen auf. Sie können eine bereits gestartete Wartungsaktion nicht verschieben.
- Wenden Sie die Wartungsaktionen sofort an.
- Wenden Sie die Wartungsaktionen für des nächsten Wartungsfensters an.
- Keine Aktion.

Um eine Maßnahme zu ergreifen, verwenden Sie AWS-Managementkonsole

1. Wählen Sie die DB-Instance oder den DB-Cluster aus, um die zugehörigen Details anzuzeigen.
2. Wählen Sie Wartung und Sicherungen aus. Die ausstehenden Wartungsaktionen werden angezeigt.
3. Wählen Sie die zu ergreifende Maßnahme aus und wählen Sie dann den Zeitpunkt ihrer Anwendung aus.



Der Wartungszeitraum legt fest, wann die ausstehenden Operationen gestartet werden, gibt aber keine Gesamtlaufzeit für diese Operationen vor. Wartungsarbeiten werden nicht zwingend vor Ende des Wartungszeitraums abgeschlossen. Sie können daher über die angegebene Endzeit hinaus fortgesetzt werden. Weitere Informationen finden Sie unter Amazon-RDS-Wartungsfenster.

Sie können auch überprüfen, ob ein Wartungsupdate für Ihren verfügbar ist, indem Sie den describe-pending-maintenance-actions AWS CLI Befehl ausführen.

Weitere Informationen zum Anwenden von Wartungsupdates finden Sie unter Anwenden von Updates auf einen DB-Cluster.

Wartungsaktionen für Amazon Aurora

Die folgenden Wartungsaktionen gelten für Aurora-DB-Cluster:

- `os-upgrade` – Aktualisieren Sie die Betriebssysteme aller DB-Instances im DB-Cluster mithilfe von fortlaufenden Upgrades. Weitere Informationen finden Sie unter the section called “Betriebssystemupdates”.
- `system-update` – Patchen Sie die DB-Engine für Aurora PostgreSQL.

Die folgenden Wartungsaktionen gelten für Aurora-DB-Instances:

- `ca-certificate-rotation` – Aktualisieren Sie das Zertifikat der Amazon-RDS-Zertifizierungsstelle für die DB-Instance.
- `hardware-maintenance` – Führen Sie Wartungsarbeiten an der zugrunde liegenden Hardware für die DB-Instance durch.
- `system-update` – Aktualisieren Sie das Betriebssystem für die DB-Instance.

Auswahl der Häufigkeit der Aurora MySQL-Wartungsupdates

Sie können steuern, ob Aurora MySQL-Upgrades für jeden DB-Cluster häufig oder selten stattfinden. Die beste Wahl hängt von Ihrer Verwendung von Aurora MySQL und den Prioritäten für Ihre Anwendungen ab, die unter Aurora ausgeführt werden. Informationen über die Aurora-MySQL-LTS-Versionen (Langzeitstabilität), die weniger häufige Upgrades erfordern, finden Sie unterAurora MySQL Long-Term Support- (LTS, Langzeit-Support) Versionenaus.

Sie können einen Aurora MySQL-Cluster selten aktualisieren, wenn einige oder alle der folgenden Bedingungen zutreffen:

- Ihr Testzyklus für Ihre Anwendung dauert bei einer Aktualisierung der Aurora MySQL-Datenbank-Engine sehr lange.
- Sie haben viele DB-Cluster oder viele Anwendungen, die alle auf der gleichen Aurora MySQL-Version ausgeführt werden. Sie ziehen es vor, alle Ihre DB-Cluster und die zugehörigen Anwendungen gleichzeitig upzugraden.
- Sie verwenden sowohl Aurora MySQL als auch RDS für MySQL. Sie möchten, dass die Aurora-MySQL-Cluster und die DB-Instances von RDS für MySQL mit demselben MySQL-Level kompatibel bleiben.
- Ihre Aurora MySQL-Anwendung befindet sich in der Produktion oder ist anderweitig geschäftskritisch. Sie können sich keine Ausfallzeiten für Upgrades außerhalb seltener Fälle für kritische Patches leisten.

- Ihre Aurora MySQL-Anwendung ist nicht durch Leistungsprobleme oder Funktionslücken eingeschränkt, die in nachfolgenden Aurora MySQL-Versionen behoben werden.

Wenn die vorstehenden Faktoren auf Ihre Situation zutreffen, können Sie die Anzahl der erzwungenen Upgrades für einen Aurora MySQL DB-Cluster begrenzen. Dies geschieht, indem Sie beim Erstellen oder Aktualisieren dieses DB-Clusters eine bestimmte Aurora MySQL-Version wählen, die als "Long-Term Support" (LTS)-Version bekannt ist. Dadurch wird die Anzahl der Upgrade-Zyklen, Testzyklen und upgradebedingten Ausfälle für diesen DB-Cluster minimiert.

Sie können einen Aurora MySQL-Cluster häufig aktualisieren, wenn einige oder alle der folgenden Bedingungen zutreffen:

- Der Testzyklus für Ihre Anwendung ist unkompliziert und kurz.
- Ihre Anwendung befindet sich noch in der Entwicklungsphase.
- Ihre Datenbankumgebung verwendet eine Vielzahl von Aurora MySQL-Versionen oder Aurora MySQL- und RDS für MySQL-Versionen. Jeder Aurora MySQL-Cluster hat seinen eigenen Upgrade-Zyklus.
- Sie warten auf spezifische Leistungs- oder Funktionsverbesserungen, bevor Sie Ihre Nutzung von Aurora MySQL erhöhen.

Wenn die vorhergehenden Faktoren auf Ihre Situation zutreffen, können Sie Aurora aktivieren, um wichtige Upgrades häufiger anzuwenden. Dazu führen Sie ein Upgrade eines DB-Clusters von Aurora MySQL auf eine aktuellere Version von Aurora MySQL als die LTS-Version durch. Auf diese Weise stehen Ihnen die neuesten Leistungssteigerungen, Bugfixes und Funktionen schneller zur Verfügung.

Amazon-RDS-Wartungsfenster

Das Wartungsfenster ist ein wöchentliches Zeitintervall, in dem alle Systemänderungen übernommen werden. Jede DB–jeder DB-Cluster verfügt über ein wöchentliches Wartungsfenster. Das Wartungsfenster ist eine Möglichkeit, zu kontrollieren, wann Änderungen und Software-Patches auftreten. Weitere Informationen über Wartungsfenster finden Sie unter Anpassen des bevorzugten DB-Cluster-Wartungsfensters.

RDS verbraucht während der Wartung einige Ressourcen auf Ihrem DB-Cluster. Sie können einen minimalen Einfluss auf die Leistung beobachten. Bei einer DB-Instance kann in seltenen Fällen ein Multi-AZ-Failover erforderlich sein, damit ein Wartungs-Update abgeschlossen werden kann.

Wenn ein Wartungsereignis für eine bestimmte Woche geplant ist, wird es während des 30-minütigen Wartungsfensters eingeleitet, das Sie festlegen. Die meisten Wartungsereignisse werden auch während des 30-minütigen Wartungsfensters abgeschlossen, obwohl größere Wartungsereignisse länger als 30 Minuten dauern können. Das Wartungsfenster wird angehalten, wenn die der DB-Cluster gestoppt wird.

Das 30-minütige Wartungsfenster wird zufällig aus einem 8-Stunden-Zeitraum pro Region ausgewählt. Wenn Sie beim Erstellen des DB-Clusters kein Wartungsfenster angeben, legt RDS ein 30-minütiges Wartungsfenster an einem zufällig ausgewählten Wochentag fest.

In der folgenden Tabelle sind die Zeitblöcke für die einzelnen AWS-Region Zeitblöcke aufgeführt, denen die Standard-Wartungsfenster zugewiesen wurden.

Name der Region	Region	Zeitblock
USA Ost (Nord-Virginia)	us-east-1	03:00 bis 11:00 Uhr UTC
USA Ost (Ohio)	us-east-2	03:00 - 11:00 UTC
USA West (Nordkalifornien)	us-west-1	06:00 - 14:00 UTC
USA West (Oregon)	us-west-2	06:00 - 14:00 UTC
Africa (Cape Town)	af-south-1	03:00 - 11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00 - 14:00 UTC
Asien-Pazifik (Hyderabad)	ap-south-2	06:30 – 14:30 Uhr UTC
Asien-Pazifik (Jakarta)	ap-southeast-3	08:00–16:00 Uhr UTC
Asien-Pazifik (Malaysia)	ap-southeast-5	09:00 bis 17:00 Uhr UTC

Name der Region	Region	Zeitblock
Asien-Pazifik (Melbourne)	ap-southeast-4	11:00–19:00 Uhr UTC
Asien-Pazifik (Mumbai)	ap-south-1	06:00 - 14:00 UTC
Asien-Pazifik (Neuseeland)	ap-southeast-6	13:00 - 21:00 UTC
Asia Pacific (Osaka)	ap-northeast-3	22:00 - 23:59 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00 - 21:00 UTC
Asien-Pazifik (Singapur)	ap-southeast-1	14:00 - 22:00 UTC
Asien-Pazifik (Sydney)	ap-southeast-2	12:00 - 20:00 UTC
Asien-Pazifik (Taipeh)	ap-east-2	9:00 bis 17:00 Uhr UTC
Asien-Pazifik (Thailand)	ap-southeast-7	08:00 bis 16:00 Uhr UTC
Asien-Pazifik (Tokio)	ap-northeast-1	13:00 - 21:00 UTC
Canada (Central)	ca-central-1	03:00 bis 11:00 Uhr UTC
Kanada West (Calgary)	ca-west-1	18:00 - 02:00 UTC
China (Beijing)	cn-north-1	06:00 - 14:00 UTC
China (Ningxia)	cn-northwest-1	06:00 - 14:00 UTC
Europe (Frankfurt)	eu-central-1	21:00 - 05:00 UTC
Europa (Irland)	eu-west-1	22:00 - 06:00 UTC

Name der Region	Region	Zeitblock
Europe (London)	eu-west-2	22:00 bis 06:00 Uhr UTC
Europa (Milan)	eu-south-1	02:00 - 10:00 UTC
Europa (Paris)	eu-west-3	23:59 - 07:29 UTC
Europa (Spain)	eu-south-2	02:00 - 10:00 UTC
Europe (Stockholm)	eu-north-1	23:00 - 07:00 UTC
Europa (Zürich)	eu-central-2	02:00 - 10:00 UTC
Israel (Tel Aviv)	il-central-1	03:00 bis 11:00 Uhr UTC
Mexiko (Zentral)	mx-central-1	19:00 bis 03:00 Uhr UTC
Naher Osten (Bahrain)	me-south-1	06:00 - 14:00 UTC
Naher Osten (VAE)	me-central-1	05:00–13:00 UHR UTC
Südamerika (São Paulo)	sa-east-1	00:00 - 08:00 UTC
AWS GovCloud (US-Ost)	us-gov-east-1	17:00 - 01:00 UTC
AWS GovCloud (US-West)	us-gov-west-1	06:00 - 14:00 UTC

Themen

- Anpassen des bevorzugten DB-Cluster-Wartungsfensters

Anpassen des bevorzugten DB-Cluster-Wartungsfensters

Der Wartungszeitraum für ein Aurora-DB-Cluster sollte in den Zeitraum mit der geringsten Nutzung fallen und daher unter Umständen von Zeit zu Zeit geändert werden. Ihr DB-Cluster ist

während dieser Zeit nur dann nicht verfügbar, wenn die Updates, die angewendet werden, einen Ausfall erfordern. Der Ausfall dauert die minimale Zeit, die erforderlich ist, um die notwendigen Aktualisierungen vorzunehmen.

> **Note**
>
> Für Upgrades zur Datenbank-Engine verwaltet Amazon Aurora das bevorzugte Wartungsfenster für ein DB-Cluster und nicht für einzelne Instances.

Konsole

So passen Sie den bevorzugten DB-Cluster-Wartungszeitraum an

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie das DB-Cluster aus, für das Sie den Wartungszeitraum ändern möchten.
4. Wählen Sie Ändern aus.
5. Aktualisieren Sie im Bereich Wartung den Wartungszeitraum.
6. Klicken Sie auf Weiter.

   Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen.
7. Um die Änderungen sofort auf das Wartungsfenster anzuwenden, wählen Sie im Abschnitt Schedule of modifications (Änderungsplan) die Option Immediately (Sofort) .
8. Wählen Sie Cluster ändern aus, um Ihre Änderungen zu speichern.

   Klicken Sie anderenfalls auf Zurück, um Ihre Änderungen zu bearbeiten, oder klicken Sie auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

Um das bevorzugte Wartungsfenster für den DB-Cluster anzupassen, verwenden Sie den AWS CLI `modify-db-cluster`Befehl mit den folgenden Parametern:

- `--db-cluster-identifier`
- `--preferred-maintenance-window`

Example

Im folgenden Codebeispiel wird der Wartungszeitraum auf dienstags von 4:00 Uhr bis 4:30 Uhr (UTC) festgelegt.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-cluster \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

Für Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS-API

Verwenden Sie die Amazon-RDS-API-Operaion `ModifyDBCluster` mit den folgenden Parametern, um das bevorzugte Wartungsfenster für Ihren DB-Cluster anzupassen:

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

Anwenden von Updates auf einen DB-Cluster

Mit Amazon RDS können Sie auswählen, zu welchem Zeitpunkt Wartungsoperationen angewendet werden sollen. Mithilfe der, oder RDS-API können Sie entscheiden AWS-Managementkonsole AWS CLI, wann Amazon RDS Updates einführt.

Konsole

Verwalten eines Update für einen DB-Cluster

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie den DB-Cluster aus, für die/den ein erforderliches Update angegeben ist.
4. Wählen Sie unter Aktionen eine der folgenden Optionen:

   - Jetzt patchen
   - Im nächsten Fenster patchen

   > **Note**
   >
   > Wenn Sie Im nächsten Wartungszeitraum Upgrade ausführen auswählen und die Installation des Updates aufschieben möchten, können Sie die Option Upgrade verschieben auswählen. Sie können eine Wartungsaktion nicht verschieben, wenn sie bereits gestartet wurde.
   > Um eine Wartungsaktion abzubrechen, ändern Sie die DB-Instance und deaktivieren Sie Auto minor version upgrade (Automatisches Upgrade einer Unterversion).

AWS CLI

Verwenden Sie den apply-pending-maintenance-action AWS CLI Befehl, um ein ausstehendes Update auf einen anzuwenden.

Example

Für Linux, macOS oder Unix:

```
aws rds apply-pending-maintenance-action \
```

```
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
    --apply-action system-update \
    --opt-in-type immediate
```

Für Windows:

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> Um eine Wartungsaktion zu verschieben, geben Sie `undo-opt-in` für `--opt-in-type` an. Sie können `undo-opt-in` nicht für `--opt-in-type` angeben, wenn die Wartungsaktion bereits gestartet wurde.
> Um eine Wartungsaktion abzubrechen, führen Sie den modify-db-instance AWS CLI Befehl aus und geben Sie Folgendes an`--no-auto-minor-version-upgrade`.

Verwenden Sie den describe-pending-maintenance-actions AWS CLI Befehl, um eine Liste der Ressourcen zurückzugeben, für die mindestens ein Update aussteht.

Example

Für Linux, macOS oder Unix:

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Für Windows:

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Sie können auch eine Liste von Ressourcen für einen zurückgeben, indem Sie den `--filters` Parameter des `describe-pending-maintenance-actions` AWS CLI Befehls angeben. Der Befehl hat das folgende Format für `--filters`: `Name=`*`filter-name`*`,Value=`*`resource-id`*`,...`.

Für den `Name`-Parameter eines Filters sind folgende Werte gültig:

- `db-instance-id`— Akzeptiert eine Liste von DB-Instance-Identifikatoren oder Amazon-Ressourcennamen (ARNs). Die zurückgegebene Liste enthält nur ausstehende Wartungsaktionen für die DB-Instances, die durch diese Identifikatoren oder identifiziert wurden. ARNs
- `db-cluster-id`— Akzeptiert eine Liste von DB-Cluster-Identifikatoren oder ARNs für Amazon Aurora. Die zurückgegebene Liste enthält nur ausstehende Wartungsaktionen für die DB-Cluster, die anhand dieser Kennungen oder identifiziert wurden. ARNs

In dem folgenden Beispiel wird eine Liste der aussehenden Wartungsaktionen für die DB-Cluster `sample-cluster1` und `sample-cluster2` zurückgegeben.

Example

Für Linux, macOS oder Unix:

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Für Windows:

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS-API

Rufen Sie die Amazon-RDS-API-Operation `ApplyPendingMaintenanceAction` auf, um ein Update auf einem DB-Cluster zu installieren.

Rufen Sie die Amazon-RDS-API-Operation `DescribePendingMaintenanceActions` auf, um eine Liste der Ressourcen zurückzugeben, für die mindestens ein Update aussteht.

Automatische Nebenversions-Upgrades für Aurora-DB-Cluster

Die Einstellung Automatisches Unterversion-Upgrade gibt an, ob Aurora Upgrades automatisch auf den DB-Cluster anwendet. Diese Upgrades umfassen neue Unterversionen mit zusätzlichen Funktionen und Patches mit Bugfixes.

Durch automatische Nebenversions-Upgrades wird Ihre Datenbank regelmäßig auf die neuesten Datenbank-Engine-Versionen aktualisiert. Das Upgrade beinhaltet jedoch möglicherweise nicht immer die aktuelle Datenbank-Engine-Version. Wenn Sie Ihre Datenbanken zu bestimmten Zeiten auf bestimmten Versionen belassen müssen, empfehlen wir Ihnen, ein manuelles Upgrade auf

die Datenbankversionen durchzuführen, die Sie gemäß Ihrem erforderlichen Zeitplan benötigen. Bei kritischen Sicherheitsproblemen oder wenn eine Version ihr end-of-support Datum erreicht, führt Aurora möglicherweise ein Upgrade für eine Nebenversion durch, auch wenn Sie die Option Automatisches Upgrade der Nebenversion nicht aktiviert haben. Weitere Informationen finden Sie in der Upgrade-Dokumentation zu Ihrer spezifischen Datenbank-Engine.

Siehe Upgrade der Nebenversion oder der Patch-Ebene eines Aurora MySQL-DB-Clusters und Durchführen eines Nebenversions-Upgrades.

> **Note**
>
> Die globale Aurora-Datenbank unterstützt keine automatischen Unterversion-Upgrades.

Diese Einstellung ist standardmäßig aktiviert. Wählen Sie für jeden neuen DB-Cluster den entsprechenden Wert für diese Einstellung aus. Dieser Wert basiert auf der Wichtigkeit, der erwarteten Lebensdauer und dem Umfang der Verifizierungstests, die Sie nach jedem Upgrade durchführen.

Anleitungen zum Ein- und Ausschalten der Einstellung Automatisches Unterversion-Upgrade finden Sie im Folgenden:

- Aktivieren automatischer Nebenversions-Upgrades für einen Aurora-DB-Cluster
- Aktivieren automatischer Unterversion-Upgrades für einzelne DB-Instances in einem Aurora-DB-Cluster

> **Important**
>
> Wir empfehlen dringend, dass Sie diese Einstellung für neue und bestehende DB-Cluster auf den DB-Cluster anwenden und nicht einzeln auf die DB-Instances im Cluster. Wenn diese Einstellung für eine DB-Instance in Ihrem Cluster deaktiviert ist, wird der DB-Cluster nicht automatisch aktualisiert.

Die folgende Tabelle zeigt, wie die Einstellung Automatisches Unterversion-Upgrade funktioniert, wenn sie auf Cluster- und Instance-Ebene angewendet wird.

Action	Cluster-Einstellung	Instance-Einstellungen	Wurde der Cluster automatisch aktualisiert?
Sie legen diese Einstellung im DB-Cluster auf „true“ fest.	Wahr	„True“ für alle neuen und bestehenden Instances	Ja
Sie legen diese Einstellung im DB-Cluster auf „false“ fest.	Falsch	„False“ für alle neuen und bestehenden Instances	Nein
Die Einstellung wurde zuvor im DB-Cluster auf „true“ festgelegt. Sie legen sie auf mindestens einer DB-Instance auf „false“ fest.	Ändert sich in „false“	„False“ für eine oder mehrere Instances	Nein
Die Einstellung wurde zuvor im DB-Cluster auf „false“ festgelegt. Sie legen die Einstellung für mindestens eine, aber nicht für alle DB-Instances auf „true“ fest.	Falsch	„True“ für eine oder mehrere Instances, aber nicht für alle Instances	Nein
Die Einstellung wurde zuvor im DB-Cluster auf „false“ festgelegt.	Ändert sich in „true“	„True“ für alle Instances	Ja

Action	Cluster-Einstellung	Instance-Einstellungen	Wurde der Cluster automatisch aktualisiert?
Sie legen die Einstellung für alle DB-Instances auf „true“ fest.			

Automatische Upgrades von Nebenversionen werden im Voraus über ein Amazon-RDS-DB-Clusterereignis mit der Kategorie `maintenance` und der ID `RDS-EVENT-0156` kommuniziert. Weitere Informationen finden Sie unter Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora.

Automatische Aktualisierungen erfolgen während des Wartungsfensters. Wenn die einzelnen DB-Instances im DB-Cluster andere Wartungsfenster haben als das Cluster-Wartungsfenster, hat das Cluster-Wartungsfenster Vorrang.

Weitere Informationen über Engine-Updates für Aurora PostgreSQL finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL.

Weitere Informationen zur Einstellung für automatische Nebenversions-Upgrades für Aurora MySQL finden Sie unter Aktivieren von automatischen Upgrades zwischen Aurora MySQL-Nebenversionen. Weitere Informationen über Engine-Updates für Aurora MySQL finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.

Aktivieren automatischer Nebenversions-Upgrades für einen Aurora-DB-Cluster

Gehen Sie vor, wie im allgemeinen Verfahren unter Ändern des DB-Clusters über die Konsole, die CLI und die API beschrieben.

Konsole

Wählen Sie im Abschnitt Wartung der Seite DB-Cluster ändern das Kontrollkästchen Automatisches Nebenversions-Upgrade aktivieren aus.

AWS CLI

Führen Sie den Befehl modify-db-cluster AWS CLI aus. Geben Sie den Namen Ihres DB-Clusters für die `--db-cluster-identifier`-Option und `true` für die `--auto-minor-version-`

`upgrade`-Option an. Optional können Sie die `--apply-immediately`-Option angeben, um diese Einstellung sofort für Ihren DB-Cluster zu aktivieren.

RDS-API

Rufen Sie den Vorgang „DBClusterAPI modifizieren“ auf und geben Sie den Namen Ihres DB-Clusters für den `DBClusterIdentifier` Parameter und `true` für den `AutoMinorVersionUpgrade` Parameter an. Legen Sie optional den Parameter `ApplyImmediately` auf `true` fest, um diese Einstellung für Ihren DB-Cluster sofort zu aktivieren.

Aktivieren automatischer Unterversion-Upgrades für einzelne DB-Instances in einem Aurora-DB-Cluster

Gehen Sie vor, wie im allgemeinen Verfahren unter Ändern einer DB-Instance in einem DB-Cluster beschrieben.

Konsole

Wählen Sie im Abschnitt Wartung der Seite DB-Instance ändern das Kontrollkästchen Automatisches Nebenversions-Upgrade aktivieren aus.

AWS CLI

Führen Sie den Befehl modify-db-instance AWS CLI aus. Geben Sie den Namen Ihrer DB-Instance für die `--db-instance-identifier`-Option und `true` für die `--auto-minor-version-upgrade`-Option an. Optional können Sie die `--apply-immediately`-Option angeben, um diese Einstellung sofort für Ihre DB-Instance zu aktivieren. Führen Sie einen separaten `modify-db-instance`-Befehl für jede DB-Instance in dem Cluster aus.

RDS-API

Rufen Sie den DBInstance API-Vorgang Modify auf und geben Sie den Namen Ihres DB-Clusters für den `DBInstanceIdentifier` Parameter und `true` für den `AutoMinorVersionUpgrade` Parameter an. Legen Sie optional den Parameter `ApplyImmediately` auf `true` fest, um diese Einstellung für Ihre DB-Instance sofort zu aktivieren. Rufen Sie für jede DB-Instance in dem Cluster eine separate `ModifyDBInstance`-Operation auf.

Sie können einen CLI-Befehl wie den folgenden verwenden, um den Status der Einstellung `AutoMinorVersionUpgrade` für alle DB-Instances in Ihren Aurora-MySQL-Clustern zu überprüfen.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Die Ausgabe dieses Befehls sieht etwa so aus:

```
[
  {
      "DBInstanceIdentifier": "db-writer-instance",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-reader-instance1",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "db-writer-instance2",
      "DBClusterIdentifier": "my-db-cluster-80",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

In diesem Beispiel ist die Einstellung Automatisches Nebenversions-Upgrade aktivieren für den DB-Cluster `my-db-cluster-57` deaktiviert, da sie für eine der DB-Instances im Cluster deaktiviert ist.

Betriebssystemupdates für Aurora-DB-Cluster

DB-Instances in DB-Cluster von Aurora MySQL und Aurora PostgreSQL erfordern gelegentlich Betriebssystemaktualisierungen. Amazon RDS aktualisiert das Betriebssystem auf eine neuere Version, um die Datenbankleistung und der allgemeine Sicherheitsstatus der Kunden zu verbessern. In der Regel dauern die Updates etwa 10 Minuten. Betriebssystemupdates ändern nicht die DB-Engine-Version oder die DB-Instance-Klasse einer DB-Instance.

Es gibt zwei Arten von Betriebssystemupdates, die sich durch die Beschreibung der ausstehenden Wartungsaktion unterscheiden:

- Upgrade der Betriebssystemdistribution – Wird für die Migration auf die neueste unterstützte Hauptversion von Amazon Linux verwendet. Die Beschreibung lautet `New Operating System upgrade is available`.

- Betriebssystem-Patch – Wird verwendet, um verschiedene Sicherheitskorrekturen anzuwenden und manchmal um die Datenbankleistung zu verbessern. Die Beschreibung lautet `New Operating System patch is available`.

Betriebssystemupdates können entweder optional oder obligatorisch sein:

- Ein optionales Update kann jederzeit angewendet werden. Obwohl diese Updates optional sind, empfehlen wir Ihnen, sie regelmäßig anzuwenden, um Ihre RDS-Flotte auf dem neuesten Stand zu halten. RDS wendet diese Updates nicht automatisch an.

  Wenn Sie benachrichtigt werden möchten, sobald ein neuer optionaler System-Patch verfügbar ist, können Sie RDS-EVENT-0230 in der Kategorie „Sicherheitspatch-Ereignis“ abonnieren. Informationen zum Abonnieren von RDS-Ereignissen finden Sie unter Abonnieren von Amazon-RDS-Ereignisbenachrichtigungen.

  > **Note**
  >
  > `RDS-EVENT-0230` gilt nicht für Upgrades der Betriebssystemdistribution.

- Ein obligatorisches Update ist erforderlich. Wir senden vor dem obligatorischen Update eine Benachrichtigung. Die Benachrichtigung kann ein Fälligkeitsdatum enthalten. Planen Sie Ihr Update vor diesem Fälligkeitsdatum. Nach dem angegebenen Fälligkeitsdatum aktualisiert Amazon RDS das Betriebssystem für Ihre DB-Instance während eines Ihrer zugewiesenen Wartungsfenster automatisch auf die neueste Version.

  Upgrades der Betriebssystemdistribution sind obligatorisch.

> **Note**
>
> Es ist möglicherweise erforderlich, im Hinblick auf alle optionalen und obligatorischen Updates auf dem Laufenden zu bleiben, um verschiedene Compliance-Auflagen zu erfüllen. Wir empfehlen Ihnen, alle von RDS zur Verfügung gestellten Updates während Ihrer Wartungsfenster routinemäßig anzuwenden.

Für Aurora-DB-Cluster können Sie die Wartungsoption auf Cluster-Ebene verwenden, um Betriebssystem (BS)-Updates durchzuführen. Suchen Sie auf der Registerkarte Wartung und

Sicherungen nach der Option zur Durchführung von Updates auf Cluster-Ebene, wenn Sie den Namen Ihres DB-Clusters in der Konsole auswählen, oder verwenden Sie den `os-upgrade`-Befehl in der AWS CLI. Diese Methode bewahrt die Leseverfügbarkeit durch fortlaufende Upgrades, bei denen Updates automatisch auf mehrere Leser-DB-Instances gleichzeitig angewendet werden. Um mehrere Failover zu verhindern und unnötige Ausfallzeiten zu reduzieren, aktualisiert Aurora die Writer-DB-Instance zuletzt.

Betriebssystemupdates auf Cluster-Ebene erfolgen während des Wartungsfensters, das Sie für den Cluster angegeben haben. Dadurch werden koordinierte Updates für den gesamten Cluster sichergestellt.

Aus Gründen der Abwärtskompatibilität behält Aurora auch die Wartungsoption auf Instance-Ebene bei. Wir empfehlen jedoch, stattdessen Updates auf Cluster-Ebene zu verwenden. Wenn Sie Updates auf Instance-Ebene verwenden müssen, aktualisieren Sie zuerst die Reader-DB-Instances in einem DB-Cluster und dann die Writer-DB-Instance. Wenn Sie Reader- und Writer-Instances gleichzeitig aktualisieren, erhöhen Sie die Wahrscheinlichkeit von Ausfallzeiten im Zusammenhang mit einem Failover. Suchen Sie auf der Registerkarte Wartung und Sicherungen nach der Option zum Durchführen von Updates auf Instance-Ebene, wenn Sie den Namen Ihrer DB-Instance in der Konsole auswählen, oder verwenden Sie den Befehl `system-update` in der AWS CLI-

Betriebssystemupdates auf Instance-Ebene erfolgen während des Wartungsfensters, das Sie für die jeweilige Instance angegeben haben. Wenn beispielsweise ein Cluster und zwei Reader-Instances unterschiedliche Wartungsfensterzeiten haben, richtet sich ein Betriebssystemupdate auf Cluster-Ebene nach dem Wartungsfenster des Clusters.

Sie können das AWS-Managementkonsole oder das verwenden AWS CLI , um Informationen über die Art des Betriebssystem-Upgrades abzurufen.

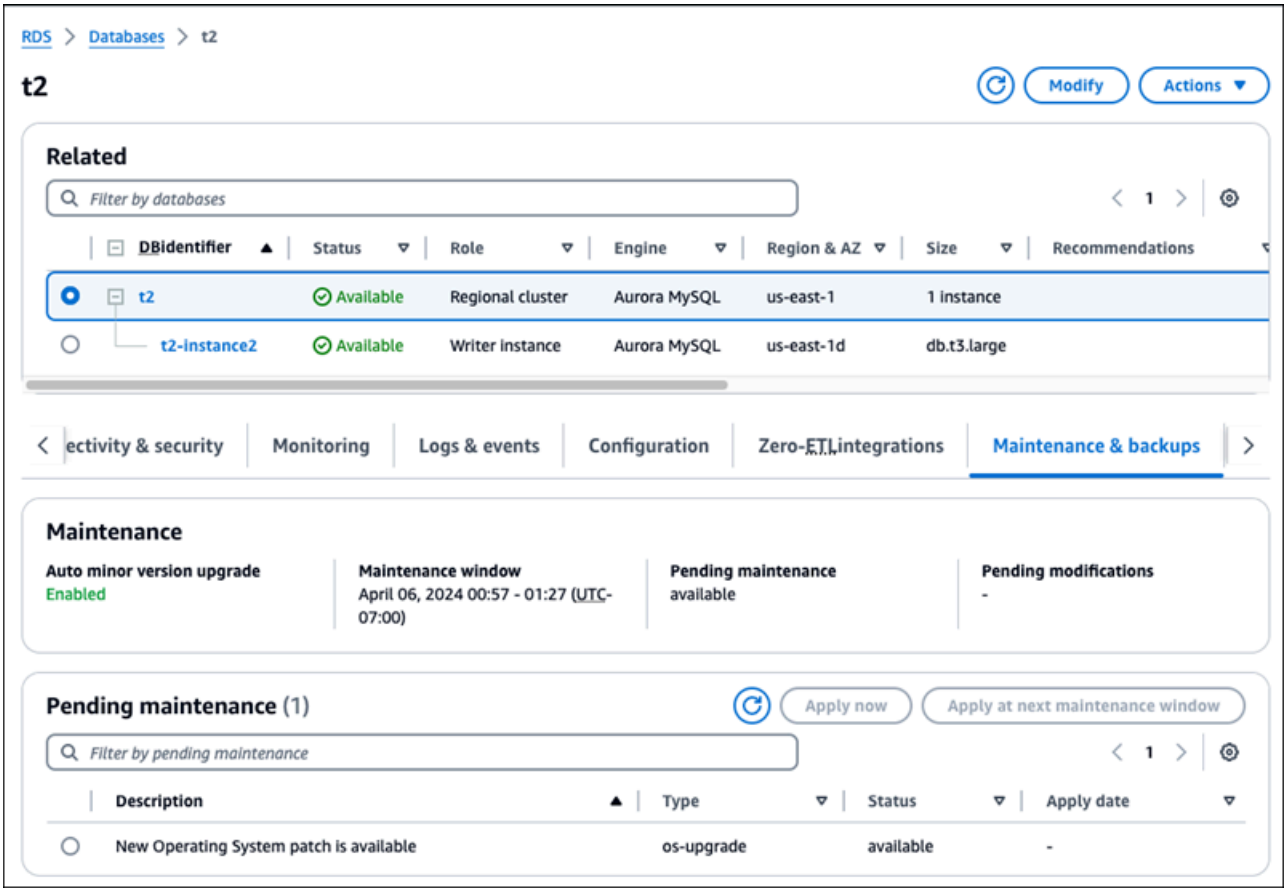
Konsole

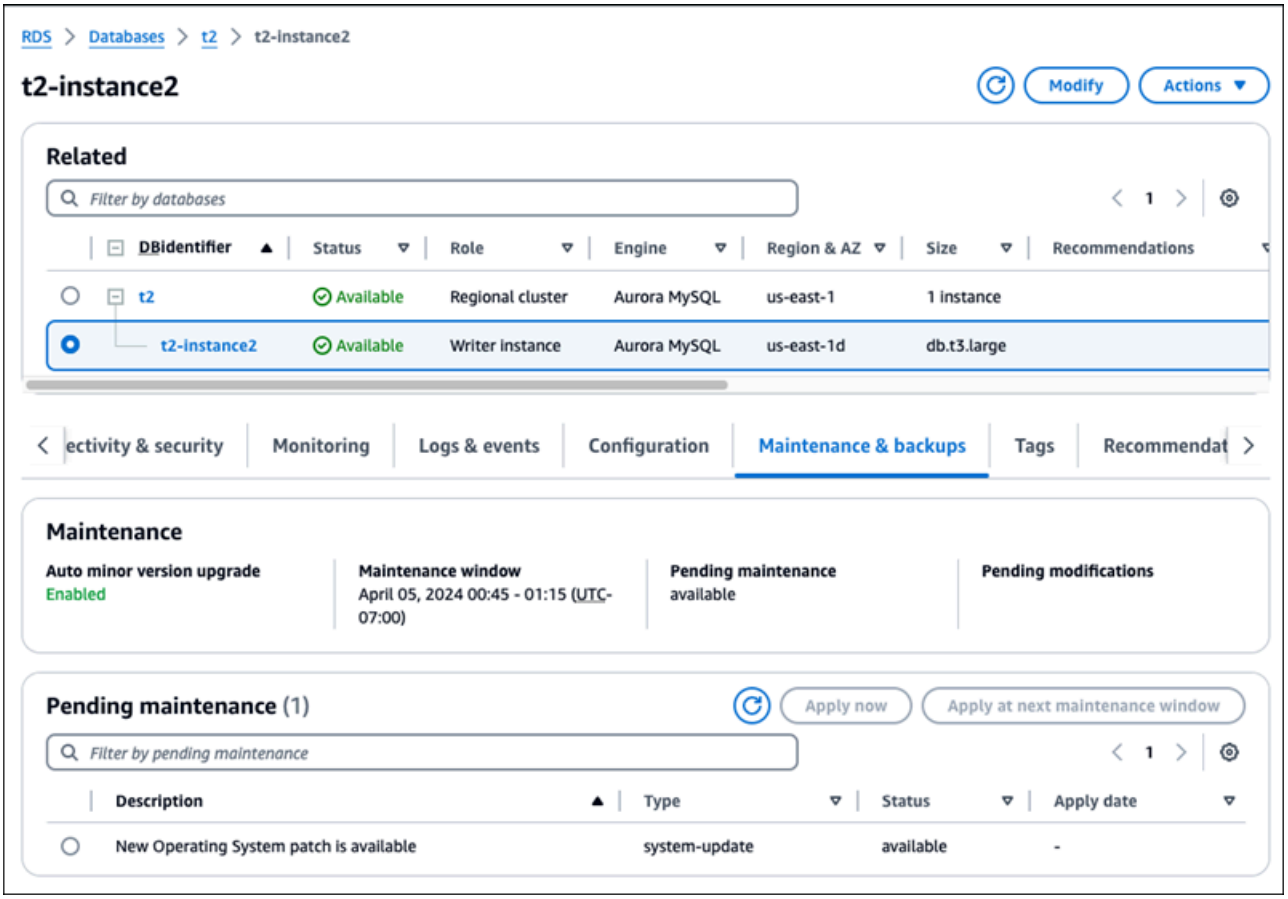
Um Aktualisierungsinformationen zu erhalten, verwenden Sie den AWS-Managementkonsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und anschließend die MySQL-DB-Instance aus.
3. Wählen Sie Wartung und Sicherungen aus.

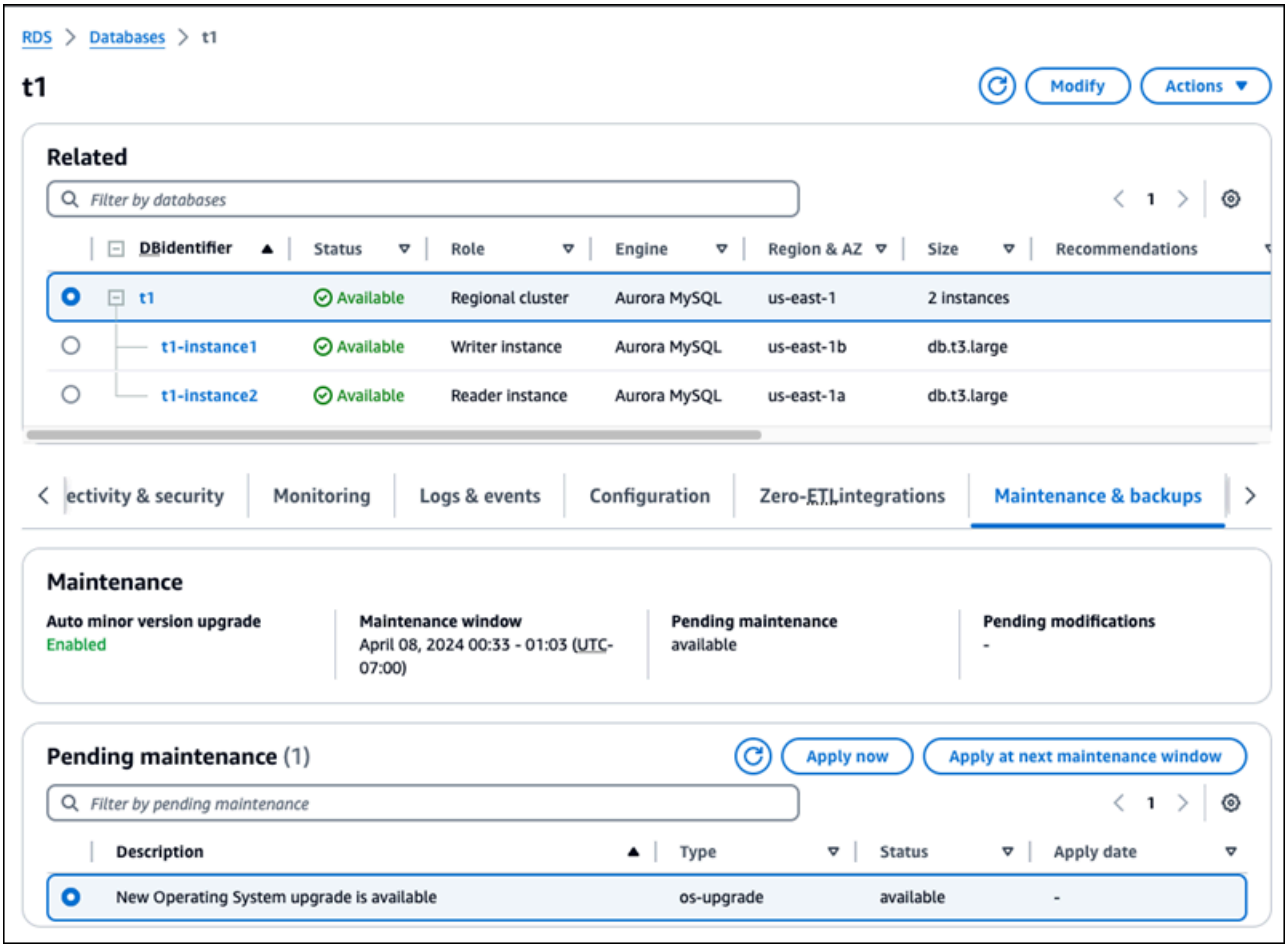
4. Suchen Sie im Abschnitt Ausstehende Wartungen das Betriebssystemupdate und überprüfen Sie den Wert Beschreibung.

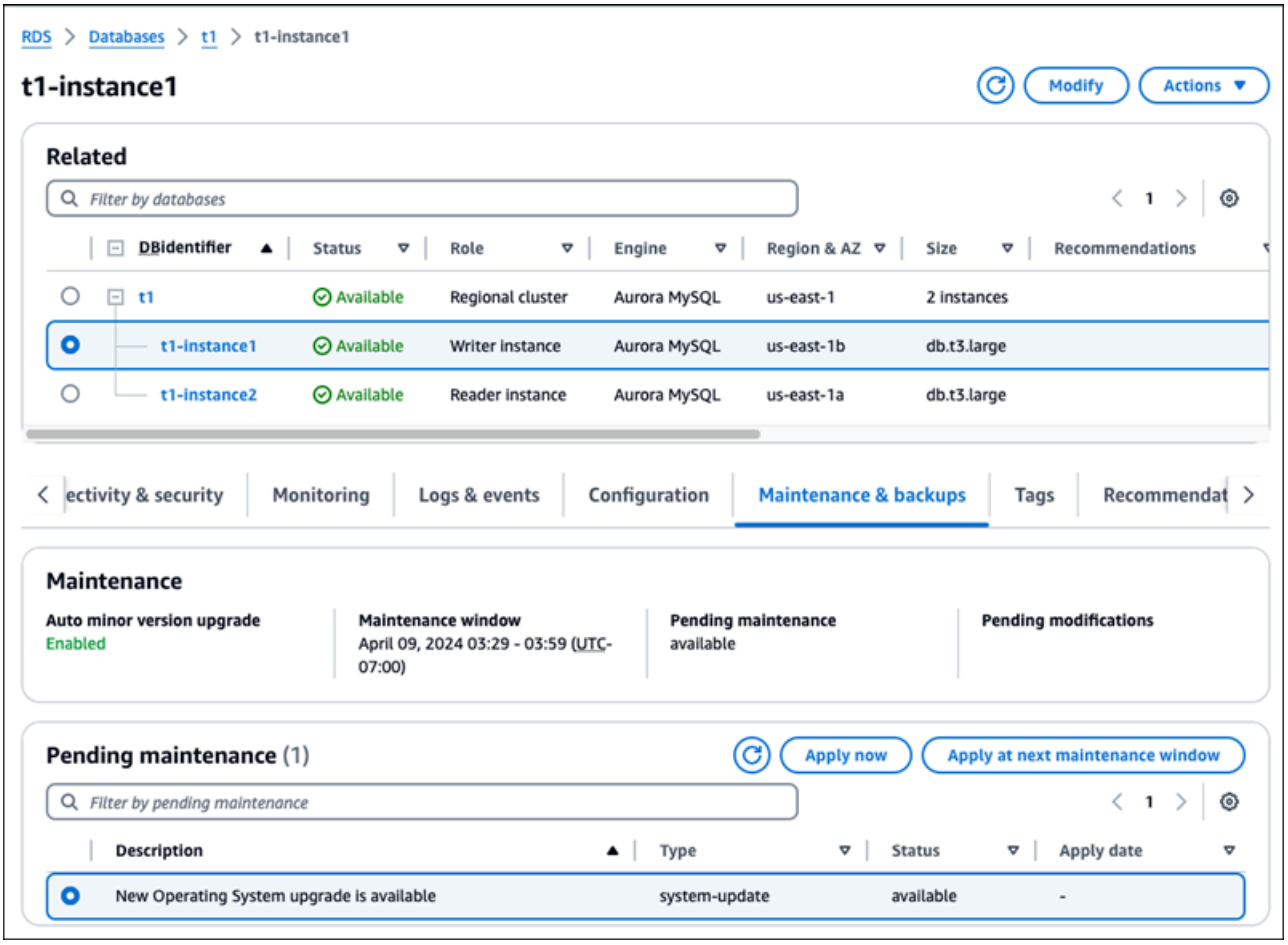
Die folgenden Abbildungen zeigen einen DB-Cluster mit einer Writer-DB-Instance, für die ein Betriebssystem-Patch verfügbar ist.

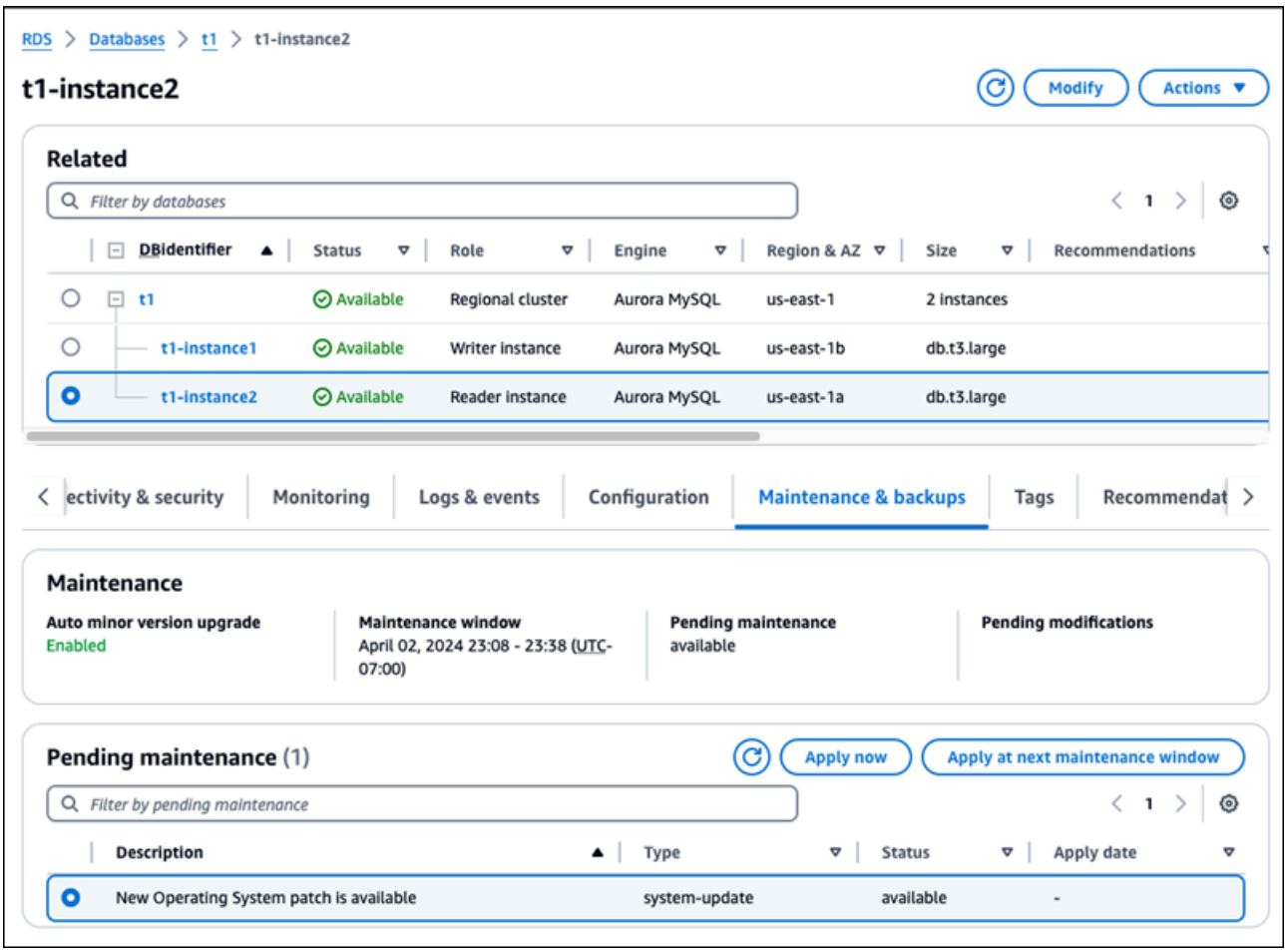




Die folgenden Abbildungen zeigen einen DB-Cluster mit einer Writer-DB-Instance und einer Reader-DB-Instance Für die Writer-Instance ist ein obligatorisches Betriebssystem-Upgrade verfügbar. Für die Reader-Instance ist ein Betriebssystem-Patch verfügbar.


RDS > Databases > t1
t1
Modify
Actions
Related
Filter by databases
1
DBidentifier
Status
Role
Engine
Region & AZ
Size
Recommendations
t1
Available
Regional cluster
Aurora MySQL
us-east-1
2 instances
t1-instance1
Available
Writer instance
Aurora MySQL
us-east-1b
db.t3.large
t1-instance2
Available
Reader instance
Aurora MySQL
us-east-1a
db.t3.large
ectivity & security
Monitoring
Logs & events
Configuration
Zero-ETL integrations
Maintenance & backups
Maintenance
Auto minor version upgrade
Enabled
Maintenance window
April 08, 2024 00:33 - 01:03 (UTC-07:00)
Pending maintenance
available
Pending modifications
-
Pending maintenance (1)
Apply now
Apply at next maintenance window
Filter by pending maintenance
1
Description
Type
Status
Apply date
New Operating System upgrade is available
os-upgrade
available
-


RDS > Databases > t1 > t1-instance1
t1-instance1
Modify
Actions
Related
Filter by databases
1
DBidentifier
Status
Role
Engine
Region & AZ
Size
Recommendations
t1
Available
Regional cluster
Aurora MySQL
us-east-1
2 instances
t1-instance1
Available
Writer instance
Aurora MySQL
us-east-1b
db.t3.large
t1-instance2
Available
Reader instance
Aurora MySQL
us-east-1a
db.t3.large
ectivity & security
Monitoring
Logs & events
Configuration
Maintenance & backups
Tags
Recommendat
Maintenance
Auto minor version upgrade
Enabled
Maintenance window
April 09, 2024 03:29 - 03:59 (UTC-07:00)
Pending maintenance
available
Pending modifications
Pending maintenance (1)
Apply now
Apply at next maintenance window
Filter by pending maintenance
1
Description
Type
Status
Apply date
New Operating System upgrade is available
system-update
available
-



AWS CLI

Verwenden Sie den describe-pending-maintenance-actionsBefehl AWS CLI, um Aktualisierungsinformationen von zu erhalten.

```
aws rds describe-pending-maintenance-actions
```

Die folgende Ausgabe zeigt ein Upgrade einer Betriebssystemdistribution für einen DB-Cluster und eine DB-Instance.

```
{
  "PendingMaintenanceActions": [
    {
      "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:cluster:t3",
      "PendingMaintenanceActionDetails": [
        {
          "Action": "os-upgrade",
          "Description": "New Operating System upgrade is available"
```

```
        }
      ]
    },
    {
      "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:t3-instance1",
      "PendingMaintenanceActionDetails": [
        {
          "Action": "system-update",
          "Description": "New Operating System upgrade is available"
        }
      ]
    },
  ]
}
```

Die folgende Ausgabe zeigt einen Betriebssystem-Patch für eine DB-Instance.

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb2",
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "Description": "New Operating System patch is available"
    }
  ]
}
```

Verfügbarkeit von Betriebssystemupdates

Betriebssystemupdates sind spezifisch für die DB-Engine-Version und die DB-Instance-Klasse. Daher erhalten oder erfordern DB-Instances Updates zu verschiedenen Zeiten. Wenn für Ihre DB-Instance basierend auf der Engine-Version und der Instance-Klasse ein Betriebssystemupdate erforderlich ist, wird das erforderliche Update in der Konsole angezeigt. Es kann auch angezeigt werden, indem describe-pending-maintenance-actions AWS CLI Sie den Befehl ausführen oder den DescribePendingMaintenanceActionsRDS-API-Vorgang aufrufen. Wenn für Ihre Instance ein Update verfügbar ist, können Sie Ihr Betriebssystem aktualisieren, indem Sie den Anleitungen unter Anwenden von Updates auf einen DB-Cluster folgen.

Verwendung der AWS Organizations Upgrade-Rollout-Richtlinie für automatische Upgrades kleinerer Versionen

Aurora unterstützt AWS Organizations Upgrade-Rollout-Richtlinien zur Verwaltung automatischer Upgrades kleinerer Versionen für mehrere Datenbankressourcen und AWS-Konten. Diese Richtlinie unterstützt Sie bei der Implementierung einer kontrollierten Upgrade-Strategie für Ihre Cluster durch:

So funktioniert die Upgrade-Rollout-Richtlinie

Wenn eine neue Engine-Nebenversion für ein automatisches Upgrade in Frage kommt, steuert die Richtlinie die Upgrade-Sequenz auf der Grundlage definierter Bestellungen:

- Ressourcen, die als [zuerst] gekennzeichnet sind (in der Regel Entwicklungsumgebungen), kommen während ihrer Wartungsfenster für Upgrades in Frage.
- Nach Ablauf einer bestimmten Backzeit kommen Ressourcen, die als [zweite] markiert sind, für sie in Frage.
- Nach einer weiteren festgelegten Backzeit kommen Ressourcen, die als [zuletzt] markiert sind (in der Regel Produktionsumgebungen), in Frage.
- Überwachung des Upgrade-Fortschritts mithilfe von AWS Health-Benachrichtigungen.

Sie können Ihre Upgrade-Bestellungen wie folgt definieren:

- Richtlinien auf Kontoebene — Gilt für alle berechtigten Ressourcen in bestimmten Konten.
- Ressourcen-Tags — Gilt für bestimmte Ressourcen, die auf Tags basieren.

> **Note**
>
> Ressourcen, die nicht mit einer Upgrade-Richtlinie konfiguriert oder von der Richtlinie ausgeschlossen sind, erhalten automatisch eine Upgrade-Reihenfolge von [Sekunde].

Voraussetzungen

- Sie AWS-Konto müssen Teil einer Organisation in Organizations sein, für die Upgrade-Rollout-Richtlinie aktiviert ist
- Aktivieren Sie automatische Upgrades für kleinere Versionen für Ihre Cluster

- Tags sind für die Upgrade-Rollout-Richtlinie nicht unbedingt erforderlich. Wenn Sie spezifische Upgrade-Bestellungen für verschiedene Umgebungen (z. B. Entwicklung, Test, Qualitätssicherung, Produktion) definieren möchten, können Sie Tags verwenden. Wenn Sie keine Tag-Einstellungen in Ihre Richtlinie aufnehmen, folgen alle Ressourcen, die dieser Richtlinie unterliegen, der Standard-Upgrade-Reihenfolge. Für Aurora-Ressourcen werden nur Tags auf Clusterebene für die Upgrade-Rollout-Richtlinie verwendet, auch wenn Sie Tags auf Instanzebene definiert haben.

Um Ihre Ressourcen zu taggen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie den Cluster aus, den Sie taggen möchten.
4. Wählen Sie „Aktionen“ und anschließend „Tags verwalten“.
5. Wählen Sie Add tag.
6. Gib deinen Tag-Schlüssel (zum Beispiel „Umgebung“) und den Wert (zum Beispiel „Entwicklung“) ein
7. Wählen Sie Tag hinzufügen und dann Speichern.

Sie können Tags auch hinzufügen, indem Sie AWS CLI:

```
aws rds add-tags-to-resource \
    --resource-name arn:aws:rds:region:account-number:cluster:cluster-name \
    --tags Key=Environment,Value=Development
```

Reihenfolge und Phasen des Upgrades

Die Upgrade-Rollout-Richtlinie unterstützt drei Upgrade-Bestellungen:

- [first] — Wird in der Regel für Entwicklungs- oder Testumgebungen verwendet
- [zweite] — Wird in der Regel für QA-Umgebungen verwendet. Standardreihenfolge für Ressourcen, wenn die Richtlinie nicht speziell konfiguriert ist
- [last] — In der Regel für Produktionsumgebungen reserviert

Wenn eine neue Minor-Engine-Version für ein automatisches Upgrade in Frage kommt:

- Ressourcen mit einem Upgrade-Auftrag [zuerst] kommen für Upgrades während der konfigurierten Wartungsfenster in Frage.
- Nach Ablauf einer bestimmten Wartezeit kommen Ressourcen mit Upgrade-Auftrag [zweiter] während ihrer Wartungsfenster für Upgrades in Frage.
- Nach einer weiteren festgelegten Backzeit kommen Ressourcen mit Upgrade-Bestellung [zuletzt] während ihrer Wartungsfenster für Upgrades in Frage.
- Die automatische Upgrade-Kampagne für kleinere Versionen endet, nachdem alle infrage kommenden Ressourcen mit Upgrade-Bestellungen [erste], [zweite] und [letzte] aktualisiert wurden oder wenn die Kampagne ihr geplantes Enddatum erreicht hat, je nachdem, was zuerst eintritt.

> **Note**
>
> Alle automatischen Upgrades der Nebenversionen werden während des für jeden Cluster konfigurierten Wartungsfensters durchgeführt, um mögliche Auswirkungen auf Ihre Anwendungen zu minimieren.

Beobachtbarkeit

AWS Health und Überwachung

Sie erhalten AWS Gesundheitsbenachrichtigungen:

- Vor dem Start einer automatischen Upgrade-Kampagne für kleinere Versionen
- Zwischen den einzelnen Phasen erfolgt der Übergang, um den Upgrade-Fortschritt besser verfolgen und überwachen zu können
- Fortschrittsaktualisierungen mit der Anzahl der Ressourcen, die in Ihrer Flotte aktualisiert wurden, in der AWS Health-Konsole

Amazon RDS-Ereignisbenachrichtigungen:

- Benachrichtigungen für Ressourcen, die für automatische Upgrades kleinerer Versionen aktiviert wurden, einschließlich:
  - Wenn Ihre Ressource aufgrund ihrer Upgrade-Bestellung ([erste], [zweite] oder [letzte]) für ein Upgrade in Frage kommt
  - Geplanter Upgrade-Zeitplan während des Wartungsfensters

  - Start- und Abschlussstatus des individuellen Datenbank-Upgrades
- Abonnieren Sie diese Ereignisse über Amazon EventBridge 0 für eine automatisierte Überwachung

Überlegungen

Einige Überlegungen, die Sie beachten sollten:

- Die Richtlinie gilt für alle future automatischen Upgrade-Kampagnen für kleinere Versionen, einschließlich Richtlinienänderungen, die während aktiver Kampagnen vorgenommen wurden.
- Wenn Sie an einer laufenden Upgrade-Kampagne teilnehmen, folgen Ihre Ressourcen der aktuellen Upgrade-Reihenfolge und warten nicht auf eine konfigurierte Richtlinie.
- Ressourcen, die nicht mit einer Upgrade-Richtlinie konfiguriert oder von der Richtlinie ausgeschlossen sind, erhalten automatisch eine Upgrade-Reihenfolge von [Sekunde].
- Die Richtlinie sieht Gültigkeitszeiträume zwischen den Upgrade-Phasen vor, bevor mit der nächsten Phase fortgefahren wird.
- Es dauert einige Zeit, bis Änderungen an der Richtlinie oder den Ressourcen-Tags wirksam werden, bevor die neue Upgrade-Reihenfolge angewendet wird.
- Die Richtlinie gilt nur für Aurora-Ressourcen, für die automatische Upgrades für kleinere Versionen aktiviert sind.
- Wenn Sie ein Problem in einer Umgebung entdecken, können Sie automatische Upgrades für Nebenversionen für nachfolgende Umgebungen deaktivieren oder den Validierungszeitraum nutzen, um Probleme zu lösen, bevor die Upgrades mit der nächsten Upgrade-Bestellung fortgeführt werden.

Weitere Informationen zum Taggen von RDS-Ressourcen finden Sie unterTaggen von Amazon-Aurora- und Amazon-RDS-Ressourcen. Ausführliche Anweisungen zur Einrichtung und Verwendung der Upgrade-Rollout-Richtlinie finden Sie AWS Organizations im AWS Organizations Benutzerhandbuch unter Erste Schritte mit.

Neustart eines Amazon Aurora DB-Clusters oder einer Amazon Aurora DB-Instance

Möglicherweise müssen Sie Ihren DB-Cluster oder einige Instances innerhalb des Clusters neu starten, normalerweise aus Wartungsgründen. Angenommen, Sie ändern die Parameter innerhalb einer Parametergruppe oder ordnen Ihrem Cluster eine andere Parametergruppe zu. In diesen Fällen müssen Sie den Cluster neu starten, damit die Änderungen wirksam werden. In ähnlicher Weise können Sie eine oder mehrere Reader-DB-Instances innerhalb des Clusters neu starten. Sie können die Neustartvorgänge für einzelne Instances anordnen, um Ausfallzeiten für den gesamten Cluster zu minimieren.

Die für den Neustart der einzelnen DB-Instances in Ihrem Cluster erforderliche Zeit hängt von der Datenbankaktivität zum Zeitpunkt des Neustarts ab. Dies hängt auch vom Wiederherstellungsprozess Ihrer spezifischen DB-Engine ab. Wenn dies praktisch ist, reduzieren Sie die Datenbankaktivität auf dieser bestimmten Instance, bevor Sie den Neustartvorgang starten. Dadurch kann die zum Neustart der Datenbank erforderliche Zeit verkürzt werden.

Sie können jede DB-Instance in Ihrem Cluster nur neu starten, wenn sie sich im verfügbaren Status befindet. Eine DB-Instance kann aus mehreren Gründen nicht verfügbar sein. Dazu gehören der Status der gestoppten Cluster, eine Änderung, die auf die Instance angewendet wird und eine Wartungsfensteraktion, sowie ein Versionsupgrade.

Beim Neustart einer DB-Instance wird der Datenbank-Engine-Prozess neu gestartet. Das Neustarten einer DB-Instance bewirkt einen vorübergehenden Nutzungsausfall, wobei der Status für diese Instance währenddessen auf Wird neu gestartet … gesetzt wird.

> **Note**
>
> Wenn auf der DB-Instance noch nicht die neuesten Änderungen der zugeordneten DB-Parametergruppe übernommen wurden, gibt die AWS-Managementkonsole für diese DB-Parametergruppe den Status pending-reboot (Neustart ausstehend) an. Die Parametergruppe pending-reboot führt während des nächsten Wartungsfensters nicht zu einem automatischen Neustart. Damit die neuesten Parameteränderungen für diese DB-Instance übernommen werden, starten Sie die DB-Instance manuell neu. Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Themen

- Neustarten einer DB-Instance in einem Aurora Cluster
- Neustarten eines Aurora-Clusters mit Leseverfügbarkeit
- Neustart eines Aurora-Clusters ohne Leseverfügbarkeit
- Überprüfung der Betriebszeit für Aurora Cluster und Instances
- Beispiele für Aurora Neustartvorgänge

Neustarten einer DB-Instance in einem Aurora Cluster

Dieses Verfahren ist der wichtigste Vorgang, den Sie beim Durchführen von Neustarts mit Aurora ausführen. Viele der Wartungsverfahren beinhalten das Neustarten einer oder mehrerer Aurora DB-Instances in einer bestimmten Reihenfolge.

Konsole

Neustarten einer DB-Instance

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die DB-Instance aus, die Sie neu starten möchten.
3. Wählen Sie unter Aktionen die Option Neustart aus.

   Die Seite DB-Instance neu starten wird angezeigt.
4. Wählen Sie Neustart aus, um Ihrer DB-Instance neu zu starten.

   Oder klicken Sie auf Abbrechen.

AWS CLI

Um eine DB-Instance mithilfe von AWS CLI neu zu starten, rufen Sie den Befehl `reboot-db-instance` auf.

Example

Linux, macOS oder Unix:

```
aws rds reboot-db-instance \
```

```
    --db-instance-identifier mydbinstance
```

Für Windows:

```
aws rds reboot-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS-API

Um eine DB-Instance mithilfe der Amazon-RDS-API neu zu starten, rufen Sie den Vorgang `RebootDBInstance` auf.

Neustarten eines Aurora-Clusters mit Leseverfügbarkeit

Mit der Funktion zur Leseverfügbarkeit können Sie die Writer-Instance Ihres Aurora-Clusters neu starten, ohne die Reader-Instances im primären oder sekundären DB-Cluster neu starten zu müssen. Dies kann dazu beitragen, die hohe Verfügbarkeit des Clusters für Lesevorgänge aufrechtzuerhalten, während Sie die Writer-Instance neu starten. Sie können die Reader-Instances später nach einem für Sie geeigneten Zeitplan neu starten. Beispielsweise können Sie für einen Produktions-Cluster die Reader-Instances nacheinander neu starten und erst beginnen, nachdem der Neustart der primären Instance abgeschlossen ist. Folgen Sie für jede DB-Instance, die Sie neu starten, dem Verfahren unter Neustarten einer DB-Instance in einem Aurora Cluster.

Die Funktion der Leseverfügbarkeit für primäre DB-Cluster ist in Aurora MySQL 2.10 und höher verfügbar. Die Leseverfügbarkeit für sekundäre DB-Cluster ist in Aurora-MySQL-Version 3.06 und höher verfügbar.

Diese Funktion steht standardmäßig für die folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- 15.2 und höhere 15-Versionen
- 14.7 und höhere 14-Versionen
- 13.10 und höhere 13-Versionen
- 12.14 und höhere 12-Versionen

Weitere Informationen zur Leseverfügbarkeitsfunktion in Aurora PostgreSQL finden Sie unter Verbesserung der Leseverfügbarkeit von Aurora Replicas.

Vor der Einführung dieser Funktion verursachte der Neustart der primären Instance gleichzeitig einen Neustart für jede Reader-Instance. Wenn auf Ihrem Aurora-Cluster eine ältere Version ausgeführt

wird, verwenden Sie stattdessen das Neustartverfahren in Neustart eines Aurora-Clusters ohne Leseverfügbarkeit.

> **Note**
>
> Die Änderung des Neustartverhaltens in Aurora-DB-Cluster mit Leseverfügbarkeit ist bei globalen Aurora-Datenbanken in Aurora-MySQL-Versionen niedriger als 3.06 anders. Wenn Sie die Writer-Instance für den primären Cluster in einer globalen Aurora-Datenbank neu starten, bleiben die Reader-Instances im primären Cluster verfügbar. Die DB-Instances in sekundären Clustern werden jedoch gleichzeitig neu gestartet.
> Eine eingeschränkte Version der verbesserten Leseverfügbarkeitsfunktion wird von globalen Aurora-Datenbanken für die Aurora-PostgreSQL-Versionen 12.16, 13.12, 14.9, 15.4 und höher unterstützt.

Sie starten den Cluster häufig neu, nachdem Sie Änderungen an Clusterparametergruppen vorgenommen haben. Sie nehmen Parameteränderungen vor, indem Sie die Verfahren in befolge Parametergruppen für Amazon Aurora. Angenommen, Sie starten die Writer-DB-Instance in einem Aurora-Cluster neu, um Änderungen an Cluster-Parametern anzuwenden. Einige oder alle Reader-DB-Instances verwenden möglicherweise weiterhin die alten Parametereinstellungen. Die verschiedenen Parametereinstellungen haben jedoch keinen Einfluss auf die Datenintegrität des Clusters. Alle Clusterparameter, die sich auf die Organisation von Datendateien auswirken, werden nur von der Writer-DB-Instance verwendet.

Beispielsweise können Sie in einem Aurora-MySQL-Cluster Cluster-Parameter wie `binlog_format` und `innodb_purge_threads` auf der Writer-Instance vor den Reader-Instances aktualisieren. Nur die Writer-Instance schreibt binäre Protokolle und löscht Datensätze für das Rückgängigmachen. Bei Parametern, die die Interpretation von SQL-Anweisungen oder Abfrageausgaben ändern, müssen Sie möglicherweise darauf achten, die Reader-Instances sofort neu zu starten. Sie tun dies, um unerwartetes Anwendungsverhalten bei Abfragen zu vermeiden. Angenommen, Sie ändern den `lower_case_table_names` Parameter und starten die Writer-Instance neu. In diesem Fall können die Reader-Instances möglicherweise erst auf eine neu erstellte Tabelle zugreifen, wenn sie alle neu gestartet wurden.

Eine Liste aller Aurora MySQL Clusterparameter finden Sie unter Parameter auf Cluster-Ebene.

Eine Liste aller Cluster-Parameter von Aurora PostgreSQL finden Sie unter Aurora-PostgreSQL-Parameter auf Cluster-Ebene.

> **Tip**
>
> Aurora MySQL startet möglicherweise noch einige der Reader-Instances zusammen mit der Writer-Instance neu, wenn Ihr Cluster einen Workload mit hohem Durchsatz verarbeitet. Die Verringerung der Anzahl der Neustarts gilt auch während des Failovervorgangs. Aurora MySQL startet nur die Writer-DB-Instance und das Failover-Ziel während eines Failovers neu. Andere Reader-DB-Instances im Cluster bleiben verfügbar, um Abfragen über Verbindungen zum Reader-Endpunkt fortzusetzen. So können Sie die Verfügbarkeit während eines Failovers verbessern, indem Sie mehr als eine Reader-DB-Instance in einem Cluster haben.

Neustart eines Aurora-Clusters ohne Leseverfügbarkeit

Ohne die Funktion zur Leseverfügbarkeit starten Sie einen ganzen Aurora-DB-Cluster neu, indem Sie die Writer-DB-Instance dieses Clusters neu starten. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Neustarten einer DB-Instance in einem Aurora Cluster.

Ein Neustart der Writer-DB-Instance initiiert auch einen Neustart für jede Reader-DB-Instance im Cluster. Auf diese Weise werden alle clusterweiten Parameteränderungen gleichzeitig auf alle DB-Instances angewendet. Der Neustart aller DB-Instances führt jedoch zu einem kurzen Ausfall für den Cluster. Die Reader DB-Instances bleiben nicht verfügbar, bis die Writer DB-Instance den Neustart beendet hat und verfügbar wird.

Dieses Neustartverhalten gilt für alle DB-Cluster, die in Aurora MySQL Version 2.09 und niedriger erstellt wurden.

Für Aurora PostgreSQL gilt dieses Verhalten für die folgenden Versionen:

- 14.6 und niedrigere 14-Versionen
- 13.9 und niedrigere 13-Versionen
- 12.13 und niedrigere 12-Versionen
- Alle PostgreSQL-11-Versionen

In der RDS-Konsole hat die Writer-DB-Instance den Wert Writer in der Spalte Rolle auf der Seite Datenbanken. In der RDS CLI enthält die Ausgabe des `describe-db-clusters`-Befehls einen Abschnitt `DBClusterMembers`. Das `DBClusterMembers`-Element, das die Writer-DB-Instance repräsentiert, hat einen Wert von `true` für das `IsClusterWriter` Feld.

Important

Mit der Funktion zur Leseverfügbarkeit ist das Neustartverhalten in Aurora MySQL anders als in Aurora PostgreSQL: Die Reader-DB-Instances bleiben normalerweise verfügbar, während Sie die Writer-Instance neu starten. Dann können Sie die Reader-Instances zu einem geeigneten Zeitpunkt neu starten. Sie können die Reader-Instances nach einem gestaffelten Zeitplan neu starten, wenn einige Reader-Instances immer verfügbar sein sollen. Weitere Informationen finden Sie unter Neustarten eines Aurora-Clusters mit Leseverfügbarkeit.

Überprüfung der Betriebszeit für Aurora Cluster und Instances

Sie können die Dauer seit dem letzten Neustart für jede DB-Instance in Ihrem Aurora Cluster überprüfen und überwachen. Die CloudWatch `EngineUptime` Amazon-Metrik gibt die Anzahl der Sekunden seit dem letzten Start einer DB-Instance an. Sie können diese Metrik zu einem bestimmten Zeitpunkt untersuchen, um die Verfügbarkeit der DB-Instance zu ermitteln. Sie können diese Metrik auch im Laufe der Zeit überwachen, um zu erkennen, wann die Instance neu gestartet wird.

Sie können die `EngineUptime` Metrik auch auf Clusterebene untersuchen. Die `Minimum`- und `Maximum`- Dimensionen geben die kleinsten und größten Verfügbarkeitswerte für alle DB-Instances im Cluster an. Um zu überprüfen, wann eine Reader-Instance in einem Cluster zuletzt neu gestartet oder aus einem anderen Grund neu gestartet wurde, überwachen Sie die Metrik auf Clusterebene mithilfe der `Minimum`-Dimension. Um zu überprüfen, welche Instance im Cluster am längsten ohne Neustart gelaufen ist, überwachen Sie die Metrik auf Clusterebene mithilfe der `Maximum`-Dimension. Sie möchten beispielsweise bestätigen, dass alle DB-Instances im Cluster nach einer Konfigurationsänderung neu gestartet wurden.

Tip

Für eine langfristige Überwachung empfehlen wir, die `EngineUptime`-Metrik für einzelne Instances statt auf Clusterebene zu überwachen. Die `EngineUptime`-Metrik auf Clusterebene wird auf Null gesetzt, wenn dem Cluster eine neue DB-Instance hinzugefügt wird. Solche Clusteränderungen können im Rahmen von Wartungs- und Skalierungsvorgängen erfolgen, wie sie beispielsweise von Amazon EC2 Auto Scaling ausgeführt werden.

Die folgenden CLI-Beispiele zeigen, wie die `EngineUptime`-Metrik für die Writer- und Reader-Instances in einem Cluster untersucht wird. Die Beispiele verwenden einen Cluster namens `tpch100g`. Dieser Cluster hat eine Writer-DB-Instance `instance-1234`. Er hat auch zwei Reader-DB-Instances, `instance-7448` und `instance-6305`.

Zuerst startet der `reboot-db-instance`-Befehl eine der Reader-Instances neu. Der `wait`-Befehl wartet, bis die Instance mit dem Neustart abgeschlossen ist.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

Der CloudWatch `get-metric-statistics` Befehl untersucht die `EngineUptime` Metrik der letzten fünf Minuten in Intervallen von einer Minute. Die Betriebszeit für die `instance-6305` Instance wird auf Null zurückgesetzt und beginnt wieder nach oben zu zählen. In diesem AWS CLI Beispiel für Linux wird die `$()` Variablenersetzung verwendet, um die entsprechenden Zeitstempel in die CLI-Befehle einzufügen. Es verwendet auch den `sort` Linux-Befehl, um die Ausgabe bis zum Zeitpunkt der Erfassung der Metrik zu ordnen. Dieser Zeitstempelwert ist das dritte Feld in jeder Ausgabezeile.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

Die minimale Betriebszeit für den Cluster wird auf Null zurückgesetzt, da eine der Instances im Cluster neu gestartet wurde. Die maximale Betriebszeit für den Cluster wird nicht zurückgesetzt, da mindestens eine der DB-Instances im Cluster verfügbar blieb.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Minimum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

Dann startet ein anderer `reboot-db-instance` Befehl die Writer-Instance des Clusters neu. Ein anderer `wait` Befehl wird pausiert, bis die Writer-Instance mit dem Neustart abgeschlossen ist.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

Jetzt zeigt die `EngineUptime` Metrik für die Writer-Instance, dass die Instance `instance-1234` kürzlich neu gestartet wurde. Die Reader-Instance `instance-6305` wurde zusammen mit der Writer-Instance ebenfalls automatisch neu gestartet. Auf diesem Cluster wird Aurora MySQL 2.09 ausgeführt, wodurch die Reader-Instances nicht weiter ausgeführt werden, wenn die Writer-Instance neu gestartet wird.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
```

```
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

Beispiele für Aurora Neustartvorgänge

Die folgenden Aurora MySQL Beispiele zeigen verschiedene Kombinationen von Neustartvorgängen für Reader- und Writer-DB-Instances in einem Aurora DB-Cluster. Nach jedem Neustart demonstrieren SQL-Abfragen die Verfügbarkeit der Instances im Cluster.

Themen

- Finden der Writer- und Reader-Instances für einen Aurora-Cluster
- Neustart einer einzelnen Reader-Instance
- Neustart der Writer-Instance
- Den Writer und die Reader unabhängig neu starten
- Anwenden einer Clusterparameteränderung auf einen Cluster der Aurora MySQL Version 2.10

Finden der Writer- und Reader-Instances für einen Aurora-Cluster

In einem Aurora MySQL Cluster mit mehreren DB-Instances ist es wichtig zu wissen, welche der Writer und welche die Reader sind. Die Writer- und Reader-Instances können auch Rollen

wechseln, wenn ein Failover-Vorgang stattfindet. Daher ist es am besten, eine Überprüfung wie die folgende durchzuführen, bevor Sie eine Operation ausführen, die eine Writer- oder Reader-Instance erfordert. In diesem Fall identifizieren die `False`-Werte für `IsClusterWriter` die Reader-Instances, `instance-6305` und `instance-7448`. Der `True` Wert identifiziert die Writer-Instance `instance-1234`.

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:     tpch100g
Instance:    instance-6305    False
Instance:    instance-7448    False
Instance:    instance-1234    True
```

Bevor wir mit den Beispielen für den Neustart beginnen, hat die Writer-Instance eine Betriebszeit von ungefähr einer Woche. Die SQL-Abfrage in diesem Beispiel zeigt eine MySQL-spezifische Möglichkeit, die Betriebszeit zu überprüfen. Sie können diese Technik in einer Datenbankanwendung verwenden. Für eine andere Technik, die AWS CLI verwendet und für beide Aurora-Engines funktioniert, siehe Überprüfung der Betriebszeit für Aurora Cluster und Instances.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+----------------------------+---------+
```

Neustart einer einzelnen Reader-Instance

In diesem Beispiel wird eine der Reader-DB-Instances neu gestartet. Vielleicht wurde diese Instance von einer riesigen Abfrage oder vielen gleichzeitigen Verbindungen überlastet. Oder vielleicht fiel sie wegen eines Netzwerkproblems hinter die Writer-Instance zurück. Nach dem Start des Neustartvorgangs verwendet das Beispiel einen `wait` Befehl, um anzuhalten, bis die Instance verfügbar ist. Zu diesem Zeitpunkt hat die Instance eine Betriebszeit von einigen Minuten.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:35:02.000000 | 00h 03m |
+----------------------------+---------+
```

Der Neustart der Reader-Instance hatte keinen Einfluss auf die Verfügbarkeit der Writer-Instance. Sie hat immer noch eine Betriebszeit von etwa einer Woche.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+----------+
| Last Startup               | Uptime   |
+----------------------------+----------+
| 2021-03-08 17:49:06.000000 | 174h 49m |
+----------------------------+----------+
```

Neustart der Writer-Instance

In diesem Beispiel wird die Writer-Instance neu gestartet. Auf diesem Cluster wird Aurora MySQL Version 2.09 ausgeführt. Da die Aurora MySQL Version niedriger als 2.10 ist, werden beim Neustart der Writer-Instance auch alle Reader-Instances im Cluster neu gestartet.

Ein `wait` Befehl wird angehalten, bis der Neustart abgeschlossen ist. Jetzt wird die Betriebszeit für diese Instance auf Null zurückgesetzt. Es ist möglich, dass ein Neustartvorgang für Writer- und Reader-DB-Instances erheblich unterschiedliche Zeiten in Anspruch nehmen kann. Die Writer- und Reader-DB-Instances führen je nach ihren Rollen verschiedene Arten von Bereinigungsvorgängen aus.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+----------------------------+---------+
```

Nach dem Neustart für die Writer-DB-Instance haben beide Reader-DB-Instances auch ihre Betriebszeit zurückgesetzt. Durch den Neustart der Writer-Instance wurden die Reader-Instances ebenfalls neu gestartet. Dieses Verhalten gilt für Aurora PostgreSQL Cluster und Aurora MySQL Cluster vor Version 2.10.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:35.000000 | 00h 00m |
+----------------------------+---------+
```

```
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+----------------------------+---------+
```

Den Writer und die Reader unabhängig neu starten

Diese nächsten Beispiele zeigen einen Cluster, der Aurora MySQL Version 2.10 ausführt. In dieser Aurora MySQL Version und höher können Sie die Writer-Instance neu starten, ohne Neustarts für alle Reader-Instances zu verursachen. Auf diese Weise kommt es bei Ihren abfrachintensiven Anwendungen zu keinem Ausfall, wenn Sie die Writer-Instance neu starten. Sie können die Reader-Instances später neu starten. Sie können diese Neustarts zu einem Zeitpunkt mit geringem Abfrageverkehr durchführen. Sie können die Reader-Instances auch nacheinander neu starten. Auf diese Weise steht immer mindestens eine Reader-Instance für den Abfrageverkehr Ihrer Anwendung zur Verfügung.

Im folgenden Beispiel wird ein Cluster namens `cluster-2393` verwendet, auf dem Aurora MySQL Version `5.7.mysql_aurora.2.10.0` ausgeführt wird. Dieser Cluster hat eine Writer-Instance namens `instance-9404` und drei Reader-Instances namens `instance-6772`, `instance-2470` und `instance-5138`.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:        cluster-2393
Instance:       instance-5138        False
Instance:       instance-2470        False
Instance:       instance-6772        False
Instance:       instance-9404        True
```

Die Überprüfung des `uptime` Werts jeder Datenbankinstance mit dem Befehl `mysql` zeigt, dass jede ungefähr die gleiche Betriebszeit hat. Hier ist zum Beispiel die Betriebszeit für `instance-5138`.

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
```

```
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Uptime        | 3866  |
+---------------+-------+
```

Durch die Verwendung von CloudWatch können wir die entsprechenden Betriebszeitinformationen abrufen, ohne sich tatsächlich bei den Instances anzumelden. Auf diese Weise kann ein Administrator die Datenbank überwachen, aber keine Tabellendaten anzeigen oder ändern. In diesem Fall geben wir einen Zeitraum von fünf Minuten an und überprüfen den Betriebszeitwert jede Minute. Die zunehmenden Betriebszeitwerte zeigen, dass die Instances in diesem Zeitraum nicht neu gestartet wurden.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds
```

Jetzt starten wir eine der Reader-Instances neu, `instance-5138`. Wir warten darauf, dass die Instance nach dem Neustart wieder verfügbar ist. Die Überwachung der Betriebszeit über einen Zeitraum von fünf Minuten zeigt, dass die Betriebszeit während dieser Zeit auf Null zurückgesetzt wurde. Der letzte Betriebszeitwert wurde fünf Sekunden nach Abschluss des Neustarts gemessen.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS  5.0 2021-03-17T23:50:00+00:00 Seconds
```

Als Nächstes führen wir einen Neustart für die Writer-Instance durch, `instance-9404`. Wir vergleichen die Betriebszeit-Werte für die Writer-Instance und eine der Reader-Instances. Auf diese Weise können wir sehen, dass der Neustart des Writers für die Reader keinen Neustart verursacht hat. In Versionen vor Aurora MySQL 2.10 würden die Verfügbarkeitswerte für alle Reader gleichzeitig mit dem Writer zurückgesetzt.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
  "DBInstanceIdentifier": "instance-9404",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
```

```
DATAPOINTS  37.0  2021-03-18T00:01:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds
```

Um sicherzustellen, dass alle Reader-Instances dieselben Änderungen an den Konfigurationsparametern haben wie die Writer-Instance, starten Sie alle Reader-Instances nach dem Writer neu. Dieses Beispiel startet alle Reader neu und wartet dann, bis alle verfügbar sind, bevor Sie fortfahren.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

Jetzt können wir sehen, dass die Writer-DB-Instance die höchste Betriebszeit hat. Der Betriebszeitwert dieser Instance stieg während des gesamten Überwachungszeitraums stetig an. Die Reader DB-Instances wurden alle nach dem Reader neu gestartet. Wir können den Punkt innerhalb des Überwachungszeitraums sehen, an dem jeder Reader neu gestartet wurde und seine Betriebszeit auf Null zurückgesetzt wurde.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS  697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-2470 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS  35.0  2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS  49.0  2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds
```

Anwenden einer Clusterparameteränderung auf einen Cluster der Aurora MySQL Version 2.10

Das folgende Beispiel veranschaulicht, wie Sie eine Parameteränderung auf alle DB-Instances in Ihrem Aurora MySQL 2.10-Cluster anwenden. Mit dieser Aurora MySQL Version starten Sie die Writer-Instance und alle Reader-Instances unabhängig neu.

Das Beispiel verwendet den MySQL-Konfigurationsparameter `lower_case_table_names` zur Veranschaulichung. Wenn sich diese Parametereinstellung zwischen den Writer- und Reader-DB-Instances unterscheidet, kann eine Abfrage möglicherweise nicht auf eine Tabelle zugreifen, die mit einem Namen in Großbuchstaben oder gemischten Buchstaben deklariert wurde. Oder wenn sich zwei Tabellennamen nur in Groß- und Kleinbuchstaben unterscheiden, kann eine Abfrage auf die falsche Tabelle zugreifen.

Dieses Beispiel zeigt, wie Sie die Writer- und Reader-Instances im Cluster ermitteln können, indem Sie das `IsClusterWriter` Attribut jeder Instance untersuchen. Der Cluster heißt `cluster-2393`. Der Cluster hat eine Writer-Instance namens `instance-9404`. Die Reader-Instances im Cluster heißen `instance-5138` und `instance-2470`.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
cluster-2393
instance-5138        False
instance-2470        False
instance-9404        True
```

Um die Auswirkungen der Änderung des `lower_case_table_names` Parameters zu demonstrieren, richten wir zwei DB-Cluster-Parametergruppen ein. Für die `lower-case-table-names-0` Parametergruppe ist dieser Parameter auf 0 festgelegt. Die `lower-case-table-names-1` Parametergruppe hat diese Parametergruppe auf 1 festgelegt.

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
```

```
        "Description": "lower-case-table-names-0"
    }
}

$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-0 \
  --parameters
 ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-1 \
    --parameters
 ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
}
```

Der Standardwert von `lower_case_table_names` ist 0. Bei dieser Parametereinstellung unterscheidet sich die Tabelle `foo` von der Tabelle `F00`. In diesem Beispiel wird überprüft, dass der Parameter immer noch seine Standardeinstellung hat. Dann erstellt das Beispiel drei Tabellen, die sich nur in Groß- und Kleinbuchstaben in ihren Namen unterscheiden.

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+--------------------------+
```

```
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

Als Nächstes verknüpfen wir die DB-Parametergruppe mit dem Cluster, um den `lower_case_table_names` Parameter auf 1 festzulegen. Diese Änderung wird erst wirksam, nachdem jede DB-Instance neu gestartet wurde.

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
```

```
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

Der erste Neustart, den wir machen, ist für die Writer-DB-Instance. Dann warten wir, bis die Instance wieder verfügbar wird. An diesem Punkt stellen wir eine Verbindung zum Writer-Endpunkt her und stellen sicher, dass die Writer-Instance den geänderten Parameterwert hat. Der `SHOW TABLES` Befehl bestätigt, dass die Datenbank die drei verschiedenen Tabellen enthält. Abfragen, die sich auf Tabellen mit den Namen `foo`, `Foo` oder `F00` beziehen, greifen jedoch alle auf die Tabelle zu, deren Name nur aus Kleinbuchstaben besteht, `foo` .

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

Abfragen, die den Cluster-Endpunkt verwenden, zeigen nun die Auswirkungen der Parameteränderung. Unabhängig davon, ob der Tabellenname in der Abfrage aus Groß-, Kleinbuchstaben- oder gemischten Buchstaben besteht, greift die SQL-Anweisung auf die Tabelle zu, deren Name ganz aus Kleinbuchstaben besteht.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+

mysql> use lctn;
mysql> show tables;
+----------------+
| Tables_in_lctn |
+----------------+
| FOO            |
| Foo            |
| foo            |
+----------------+

mysql> select * from foo;
+--------------------------+
| s                        |
```

```
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

Das nächste Beispiel zeigt die gleichen Abfragen wie das vorherige. In diesem Fall verwenden die Abfragen den Reader-Endpunkt und werden auf einer der Reader DB-Instances ausgeführt. Diese Instances wurden noch nicht neu gestartet. Somit haben sie immer noch die ursprüngliche Einstellung für den `lower_case_table_names` Parameter. Das bedeutet, dass Abfragen auf jede der `foo`-, `Foo`- und `F00`-Tabellen zugreifen können.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                         |
```

```
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+---------------------------+
| s                         |
+---------------------------+
| Uppercase table name FOO |
+---------------------------+
```

Als Nächstes starten wir eine der Reader-Instances neu und warten darauf, dass sie wieder verfügbar ist.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-2470
```

Während eine Abfrage mit dem Instance-Endpunkt für `instance-2470` verbunden ist, zeigt eine Abfrage, dass der neue Parameter in Kraft ist.

```
mysql> select @@lower_case_table_names;
+---------------------------+
| @@lower_case_table_names |
+---------------------------+
|                         1 |
+---------------------------+
```

Zu diesem Zeitpunkt werden die beiden Reader-Instances im Cluster mit unterschiedlichen `lower_case_table_names` Einstellungen ausgeführt. Daher verwendet jede Verbindung zum Reader-Endpunkt des Clusters einen Wert für diese Einstellung, der unvorhersehbar ist. Es ist wichtig, die andere Reader-Instance sofort neu zu starten, damit beide konsistente Einstellungen haben.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
```

```
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

Das folgende Beispiel bestätigt, dass alle Reader-Instances dieselbe Einstellung für den `lower_case_table_names` Parameter haben. Die Befehle überprüfen den `lower_case_table_names` Einstellungswert für jede Reader-Instance. Dann zeigt derselbe Befehl, der den Reader-Endpunkt verwendet, dass jede Verbindung zum Reader-Endpunkt eine der Reader-Instances verwendet, welche jedoch nicht vorhersehbar ist.

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-2470            |                        1 |
+--------------------------+--------------------------+

# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
```

```
+--------------------------+--------------------------+
| instance-9404            |                        1 |
+--------------------------+--------------------------+
```

Wenn die Parameteränderung überall angewendet wird, können wir die Auswirkungen der Einstellung von sehe `lower_case_table_names=1`. Unabhängig davon, ob die Tabelle als `foo` , `Foo` oder `F00` bezeichnet wird, wandelt die Abfrage den Namen in `foo` um und greift jeweils auf dieselbe Tabelle zu.

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

Ausführen eines Failovers eines DB-Clusters von Amazon Aurora

Sie können ein manuelles Failover eines Aurora-DB-Clusters durchführen, wenn Sie beispielsweise eine bereitgestellte Writer-DB-Instance durch eine Aurora Serverless v2-Writer-Instance ersetzen möchten.

Aurora führt ein Failover zu einer neuen primären DB-Instance auf eine von zwei Arten durch:

- Hochstufen einer bestehenden Reader-DB-Instance zur neuen primären Instance
- Erstellen einer neuen primären Instance

Wenn ein DB-Cluster über ein oder mehrere Reader-Instances verfügt, wird ein Reader während eines Ausfallereignisses zur primären Instance hochgestuft. Wir empfehlen Ihnen, mindestens eine Reader-Instance in mindestens zwei verschiedenen Availability Zones zu erstellen, um die Verfügbarkeit Ihres DB-Clusters zu erhöhen. Weitere Informationen zum Failover-Mechanismus finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.

Für ein manuelles Failover können Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API verwenden.

Konsole

Failover eines DB-Clusters

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken und dann eine DB-Instance in dem DB-Cluster aus, für den ein Failover durchgeführt werden soll.
3. Wählen Sie für Aktionen die Option Failover aus.

   Die Bestätigungsseite wird angezeigt.
4. Klicken Sie auf Failover.

   Auf der Seite Datenbanken wird angezeigt, dass der DB-Cluster-Status Failover wird ausgeführt lautet. Der Status lautet wieder Verfügbar, wenn das Failover abgeschlossen ist, und die Rollen für die neuen und ehemals primären DB-Instances werden angezeigt.

AWS CLI

Für ein Failover eines DB-Clusters mit der AWS CLI rufen Sie den Befehl failover-db-cluster auf. Geben Sie die folgenden Parameter an:

- `--db-cluster-identifier` – Dies ist der DB-Cluster, für den ein Failover durchgeführt wird.
- `--target-db-instance-identifier` – Dies ist der Name der DB-Instance, die zur primären Instance hochgestuft werden soll.

Example

Für Linux, macOS oder Unix:

```
aws rds failover-db-cluster \
    --db-cluster-identifier mydbcluster \
    --target-db-instance-identifier mydbcluster-instance-2
```

Für Windows:

```
aws rds failover-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --target-db-instance-identifier mydbcluster-instance-2
```

RDS-API

Um einen DB-Cluster mit der Amazon-RDS-API zu ändern, rufen Sie die Operation FailoverDBCluster auf. Geben Sie die folgenden Parameter an:

- DBClusterIdentifier
- TargetDBInstanceIdentifier

Löschen von Aurora-DB-Clustern und -DB-Instances

Sie können einen Aurora-DB-Cluster löschen, wenn Sie ihn nicht mehr benötigen. Durch Löschen des Clusters wird das Cluster-Volume entfernt, das all Ihre Daten enthält. Bevor Sie den Cluster löschen, können Sie einen Snapshot Ihrer Daten speichern. Sie können den Snapshot später wiederherstellen, um einen neuen Cluster erstellen, der dieselben Daten enthält.

Sie können auch DB-Instances aus einem Cluster löschen, während der Cluster selbst und die darin enthaltenen Daten erhalten bleiben. Durch Löschen von DB-Instances können Sie die Kosten senken, wenn der Cluster nicht ausgelastet ist und Sie nicht die Rechenkapazität mehrerer DB-Instances benötigen.

Themen

- Löschen eines Aurora-Dönnen einen Cluster nicht löschen,B-Clusters
- Löschschutz für Aurora-DB-Cluster
- Löschen eines angehaltenen Aurora-Clusters
- Löschen von Aurora MySQL-Clustern, die Read Replicas sind
- Der letzte Snapshot beim Löschen eines Clusters
- Löschen einer DB-Instance aus einem Aurora-DB-Cluster

Löschen eines Aurora-Dönnen einen Cluster nicht löschen,B-Clusters

Aurora bietet keine Methode zum Löschen eines DB-Clusters in einem Schritt. Dadurch soll bewusst verhindert werden, dass Sie versehentlich Daten verlieren oder Ihre Anwendung offline schalten. Aurora-Anwendungen sind in der Regel geschäftskritisch und erfordern hohe Verfügbarkeit. Deshalb lässt sich mit Aurora die Kapazität des Clusters einfach durch Hinzufügen und Entfernen von DB-Instances aufwärts und abwärts skalieren. Um den Cluster selbst zu entfernen, müssen Sie eine separate Auswahl treffen.

Verwenden Sie das folgende allgemeine Verfahren, um alle DB-Instances aus einem Cluster zu entfernen und dann den Cluster selbst zu löschen.

1. Löschen Sie alle Reader-Instances im Cluster. Verwenden Sie das Verfahren in Löschen einer DB-Instance aus einem Aurora-DB-Cluster.

   Wenn der Cluster Reader-Instances enthält, reduziert das Löschen einer der Instances nur die Rechenkapazität des Clusters. Indem die Reader-Instances zuerst gelöscht werden, wird

sichergestellt, dass der Cluster während des gesamten Verfahrens verfügbar bleibt und keine unnötigen Failover-Vorgänge durchführt.

2. Löschen Sie die Writer-Instance aus dem Cluster. Verwenden Sie erneut das Verfahren in Löschen einer DB-Instance aus einem Aurora-DB-Cluster.

   Wenn Sie die DB-Instances löschen, bleiben der Cluster und das zugehörige Cluster-Volume auch nach dem Löschen aller DB-Instances bestehen.

3. Löschen Sie den DB-Cluster.

   - AWS-Managementkonsole – Wählen Sie den Cluster aus und wählen Sie dann Löschen im Menü Aktionen aus. Sie können die folgenden Optionen auswählen, um die Daten aus dem Cluster beizubehalten, falls Sie sie später benötigen:
     - Erstellen Sie einen finalen Snapshot des Cluster-Volumes. Standardmäßig wird ein finaler Snapshot erstellt.
     - Sie müssen automatisierte Sicherungen aufbewahren. Die Standardeinstellung ist, automatisierte Backups nicht beizubehalten.

     > **Note**
     >
     > Automatisierte Backups für DB-Cluster von Aurora Serverless v1 werden nicht aufbewahrt.

     Aurora erfordert außerdem, dass Sie bestätigen, dass Sie den Cluster löschen möchten.

   - CLI und API – Rufen Sie den CLI-Befehl `delete-db-cluster` oder die API-Operation `DeleteDBCluster` auf. Sie können die folgenden Optionen auswählen, um die Daten aus dem Cluster beizubehalten, falls Sie sie später benötigen:
     - Erstellen Sie einen finalen Snapshot des Cluster-Volumes.
     - Sie müssen automatisierte Sicherungen aufbewahren.

     > **Note**
     >
     > Automatisierte Backups für DB-Cluster von Aurora Serverless v1 werden nicht aufbewahrt.

Themen

- Erstellen eines leeren Aurora-Clusters
- Löschen eines Aurora-Clusters mit einer einzigen DB-Instance
- Löschen eines Aurora-Clusters mit mehreren DB-Instances

Erstellen eines leeren Aurora-Clusters

Sie können einen leeren DB-Cluster mit der AWS-Managementkonsole AWS CLI, oder Amazon RDS-API löschen.

> **Tip**
>
> Sie können einen Cluster ohne DB-Instances behalten, um Ihre Daten zu behalten, ohne dass CPU-Gebühren für den Cluster anfallen. Sie können den Cluster schnell wieder verwenden, indem Sie eine oder mehrere neue DB-Instances für den Cluster erstellen. Sie können neue DB-Instances jedoch nur mithilfe der AWS CLI oder der RDS-API hinzufügen. Sie können keine neuen DB-Instances über die Amazon-RDS-Konsole hinzufügen. Sie können auch Aurora-spezifische administrative Vorgänge auf dem Cluster durchführen, während ihm keine DB-Instances zugeordnet sind. Sie können bloß nicht auf die Daten zugreifen oder Vorgänge durchführen, die eine Verbindung mit einer DB-Instance erfordern.

Konsole

So löschen Sie einen DB-Cluster

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken und dann den DB-Cluster aus, den Sie löschen möchten.
3. Klicken Sie bei Actions auf Delete.
4. Wählen Sie Abschließenden Snapshot erstellen? aus, um einen endgültigen DB-Snapshot für den DB-Cluster zu erstellen. Dies ist die Standardeinstellung.
5. Wenn Sie einen endgültigen Snapshot erstellen möchten, geben Sie den Final snapshot name (Name des endgültigen Snapshots) ein.
6. Wählen Sie Retain automated backups (Automatisierte Sicherungen aufbewahren), um automatisierte Sicherungen aufzubewahren. Dies ist nicht die Standardeinstellung.

7. Geben Sie **`delete me`** in das Feld ein.
8. Wählen Sie Löschen aus.

CLI

Um einen leeren Aurora-DB-Cluster mit dem zu löschen AWS CLI, rufen Sie den delete-db-clusterBefehl auf.

Angenommen, der leere Cluster `deleteme-zero-instances` wurde nur für Entwicklungs- und Testzwecke verwendet und enthält keine wichtigen Daten. In diesem Fall müssen Sie beim Löschen des Clusters keinen Snapshot des Cluster-Volumes aufbewahren. Das folgende Beispiel zeigt, dass ein Cluster keine DB-Instances enthält, und veranschaulicht dann, wie der leere Cluster gelöscht wird, ohne einen finalen Snapshot zu erstellen oder automatisierte Backups beizubehalten.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output
 text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

RDS-API

Um einen leeren Aurora-DB-Cluster mithilfe der Amazon RDS-API zu löschen, rufen Sie den DBCluster Vorgang Löschen auf.

Löschen eines Aurora-Clusters mit einer einzigen DB-Instance

Sie können die letzte DB-Instance auch dann löschen, wenn der DB-Cluster einen aktivierten Löschschutz hat. In diesen Fall bleibt der DB-Cluster selbst bestehen und die Daten werden beibehalten. Sie können wieder auf die Daten zugreifen, indem Sie dem Cluster eine neue DB-Instance hinzufügen.

Das folgende Beispiel zeigt, wie der Befehl `delete-db-cluster` fehlschlägt, wenn der Cluster noch zugeordnete DB-Instances hat. Dieser Cluster hat eine einzige Writer-DB-Instance. Wenn wir die DB-Instances im Cluster untersuchen, überprüfen wir das Attribut `IsClusterWriter` der einzelnen Instances. Der Cluster könnte eine oder keine Writer-DB-Instance haben. Der Wert `true` steht für eine Writer-DB-Instance. Der Wert `false` steht für eine Reader-DB-Instance. Der Cluster könnte keine, eine oder viele Reader-DB-Instances haben. In diesem Fall löschen wir die Writer-DB-Instance mit dem Befehl `delete-db-instance`. Sobald die DB-Instance in den Status `deleting` wechselt, können wir auch den Cluster löschen. Nehmen wir für dieses Beispiel außerdem an, dass der Cluster keine Daten enthält, die erhalten bleiben müssen. Daher erstellen wir keinen Snapshot des Cluster-Volumes oder behalten automatisierte Backups bei.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-
snapshot
An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
  --query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]'
[
    [
        "deleteme-writer-only",
        "available",
        [
            [
                "instance-2130",
                true
            ]
        ]
    ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only \
  --skip-final-snapshot \
```

```
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Löschen eines Aurora-Clusters mit mehreren DB-Instances

Wenn der Cluster mehrere DB-Instances enthält, gibt es normalerweise eine einzige Writer-Instance und eine oder mehrere Reader-Instances. Die Reader-Instances unterstützen die hohe Verfügbarkeit, indem sie im Standby-Modus sind und übernehmen, wenn die Writer-Instance auf ein Problem stößt. Sie können Reader-Instances auch verwenden, um den Cluster aufwärts zu skalieren, um einen leseintensiven Workload zu bewältigen, ohne zusätzlichen Overhead für die Writer-Instance zu verursachen.

Um einen Cluster mit mehreren Reader-DB-Instances zu löschen, löschen Sie zuerst die Reader-Instances und dann die Writer-Instance. Wenn Sie die Writer-Instance löschen, bleiben der Cluster und seine Daten bestehen. Sie löschen den Cluster separat.

- Informationen zum Löschen einer Aurora-DB-Instance finden Sie unter Löschen einer DB-Instance aus einem Aurora-DB-Cluster.
- Informationen zum Löschen der Writer-DB-Instance in einem Aurora-Cluster finden Sie unter Löschen eines Aurora-Clusters mit einer einzigen DB-Instance.
- Informationen zum Löschen eines leeren Aurora-Clusters finden Sie unter Erstellen eines leeren Aurora-Clusters.

Dieses CLI-Beispiel zeigt, wie ein Cluster gelöscht wird, der sowohl eine Writer-DB-Instance als auch eine einzige Reader-DB-Instance enthält. Die `describe-db-clusters`-Ausgabe zeigt, dass `instance-7384` die Writer-Instance und `instance-1039` die Reader-Instance ist. Im Beispiel wird zuerst die Reader-Instance gelöscht, da das Löschen der Writer-Instance, während noch eine Reader-Instance vorhanden ist, zu einem Failover-Vorgang führen würde. Es ist nicht sinnvoll, die Reader-Instance zu einer Writer-Instance zu befördern, wenn Sie diese Instance ebenfalls löschen möchten. Nehmen wir wiederum an, dass diese `db.t2.small`-Instances nur für Entwicklung und Tests verwendet werden, sodass der Löschvorgang den finalen Snapshot überspringt und keine automatisierten Backups beibehält.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-
final-snapshot

An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-writer-and-reader
Instance:       instance-1039  False
Instance:       instance-7384   True

$ aws rds delete-db-instance --db-instance-identifier instance-1039
{
  "DBInstanceIdentifier": "instance-1039",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384
{
  "DBInstanceIdentifier": "instance-7384",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-and-reader",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Das folgende Beispiel zeigt, wie ein DB-Cluster mit einer Writer-DB-Instance und mehreren Reader-DB-Instances gelöscht wird. Dabei wird die präzise Ausgabe des `describe-db-clusters`-Befehls verwendet, um einen Bericht über die Writer- und Reader-Instances abzurufen. Auch hier löschen wir

alle Reader-DB-Instances, bevor wir die Writer-DB-Instance löschen. Es spielt keine Rolle, in welcher Reihenfolge wir die Reader-DB-Instances löschen.

Angenommen, dieser Cluster mit mehreren DB-Instances enthält Daten, die aufbewahrt werden sollen. Daher enthält der `delete-db-cluster`-Befehl in diesem Beispiel die Parameter `--no-skip-final-snapshot` und `--final-db-snapshot-identifier`, um Details zu dem zu erstellenden Snapshot anzugeben. Enthalten ist außerdem der Parameter `--no-delete-automated-backups` zur Beibehaltung automatisierter Backups.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-multiple-readers
Instance:       instance-1010   False
Instance:       instance-5410   False
Instance:       instance-9948   False
Instance:       instance-8451   True

$ aws rds delete-db-instance --db-instance-identifier instance-1010
{
  "DBInstanceIdentifier": "instance-1010",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410
{
  "DBInstanceIdentifier": "instance-5410",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
  "DBInstanceIdentifier": "instance-9948",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
  "DBInstanceIdentifier": "instance-8451",
```

```
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers \
  --no-delete-automated-backups \
  --no-skip-final-snapshot \
  --final-db-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
  "DBClusterIdentifier": "deleteme-multiple-readers",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Das folgende Beispiel zeigt, wie Sie prüfen können, ob der angeforderte Snapshot von Aurora erstellt wurde. Sie können Details zu diesem Snapshot anfordern, indem Sie seine Kennung angebe `deleteme-multiple-readers-final-snapshot`. Sie können auch einen Bericht zu allen Snapshots für den gelöschten Cluster abrufen, indem Sie die Cluster-Kennung angebe `deleteme-multiple-readers`. Beide Befehle liefern Informationen über denselben Snapshot.

```
$ aws rds describe-db-cluster-snapshots \
  --db-cluster-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...

$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-
readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
```

```
...
```

Löschschutz für Aurora-DB-Cluster

Sie können keine Cluster löschen, für die der Löschschutz aktiviert ist. Sie können DB-Instances innerhalb des Clusters löschen, aber nicht den Cluster selbst. Auf diese Weise ist das Cluster-Volume, das all Ihre Daten enthält, vor einem versehentlichen Löschen geschützt. Aurora erzwingt den Löschschutz für einen DB-Cluster, unabhängig davon, ob Sie versuchen, den Cluster über die Konsole AWS CLI, die oder die RDS-API zu löschen.

Wenn Sie mithilfe der einen Produktions-DB-Cluster erstellen, ist der Löschschutz standardmäßig aktivier AWS-Managementkonsole. Der Löschschutz ist jedoch standardmäßig deaktiviert, wenn Sie einen Cluster mithilfe der AWS CLI OR-API erstellen. Das Aktivieren oder Deaktivieren des Löschschutzes führt nicht zu einem Ausfall. Um den Cluster löschen zu können, müssen Sie den Cluster bearbeiten und den Löschschutz deaktivieren. Weitere Informationen zum Aktivieren und Deaktivieren des Löschschutzes finden Sie unter Ändern des DB-Clusters über die Konsole, die CLI und die API.

> **Tip**
>
> Auch wenn alle DB-Instances gelöscht sind, können Sie auf die Daten zugreifen, indem Sie eine neue DB-Instance im Cluster erstellen.

Löschen eines angehaltenen Aurora-Clusters

Sie können einen Cluster nicht löschen, wenn er sich im Status `stopped` befindet. Starten Sie in diesem Fall den Cluster, bevor Sie ihn löschen. Weitere Informationen finden Sie unter Starten eines Aurora-DB-Clusters.

Löschen von Aurora MySQL-Clustern, die Read Replicas sind

Sie können eine DB-Instance in einem DB-Cluster für Aurora MySQL nicht löschen, wenn beide der folgenden Bedingungen erfüllt sind:

- Der DB-Cluster ist ein Lesereplikat eines anderen Aurora-DB-Clusters.
- Die DB-Instance ist die einzige Instance im DB-Cluster.

Um in diesem Fall eine DB-Instance zu löschen, stufen Sie zunächst den DB-Cluster hoch, sodass er kein Lesereplikat mehr ist. Nach dem Hochstufen können Sie die endgültige DB-Instance im DB-Cluster löschen. Weitere Informationen finden Sie unter Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg.

Der letzte Snapshot beim Löschen eines Clusters

In diesem Abschnitt zeigen die Beispiele, wie Sie auswählen können, ob Sie beim Löschen eines Aurora-Clusters einen endgültigen Snapshot erstellen möchten. Wenn Sie einen endgültigen Snapshot erstellen möchten, der angegebene Name jedoch mit einem vorhandenen Snapshot übereinstimmt, wird der Vorgang mit einem Fehler beendet. Überprüfen Sie in diesem Fall die Snapshot-Details, um zu bestätigen, ob sie Ihr aktuelles Detail darstellt oder ob es sich um einen älteren Snapshot handelt. Wenn der vorhandene Snapshot nicht über die neuesten Daten verfügt, die Sie beibehalten möchten, benennen Sie den Snapshot um und versuchen Sie es erneut, oder geben Sie einen anderen Namen für den Parameter endgültiger Snapshot aus.

Löschen einer DB-Instance aus einem Aurora-DB-Cluster

Sie können eine DB-Instance aus einem Aurora-DB-Cluster im Rahmen des Löschens des gesamten Clusters löschen. Wenn Ihr Cluster eine bestimmte Anzahl von DB-Instances enthält, so müssen zum Löschen des Clusters all diese DB-Instances gelöscht werden. Sie können auch eine oder mehrere Reader-Instances aus einem Cluster löschen, während der Cluster weiter ausgeführt wird. Dadurch können Sie die Rechenkapazität und die damit verbundenen Kosten reduzieren, wenn Ihr Cluster nicht ausgelastet ist.

Sie müssen den Namen der Instance angeben, um eine DB-Instance zu löschen.

Sie können eine DB-Instance mithilfe der AWS-Managementkonsole AWS CLI, der oder der RDS-API löschen.

> **Note**
>
> Wenn einAurora-Replikat gelöscht wird, wird der Instance-Endpunkt sofort entfernt und dasAurora-Replikat wird vom Reader-Endpunkt entfernt. Wenn Anweisungen vorhanden sind, die auf dem Aurora Replica ausgeführt werden, das gerade gelöscht wird, besteht eine Übergangsfrist von drei Minuten. Vorhandene Anweisungen können während der Nachfrist beendet werden. Nach Ablauf der Nachfrist wird das Aurora-Replikat geschlossen und gelöscht.

Bei Aurora-DB-Clustern wird durch das Löschen einer DB-Instance nicht unbedingt der gesamte Cluster gelöscht. Sie können eine DB-Instance in einem Aurora-Cluster löschen, um die Rechenkapazität und die damit verbundenen Kosten zu reduzieren, wenn der Cluster nicht ausgelastet ist. Informationen zu den besonderen Bedingungen bei Aurora-Clustern mit einer einzigen oder keiner DB-Instance finden Sie unter Löschen eines Aurora-Clusters mit einer einzigen DB-Instance und Erstellen eines leeren Aurora-Clusters.

> **Note**
>
> Sie können einen DB-Cluster nicht löschen, wenn für ihn Löschschutz aktiviert ist. Weitere Informationen finden Sie unter Löschschutz für Aurora-DB-Cluster.
> Sie können den Löschschutz deaktivieren, indem Sie den DB-Cluster bearbeiten. Weitere Informationen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Konsole

So löschen Sie eine DB-Instance in einem DB-Cluster

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die DB-Instance aus, die Sie löschen möchten.
3. Klicken Sie bei Actions auf Delete.
4. Geben Sie **`delete me`** in das Feld ein.
5. Wählen Sie Löschen aus.

AWS CLI

Um eine DB-Instance mithilfe von zu löschen AWS CLI, rufen Sie den delete-db-instanceBefehl auf und geben Sie den `--db-instance-identifier` Wert an.

Example

Für Linux, macOS oder Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance
```

Für Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS-API

Um eine DB-Instance mithilfe der Amazon RDS-API zu löschen, rufen Sie den DBInstance Vorgang Delete auf und geben Sie den `DBInstanceIdentifier` Parameter an.

> **Note**
>
> Wenn der Status einer DB-Instance lautet`deleting`, erscheint ihr CA-Zertifikatswert nicht in der RDS-Konsole oder in der Ausgabe für AWS CLI Befehle oder RDS-API-Operationen. Weitere Informationen zu CA-Zertifikaten finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen

Ein Amazon-RDS-Tag ist ein Name-Wert-Paar, das Sie definieren und mit einer Amazon-RDS-Ressource wie einer DB-Instance oder einem DB-Snapshot verknüpfen. Der Name wird als der Schlüssel bezeichnet. Optional können Sie einen Wert für den Schlüssel angeben.

Sie können die AWS-Managementkonsole AWS CLI, oder die Amazon RDS-API verwenden, um Tags zu Amazon RDS-Ressourcen hinzuzufügen, aufzulisten und zu löschen. Wenn Sie die CLI oder API verwenden, müssen Sie den Amazon-Ressourcennamen (ARN) für die RDS-Ressource angeben, mit der Sie arbeiten möchten. Weitere Informationen zum Konstruieren eines ARN finden Sie unter Erstellen eines ARN für Amazon RDS.

Mithilfe von Tags fügen Sie Metadaten zu Ihren Aurora- und Amazon-RDS-Ressourcen hinzu. Sie können die Tags verwenden, um Ihre eigenen Notationen unter anderem zu Datenbank-Instances, Snapshots und Aurora-Clustern hinzuzufügen. Dies kann Ihnen dabei helfen, Ihre Aurora- und Amazon-RDS-Ressourcen zu dokumentieren. Sie können die Tags auch mit automatisierten Wartungsvorgängen verwenden.

Insbesondere können Sie diese Tags mit IAM-Richtlinien verwenden. Sie können sie verwenden, um den Zugriff auf Aurora- und Amazon-RDS-Ressourcen zu verwalten und zu steuern, welche Aktionen auf diese Ressourcen angewendet werden können. Sie können Tags auch verwenden, um Kosten zu verfolgen, indem Ausgaben für ähnlich markierte Ressourcen gruppiert werden.

Sie können die folgenden Aurora- und Amazon-RDS-Ressourcen mit Tags versehen:

- DB-Instances
- DB-Cluster
- Globale Aurora-Cluster
- DB-Cluster-Endpunkte
- Read Replicas
- DB-Snapshots
- DB-Cluster-Snapshots
- Reservierte DB-Instances
- Ereignisabonnements
- DB-Optionsgruppen
- DB-Parametergruppen

- DB-Cluster-Parametergruppen
- DB-Subnetzgruppen
- RDS-Proxys
- RDS-Proxy-Endpunkte
- Blau/Grün-Bereitstellungen
- Zero-ETL-Integrationen

> **Note**
>
> Wenn Sie eine DB-Instance taggen, wendet Aurora diese Tags automatisch auf die zugehörigen Performance Insights Insights-Ressourcen an. Derzeit können Sie RDS-Proxys und RDS-Proxy-Endpunkte nicht mit dem taggen. AWS-Managementkonsole

Themen

- Warum Tags für Amazon-RDS-Ressourcen verwenden?
- So funktionieren Tags für Amazon-RDS-Ressourcen
- Bewährte Methoden für das Taggen von Amazon-RDS-Ressourcen
- Kopieren von Tags in DB-Cluster-Snapshots
- Hinzufügen und Löschen von Tags in Amazon RDS
- Tutorial: Verwenden Sie Tags, um festzulegen, welcher Aurora-DB-Cluster stoppt.

Warum Tags für Amazon-RDS-Ressourcen verwenden?

Mit Tags können Sie die folgenden Aktionen ausführen:

- Kategorisieren Sie Ihre RDS-Ressourcen nach Anwendung, Projekt, Abteilung, Umgebung usw. Sie können einen Tag-Schlüssel z. B. dazu verwenden, eine Kategorie zu definieren, wobei ein Tag-Wert ein Element in dieser Kategorie ist. Sie könnten das Tag `environment=prod` erstellen. Alternativ könnten Sie beispielsweise den Tag-Schlüssel `project` und den Tag-Wert `Salix` definieren, was angibt, dass die Amazon-RDS-Ressource dem Projekt „Salix“ zugewiesen ist.
- Automatisieren Sie Aufgaben im Ressourcenmanagement. Sie könnten beispielsweise ein Wartungsfenster für Instances mit dem Tag `environment=prod` erstellen, das sich vom Fenster

für Instances mit dem Tag `environment=test` unterscheidet. Sie könnten auch automatische DB-Snapshots für markierte Instances mit dem Tag `environment=prod` konfigurieren.

- Steuern Sie den Zugriff auf RDS-Ressourcen innerhalb einer IAM-Richtlinie. Hierzu können Sie den globalen Bedingungsschlüssel `aws:ResourceTag/tag-key` verwenden. Eine Richtlinie könnte es beispielsweise nur Benutzern in der Gruppe `DBAdmin` gestatten, DB-Instances mit dem Tag `environment=prod` zu ändern. Informationen zur Verwaltung des Zugriffs auf markierte Ressourcen mit IAM-Richtlinien finden Sie unter Identity and Access Management für Amazon Aurora Steuern des Zugriffs auf AWS Ressourcen im AWS Identity and Access Management-Benutzerhandbuch.
- Überwachen Sie Ressourcen anhand eines Tags. Sie können beispielsweise ein CloudWatch Amazon-Dashboard für DB-Instances erstellen, die mit gekennzeichnet sind`environment=prod`.
- Verfolgen Sie Kosten, indem Ausgaben für Ressourcen mit ähnlichen Tags gruppiert werden. Wenn Sie beispielsweise RDS-Ressourcen, die mit dem Projekt „Salix“ verknüpft sind, mit dem Tag `project=Salix` versehen, können Sie Kostenberichte für dieses Projekt erstellen und Ausgaben diesem Projekt zuordnen. Weitere Informationen finden Sie unter So funktioniert die AWS Abrechnung mit Tags in Amazon RDS.

So funktionieren Tags für Amazon-RDS-Ressourcen

AWS wendet Ihren Tags keine semantische Bedeutung an. Tags werden streng als Zeichenfolgen interpretiert.

Themen

- Tag-Sätze in Amazon RDS
- Tag-Struktur in Amazon RDS
- Für das Taggen infrage kommende Amazon-RDS-Ressourcen
- So funktioniert die AWS Abrechnung mit Tags in Amazon RDS

Tag-Sätze in Amazon RDS

Jede Amazon-RDS-Ressource hat einen Container, der als Tag-Satz bezeichnet wird. Der Container enthält alle Tags, die der Ressource zugewiesen sind. Eine Ressource hat genau einen Tag-Satz.

Ein Tag-Satz enthält 0–50 Tags. Wenn Sie einer RDS-Ressource ein Tag mit demselben Schlüssel hinzufügen wie ein bereits vorhandenes Tag der Ressource, überschreibt der neue Wert den alten.

Tag-Struktur in Amazon RDS

Die Struktur eines RDS-Tags sieht wie folgt aus:

Tag-Schlüssel

Der Schlüssel ist der erforderliche Name des Tags. Der Zeichenfolgenwert muss aus 1 bis 128 Unicode-Zeichen bestehen. Ihm darf kein „`aws:`“ oder „`rds:`“ als Präfix vorangestellt werden. Die Zeichenfolge darf nur Unicode-Zeichen, Ziffern, Leerzeichen sowie _, `.`, `:`, `/`, =, +, `-` und `@` enthalten. Der Java-Regex ist `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`. Bei Tag-Schlüsseln wird die Groß- und Kleinschreibung beachtet. Somit sind die Schlüssel `project` und `Project` unterschiedlich.

Ein Schlüssel in einem Tag-Satz ist eindeutig. Sie können z. B. in einem Tag-Satz kein Schlüsselpaar mit gleichem Schlüssel, aber unterschiedlichen Werten verwenden, wie `project=Trinity` und `project=Xanadu`.

Tag-Wert

Der Tag-Wert ist ein optionaler Zeichenfolgenwert des Tags. Der Zeichenfolgenwert muss aus 1 bis 256 Unicode-Zeichen bestehen. Die Zeichenfolge darf nur Unicode-Zeichen, Ziffern, Leerzeichen sowie _, `.`, `:`, `/`, =, +, `-` und `@` enthalten. Der Java-Regex ist `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`. Bei Tag-Werten muss die Groß- und Kleinschreibung beachtet werden. Somit sind die Werte `prod` und `Prod` unterschiedlich.

Die Werte innerhalb eines Tag-Satzes müssen nicht eindeutig und können null sein. Es ist z. B. ein Schlüssel-Wert-Paar in einem Tag-Satz `project=Trinity` und `cost-center=Trinity` möglich.

Für das Taggen infrage kommende Amazon-RDS-Ressourcen

Sie können die folgenden Amazon-RDS-Ressourcen kennzeichnen:

- DB-Instances
- DB-Cluster
- DB-Cluster-Endpunkte
- Read Replicas
- DB-Snapshots

- DB-Cluster-Snapshots
- Reservierte DB-Instances
- Ereignisabonnements
- DB-Optionsgruppen
- DB-Parametergruppen
- DB-Cluster-Parametergruppen
- DB-Subnetzgruppen
- RDS-Proxys
- RDS-Proxy-Endpunkte

> **Note**
>
> Derzeit können Sie RDS-Proxys und RDS-Proxy-Endpunkte nicht per AWS-Managementkonsole markieren.

- Blau/Grün-Bereitstellungen
- Null-ETL-Integrationen (Vorschau)

So funktioniert die AWS Abrechnung mit Tags in Amazon RDS

Verwenden Sie Tags, um Ihre AWS Rechnung so zu organisieren, dass sie Ihrer eigenen Kostenstruktur entspricht. Melden Sie sich dazu an, um Ihre AWS-Konto Rechnung mit den Tag-Schlüsselwerten zu erhalten. Um dann die Kosten kombinierter Ressourcen anzuzeigen, organisieren Sie Ihre Fakturierungsinformationen nach Ressourcen mit gleichen Tag-Schlüsselwerten. Beispielsweise können Sie mehrere Ressourcen mit einem bestimmten Anwendungsnamen markieren und dann Ihre Fakturierungsinformationen so organisieren, dass Sie die Gesamtkosten dieser Anwendung über mehrere Services hinweg sehen können. Weitere Informationen finden Sie unter Verwendung von Kostenzuordnungs-Tags im AWS Billing -Benutzerhandbuch.

So funktionieren Kostenzuordnungs-Tags mit DB-Cluster-Snapshots

Sie können einem DB-Cluster-Snapshot ein Tag hinzufügen. Diese Gruppierung erscheint jedoch nicht in Ihrer Rechnung. Für die Anwendung von Kostenzuordnung-Tags für DB-Cluster-Snapshots müssen die folgenden Bedingungen erfüllt sei:

- Die Tags müssen an die übergeordnete DB-Instance angehängt werden.

- Die übergeordnete DB-Instance muss genauso vorhanden sein AWS-Konto wie der DB-Cluster-Snapshot.
- Die übergeordnete DB-Instance muss genauso vorhanden sein AWS-Region wie der DB-Cluster-Snapshot.

DB-Cluster-Snapshots gelten als verwaist, wenn sie nicht in derselben Region wie die übergeordnete DB-Instance existieren oder wenn die übergeordnete DB-Instance gelöscht wurde. Verwaiste DB-Snapshots unterstützen keine Kostenzuordnung-Tags. Die Kosten für verwaiste Snapshots werden in einer einzigen Zeile ohne Tags zusammengefasst. Kontoübergreifende DB-Cluster-Snapshots gelten nicht als verwaist, wenn die folgenden Bedingungen erfüllt sind:

- Sie müssen sich in derselben Region wie die DB-Instance befinden.
- Das Quellkonto ist Eigentümer der übergeordneten DB-Instance.

> **Note**
>
> Wenn ein anderes Konto Eigentümer der übergeordneten DB-Instance ist, gelten die Kostenzuordnungs-Tags nicht für kontenübergreifende Snapshots im Zielkonto.

Bewährte Methoden für das Taggen von Amazon-RDS-Ressourcen

Wir empfehlen Ihnen, bei der Verwendung von Tags folgende bewährte Methode befolgen:

- Dokumentieren Sie Konventionen für die Verwendung von Tags, die von allen Teams in Ihrer Organisation befolgt werden. Stellen Sie insbesondere sicher, dass die Namen sowohl beschreibend als auch einheitlich sind. Verwenden Sie beispielsweise standardmäßig das Format `environment:prod`, anstatt einige Ressourcen mit dem Tag `env:production` zu versehen.

> **Important**
>
> Speichern Sie keine personenbezogenen Daten (PII) oder andere vertrauliche Informationen in Tags.

- Automatisieren Sie das Tagging, um Einheitlichkeit zu gewährleisten. Sie können die folgenden Techniken verwenden:

  - Fügen Sie Tags in eine CloudFormation Vorlage ein. Wenn Sie Ressourcen mit der Vorlage erstellen, werden die Ressourcen automatisch mit Tags versehen.
  - Definieren und wenden Sie Tags mithilfe von AWS Lambda Funktionen an.
  - Erstellen Sie ein SSM-Dokument, das Schritte zum Hinzufügen von Tags zu Ihren RDS-Ressourcen enthält.
- Verwenden Sie Tags nur bei Bedarf. Sie können bis zu 50 Tags für eine einzelne RDS-Ressource hinzufügen. Eine bewährte Methode besteht jedoch darin, unnötig viele und komplexe Tags zu vermeiden.
- Überprüfen Sie die Tags regelmäßig auf Relevanz und Richtigkeit. Entfernen oder ändern Sie veraltete Tags nach Bedarf.
- Erwägen Sie das Erstellen von Tags mit dem AWS Tag-Editor in der AWS-Managementkonsole. Sie können den Tag-Editor verwenden, um mehreren unterstützten AWS Ressourcen, einschließlich RDS-Ressourcen, gleichzeitig Tags hinzuzufügen. Weitere Informationen finden Sie unter Tag Editor im Benutzerhandbuch von AWS Resource Groups.

Kopieren von Tags in DB-Cluster-Snapshots

Wenn Sie einen DB-Cluster erstellen oder wiederherstellen, können Sie festlegen, dass die Tags aus dem Cluster in Snapshots des DB-Clusters kopiert werden. Das Kopieren von Tags stellt sicher, dass die Metadaten für die DB-Snapshots mit denen des Quell-DB-Clusters übereinstimmen. Es wird außerdem sichergestellt, dass alle Zugriffsrichtlinien für die DB-Snapshots auch mit denen des Quell-DB-Clusters übereinstimmen. Tags werden nicht standardmäßig kopiert.

Sie können für die folgenden Aktionen festlegen, dass Tags in DB-Snapshots kopiert werden:

- Erstellen eines DB-Clusters
- Wiederherstellen eines DB-Clusters
- Erstellen eines Lesereplikats
- Kopieren eines DB-Cluster-Snapshots

> **Note**
>
> In einigen Fällen können Sie einen Wert für den `--tags` Parameter des create-db-snapshot AWS CLI Befehls angeben. Oder Sie können mindestens ein Tag für den Vorgang Create DBSnapshot API angeben. In diesen Fällen kopiert RDS keine Tags von der Quell-DB-

Instance in den neuen DB-Snapshot. Diese Funktionalität gilt sogar, wenn in der Quell-DB-Instance die Option `--copy-tags-to-snapshot` (`CopyTagsToSnapshot`) aktiviert ist. Wenn Sie diesen Ansatz verwenden, können Sie eine Kopie einer DB-Instance aus einem DB-Snapshot erstellen. Dieser Ansatz vermeidet das Hinzufügen von Tags, die nicht für die neue DB-Instance gelten. Sie erstellen Ihren DB-Snapshot mit dem AWS CLI `create-db-snapshot` Befehl (oder dem `CreateDBSnapshot` RDS-API-Vorgang). Nachdem Sie Ihren DB-Snapshot erstellt haben, können Sie Tags hinzufügen. Dieser Vorgang wird später in diesem Thema beschrieben.

Hinzufügen und Löschen von Tags in Amazon RDS

Sie haben die folgenden Möglichkeiten:

- Erstellen Sie Tags, wenn Sie eine Ressource erstellen, z. B. wenn Sie den AWS CLI Befehl ausführen`create-db-instance`.
- Fügen Sie Tags über den Befehl `add-tags-to-resource` zu einer vorhandenen Ressource hinzu.
- Listen Sie über den Befehl `list-tags-for-resource` Tags auf, die einer bestimmten Ressource zugeordnet sind.
- Aktualisieren Sie die Tags über den Befehl `add-tags-to-resource`.
- Entfernen Sie Tags über den Befehl `remove-tags-from-resource` aus einer Ressource.

Die folgenden Verfahren zeigen, wie typische Tagging-Operationen für Ressourcen im Zusammenhang mit DB-Instances und Aurora-DB-Clustern durchgeführt werden. Beachten Sie, dass Tags für Autorisierungszwecke zwischengespeichert werden. Aus diesem Grund können beim Hinzufügen oder Aktualisieren von Tags zu Amazon-RDS-Ressourcen mehrere Minuten vergehen, bis die Änderungen sichtbar sind.

Konsole

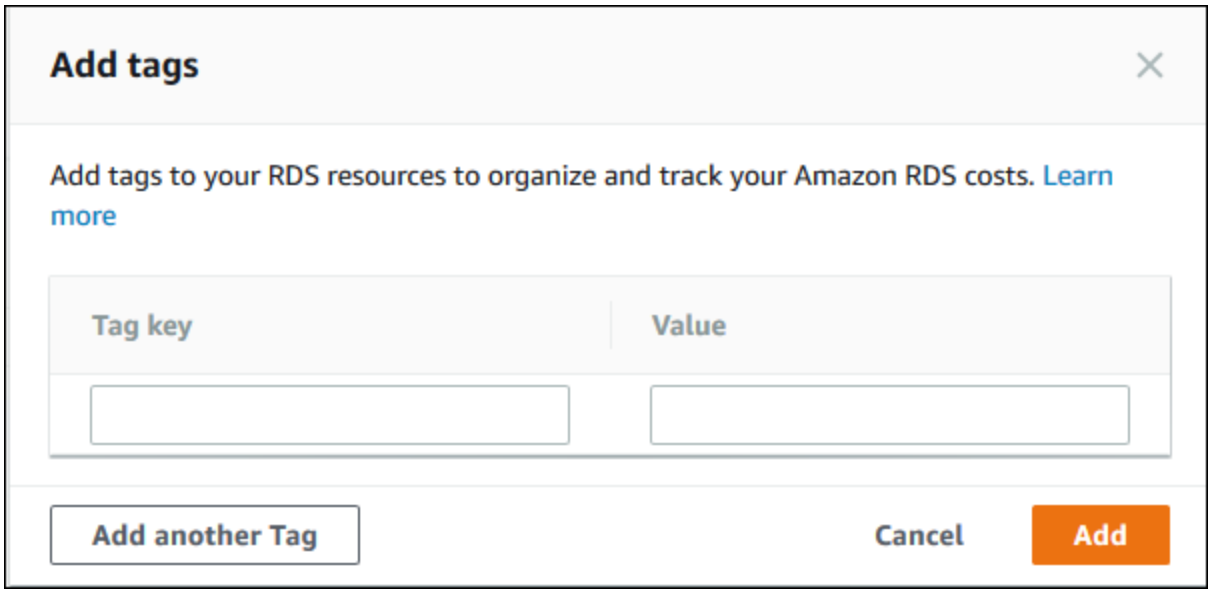
Amazon-RDS-Ressourcen werden auf die gleiche Weise wie andere Ressourcen getaggt. Die folgenden Schritte zeigen, wie Sie eine Amazon-RDS-DB-Instance taggen.

So fügen Sie ein Tag zu einer DB-Instance hinzu

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.

   > **Note**
   >
   > Geben Sie unter Filter databases (Datenbanken filtern) eine Textzeichenfolge ein, um die Liste der DB-Instances im Bereich Databases (Datenbanken) zu filtern. Es werden nur DB-Instances angezeigt, welche die Zeichenfolge enthalten.

3. Wählen Sie den Namen der DB-Instance aus, die Sie mit einem Tag versehen möchten, um deren Details anzuzeigen.
4. Scrollen Sie im Detailbereich nach unten zum Bereich Tags.
5. Wählen Sie Add aus. Das Fenster Add tags (Tags hinzufügen) wird angezeigt.



6. Geben Sie einen Wert für Tag key (Tag-Schlüssel) und Wert ein.
7. Wenn Sie ein weiteres Tag hinzufügen möchten, wählen Sie Add another Tag (Weiteres Tag hinzufügen) aus und geben Sie unter Tag key (Tag-Schlüssel) und Wert einen Wert ein.

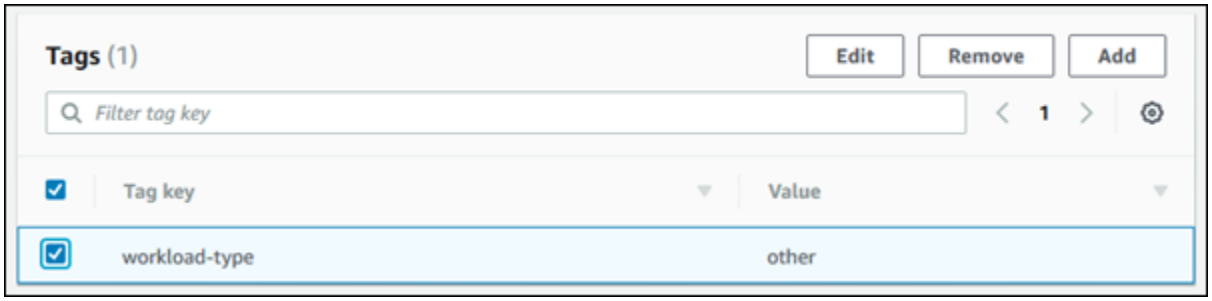
   Wiederholen Sie diesen Schritt, bis Sie alle Tags hinzugefügt haben.
8. Wählen Sie Add aus.

So löschen Sie ein Tag aus einer DB-Instance

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.

   > **Note**
   >
   > Geben Sie im Feld Databases (Datenbanken) eine Textzeichenfolge ein, um die Liste der DB-Instances im Bereich Databases (Datenbanken) zu filtern. Es werden nur DB-Instances angezeigt, welche die Zeichenfolge enthalten.

3. Wählen Sie den Namen der entsprechenden DB-Instance aus, um deren Details anzuzeigen.
4. Scrollen Sie im Detailbereich nach unten zum Bereich Tags.
5. Wählen Sie das Tag aus, das Sie löschen möchten.



6. Wählen Sie Löschen und dann im Fenster Löschen von Tags erneut Löschen aus.

AWS CLI

Mithilfe der können Sie Tags für eine DB-Instance hinzufügen, auflisten oder entferne AWS CLI.

- Verwenden Sie den AWS CLI Befehl, um einer Amazon RDS-Ressource ein oder mehrere Tags hinzuzufügen `add-tags-to-resource`.
- Verwenden Sie den AWS CLI Befehl, um die Tags auf einer Amazon RDS-Ressource aufzulisten `list-tags-for-resource`.
- Verwenden Sie den AWS CLI Befehl, um ein oder mehrere Tags aus einer Amazon RDS-Ressource zu entfernen `remove-tags-from-resource`.

Weitere Informationen zum Erstellen des erforderlichen ARN finden Sie unter Erstellen eines ARN für Amazon RDS.

RDS-API

Mithilfe der Amazon-RDS-API können Sie Tags für eine DB-Instance hinzufügen, auflisten oder entfernen.

- Verwenden Sie die API-Operation `AddTagsToResource`, um ein Tag zu einer Amazon-RDS-Ressource hinzuzufügen.
- Verwenden Sie die API-Operation , um die Tags für eine Amazon-RDS-Ressource aufzulisten `ListTagsForResource`.
- Verwenden Sie die API-Operation `RemoveTagsFromResource`, um Tags aus einer Amazon-RDS-Ressource zu entfernen.

Weitere Informationen zum Erstellen des erforderlichen ARN finden Sie unter Erstellen eines ARN für Amazon RDS.

Wenn Sie in der Amazon-RDS-API mit XML arbeiten, nutzen Sie das folgende Schema:

```
<Tagging>
      <TagSet>
          <Tag>
              <Key>Project</Key>
              <Value>Trinity</Value>
          </Tag>
          <Tag>
              <Key>User</Key>
              <Value>Jones</Value>
          </Tag>
      </TagSet>
  </Tagging>
```

Die folgende Tabelle enthält eine Liste der zulässigen XML-Tags und deren Eigenschaften. Bei Werten für `Key` und `Value` wird die Groß-/Kleinschreibung berücksichtigt. Beispiel: `project=Trinity` und `PROJECT=Trinity` sind verschiedene Tags.

Markieren von Elementen	Description
TagSet	Ein Tag-Satz ist ein Container für alle Tags, die einer Amazon Amazon-RDS-Ressource zugewiesen sind. Es ist nur ein Tag-Satz pro Ressource zulässig. Sie arbeiten mit einem TagSet nur über die Amazon RDS-API.
Tag	Ein Tag ist ein benutzerdefiniertes Schlüssel-Wert-Paar. Ein Tag-Satz kann 1 bis 50 Tags enthalten.
Schlüssel	Ein Schlüssel ist der erforderliche Name des Tags. Einschränkungen finden Sie unter Tag-Struktur in Amazon RDS. Der Zeichenfolgenwert kann aus 1 bis 128 Unicode-Zeichen bestehen. Ihm darf kein "`aws:`" oder "`rds:`" als Präfix vorangestellt werden. Die Zeichenfolge darf nur Unicode-Zeichen, Ziffern, Leerzeichen sowie "_", ".", "/", "=", "+", "-" enthalten (Java-Regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)"). Schlüssel müssen in einem Tag-Satz eindeutig sein. Sie können beispielsweise kein Schlüsselpaar in einem Tag-Set mit demselben Schlüssel, aber mit unterschi edlichen Werten haben, wie project/Trinity z. B. Project/Xanadu.
Wert	Ein Wert ist der optionale Wert des Tags. Einschränkungen finden Sie unter Tag-Struktur in Amazon RDS. Der Zeichenfolgenwert kann aus 1 bis 256 Unicode-Zeichen bestehen. Ihm darf kein "`aws:`" oder "`rds:`" als Präfix vorangestellt werden. Die Zeichenfolge darf nur Unicode-Zeichen, Ziffern, Leerzeichen sowie "_", ".", "/", "=", "+", "-" enthalten (Java-Regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)"). Die Werte innerhalb eines Tag-Satzes müssen nicht eindeutig und können null sein. Sie können beispielsweise ein Schlüssel-Wert-Paar in einem Tag-Set von und Cost-Center/Trinity haben. project/Trinity

Tutorial: Verwenden Sie Tags, um festzulegen, welcher Aurora-DB-Cluster stoppt.

Angenommen, Sie erstellen eine Reihe von Aurora-DB-Clustern in einer Entwicklungs- oder Testumgebung. Sie müssen alle diese Cluster mehrere Tage lang beibehalten. Einige der Cluster führen über Nacht Tests durch. Andere Cluster können über Nacht angehalten und am nächsten Tag wieder gestartet werden. Das folgende Beispiel zeigt, wie Sie denjenigen Clustern ein Tag zuweisen, die sich dafür eignen, über Nacht angehalten zu werden. Dann zeigt das Beispiel, wie ein Skript erkennen kann, welche Cluster dieses Tag haben und wie diese Cluster anschließend angehalten werden. In diesem Beispiel spielt der Wertanteil des Schlüssel-Wert-Paares keine Rolle. Das Vorhandensein des `stoppable`-Tags bedeutet, dass das Cluster diese benutzerdefinierte Eigenschaft besitzt.

Festlegen, welche Aurora-DB-Cluster angehalten werden sollen

1. Bestimmen Sie den ARN eines Clusters, den Sie als anhaltbar kennzeichnen wollen.

   Die Befehle und APIs zum Markieren funktionieren mit Hilfe von ARNs. Auf diese Weise können sie nahtlos über AWS-Regionen, AWS-Konten und verschiedene Ressourcentypen hinweg arbeiten, die möglicherweise identische Kurznamen haben. Sie können in CLI-Befehlen, die mit Clustern arbeiten, den ARN anstelle der Cluster-ID angeben. Ersetzen Sie den Namen Ihres eigenen Clusters durch *`dev-test-cluster`*. Ersetzen Sie in nachfolgenden Befehlen mit ARN-Parametern den ARN Ihres eigenen Clusters. Der ARN enthält Ihre eigene AWS-Konto-ID und den Namen der AWS-Region, in der sich Ihr Cluster befindet.

   ```
   $ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
     --query "*[].{DBClusterArn:DBClusterArn}" --output text
   arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
   ```

2. Fügen Sie das Tag `stoppable` zu diesem Cluster hinzu.

   Der Name für dieses Tag wird von Ihnen ausgewählt. Dieser Ansatz bedeutet, dass Sie vermeiden können, eine Namenskonvention zu entwickeln, die alle relevanten Informationen in Namen codiert. In einer solchen Konvention können Sie Informationen im DB-Instance-Namen oder Namen anderer Ressourcen codieren. Da das Tag in diesem Beispiel als Attribut behandelt wird, das entweder vorhanden ist oder nicht, wird der `Value=`-Teil des `--tags`-Parameters weggelassen.

   ```
   $ aws rds add-tags-to-resource \
   ```

```
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
  --tags Key=stoppable
```

3. Bestätigen Sie, dass das Tag im Cluster vorhanden ist.

   Diese Befehle rufen die Tag-Informationen für das Cluster im JSON-Format und in einfachem tabulatorgetrenntem Text ab.

```
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
{
    "TagList": [
        {
            "Key": "stoppable",
            "Value": ""

        }
    ]
}
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
 text
TAGLIST stoppable
```

4. Um alle Cluster anzuhalten, die als `stoppable` bezeichnet werden, bereiten Sie eine Liste aller Ihrer Cluster vor. Gehen Sie die Liste durch und prüfen Sie, ob jedes Cluster mit dem entsprechenden Attribut gekennzeichnet ist.

   Dieses Linux-Beispiel verwendet Shell-Skripts, um die Liste der Cluster-ARNs in einer temporären Datei zu speichern und dann CLI-Befehle für jedes Cluster auszuführen.

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 'TAGLIST\tstoppable')"
  if [[ ! -z "$match" ]]
  then
      echo "Cluster $arn is tagged as stoppable. Stopping it now."
# Note that you can specify the full ARN value as the parameter instead of the
 short ID 'dev-test-cluster'.
      aws rds stop-db-cluster --db-cluster-identifier $arn
```

```
  fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
 stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
        ...
```

Sie können ein solches Skript am Ende eines jeden Tages ausführen, um sicherzustellen, dass nicht benötigte Cluster angehalten werden. Sie können einen Job auch mit einem Dienstprogramm wie `cron` planen, um jede Nacht eine solche Überprüfung durchzuführen. Sie könnten so beispielsweise vorgehen, wenn einige Cluster versehentlich weiter ausgeführt wurden. Hier könnten Sie eine Feinabstimmung des Befehls vornehmen, mit dem die Liste der zu prüfenden Cluster vorbereitet wird.

Der folgende Befehl erzeugt eine Liste Ihrer Cluster mit dem Status `available`. Das Skript kann Cluster ignorieren, die bereits angehalten wurden, da sie unterschiedliche Statuswerte wie `stopped` oder `stopping` haben.

```
$ aws rds describe-db-clusters \
  --query '*[].{DBClusterArn:DBClusterArn,Status:Status}|[?Status == `available`]|[].
{DBClusterArn:DBClusterArn}' \
  --output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

> **Tip**
>
> Sie können die Tags zuweisen und Cluster mit Hilfe dieser Tags finden, um die Kosten auf andere Weise zu senken. Nehmen wir zum Beispiel das Szenario mit Aurora-DB-Clustern,

die für Entwicklungs- und Testzwecke verwendet werden. Hier können Sie festlegen, dass einige Cluster am Ende eines jeden Tages gelöscht werden oder dass nur ihre Reader-DB-Instances gelöscht werden. Oder Sie können einige so festlegen, dass ihre DB-Instances in Zeiten erwarteter geringer Auslastung in kleine DB-Instance-Klassen geändert werden.

Amazon-Ressourcennamen (ARN) in Amazon RDS

In Amazon Web Services erstellte Ressourcen werden anhand eines Amazon-Ressourcennamens (ARN) eindeutig identifiziert. Für bestimmte Amazon-RDS-Operationen müssen Sie eine Amazon-RDS-Ressource eindeutig identifizieren, indem Sie ihren ARN angeben. Wenn Sie beispielsweise ein Lesereplikat einer RDS-DB-Instance erstellen, müssen Sie den ARN für die Quell-DB-Instance bereitstellen.

Informationen zum Konstruieren eines ARN und zum Abrufen eines vorhandenen ARN finden Sie in den folgenden Themen.

Themen

- Erstellen eines ARN für Amazon RDS
- Abrufen eines vorhandenen ARN für Amazon RDS

Erstellen eines ARN für Amazon RDS

In Amazon Web Services erstellte Ressourcen werden anhand eines Amazon-Ressourcennamens (ARN) eindeutig identifiziert. Sie können mithilfe der folgenden Syntax einen ARN für eine Amazon-RDS-Ressource erstellen.

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

Für globale Clusterressourcen enthält der ARN kein AWS-Region:`arn:aws:rds::<account number>:global-cluster:<name>`. ARNs für globale Cluster erscheinen nicht in der AWS-Managementkonsole.

Name der Region	Region	Endpunkt	Protocol (Protokoll)	
USA Ost (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
		rds-fips.us-east-2.api.aws	HTTPS	
		rds.us-east-2.api.aws	HTTPS	
		rds-fips.us-east-2.amazonaws.com	HTTPS	

Name der Region	Region	Endpunkt	Protocol (Protokoll)
USA Ost (Nord-Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS
		rds-fips.us-east-1.api.aws	HTTPS
		rds-fips.us-east-1.amazonaws.com	HTTPS
		rds.us-east-1.api.aws	HTTPS
USA West (Nordkalifornien)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
		rds.us-west-1.api.aws	HTTPS
		rds-fips.us-west-1.amazonaws.com	HTTPS
		rds-fips.us-west-1.api.aws	HTTPS
USA West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
		rds-fips.us-west-2.amazonaws.com	HTTPS
		rds.us-west-2.api.aws	HTTPS
		rds-fips.us-west-2.api.aws	HTTPS
Afrika (Kapstadt)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
		rds.af-south-1.api.aws	HTTPS
Asien-Pazifik (Hongkong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
		rds.ap-east-1.api.aws	HTTPS
Asien-Pazifik (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
		rds.ap-south-2.api.aws	HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Asien-Pazifik (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com rds.ap-southeast-3.api.aws	HTTPS HTTPS
Asien-Pazifik (Malaysia)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS
Asien-Pazifik (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com rds.ap-southeast-4.api.aws	HTTPS HTTPS
Asien-Pazifik (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com rds.ap-south-1.api.aws	HTTPS HTTPS
Asien-Pazifik (Neuseeland)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS
Asien-Pazifik (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com rds.ap-northeast-3.api.aws	HTTPS HTTPS
Asien-Pazifik (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com rds.ap-northeast-2.api.aws	HTTPS HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Asien-Pazifik (Singapur)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
		rds.ap-southeast-1.api.aws	HTTPS
Asien-Pazifik (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
		rds.ap-southeast-2.api.aws	HTTPS
Asien-Pazifik (Taipeh)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS
Asien-Pazifik (Thailand)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS
Asien-Pazifik (Tokio)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
		rds.ap-northeast-1.api.aws	HTTPS
Kanada (Zentral)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
		rds.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.amazonaws.com	HTTPS
Kanada West (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
		rds-fips.ca-west-1.amazonaws.com	HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Europa (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com rds.eu-central-1.api.aws	HTTPS HTTPS
Europa (Irland)	eu-west-1	rds.eu-west-1.amazonaws.com rds.eu-west-1.api.aws	HTTPS HTTPS
Europa (London)	eu-west-2	rds.eu-west-2.amazonaws.com rds.eu-west-2.api.aws	HTTPS HTTPS
Europa (Mailand)	eu-south-1	rds.eu-south-1.amazonaws.com rds.eu-south-1.api.aws	HTTPS HTTPS
Europa (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com rds.eu-west-3.api.aws	HTTPS HTTPS
Europa (Spanien)	eu-south-2	rds.eu-south-2.amazonaws.com rds.eu-south-2.api.aws	HTTPS HTTPS
Europa (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com rds.eu-north-1.api.aws	HTTPS HTTPS
Europa (Zürich)	eu-central-2	rds.eu-central-2.amazonaws.com rds.eu-central-2.api.aws	HTTPS HTTPS
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com rds.il-central-1.api.aws	HTTPS HTTPS

Name der Region	Region	Endpunkt	Protocol (Protokoll)
Mexiko (Zentral)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS
Naher Osten (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com rds.me-south-1.api.aws	HTTPS HTTPS
Naher Osten (VAE)	me-central-1	rds.me-central-1.amazonaws.com rds.me-central-1.api.aws	HTTPS HTTPS
Südamerika (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com rds.sa-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (US-Ost)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com rds.us-gov-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com rds.us-gov-west-1.api.aws	HTTPS HTTPS

Die folgende Tabelle zeigt das Format, das Sie beim Erstellen eines ARN für einen bestimmten Amazon-RDS-Ressourcentyp verwenden sollten.

Ressourcentyp	ARN-Format
DB-Instance	arn:aws:rds: ::db: *<region>* *<account>* *<name>* Zum Beispiel:

Ressourcentyp	ARN-Format
	`arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1`
DB-Cluster	arn:aws:rds: *<region>* ::cluster: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :cluster: my-aurora-cluster-1`
Ereignisabonnement	arn:aws:rds: *<region>* ::es: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription`
DB-Parametergruppe	arn:aws:rds: *<region>* ::pg: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs`
DB-Cluster-Parametergruppe	arn:aws:rds: *<region>* :cluster-pg: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone`

Ressourcentyp	ARN-Format
Reservierte DB-Instance	arn:aws:rds: *<region>* ::ri: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :ri:my-reserved-postgresql`
DB-Sicherheitsgruppe	arn:aws:rds: *<region>* ::secgrp: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :secgrp:my-public`
Automatisierter DB-Snapshot	arn:aws:rds: *<region>* :snapshot:rds: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23`
Automatisierter DB-Cluster-Snapshot	arn:aws:rds: *<region>* *<account>* :cluster-snapshot:rds: *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds: my-aurora-cluster-2019-07-22-16-16`
Manueller DB-Snapshot	arn:aws:rds: *<region>* ::snapshot: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap`

Ressourcentyp	ARN-Format
Manueller DB-Cluster-Snapshot	arn:aws:rds: *<region>* :cluster-snapshot: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-aurora-cluster-snap`
DB-Subnetzgruppe	arn:aws:rds: *<region>* ::subgrp: *<account>* *<name>* Zum Beispiel: `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`
Globaler Cluster	arn:aws:rds: ::global-cluster: *<account>* *<name>* Zum Beispiel: `arn:aws:rds:: 123456789012 :global-cluster: my-aurora-global-cluster-1`

Abrufen eines vorhandenen ARN für Amazon RDS

Sie können den ARN für eine RDS-Ressource über die AWS-Managementkonsole, AWS Command Line Interface (AWS CLI) oder RDS-API abrufen.

Konsole

Um einen ARN aus der AWS-Managementkonsole abzurufen, navigieren Sie zu der Ressource, für die Sie einen ARN erhalten möchten, und zeigen Sie die Details für diese Ressource an.

Sie können den ARN für einen DB-Cluster beispielsweise auf der Seite Konfiguration der DB-Cluster-Details abrufen.
DB-Cluster-ARN.

AWS CLI

Um einen ARN aus der AWS CLI für eine bestimmte RDS-Ressource abzurufen, verwenden Sie den Befehl `describe` für diese Ressource. Die folgende Tabelle zeigt die einzelnen AWS CLI-ARN-Befehle sowie die ARN-Eigenschaft, die mit dem Befehl verwendet wird, um einen ARN abzurufen.

AWS CLI command	ARN-Eigenschaft
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DBParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn
describe-db-instances	DBInstanceArn
describe-db-security-groups	DBSecurityGroupArn
describe-db-snapshots	DBSnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDBInstanceArn
describe-db-subnet-groups	DBSubnetGroupArn
describe-db-clusters	DBClusterArn
describe-db-cluster-snapshots	DBClusterSnapshotArn

Der folgende AWS CLI-Befehl ruft die ARN für eine DB-Instance ab.

Example

Für Linux, macOS oder Unix:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
```

```
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Für Windows:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Die Ausgabe dieses Befehls sieht wie folgt aus:

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
    }
]
```

RDS-API

Um einen ARN für eine bestimmte RDS-Ressource abzurufen, können Sie die folgenden RDS-API-Operationen aufrufen und die folgenden ARN-Eigenschaften verwenden.

RDS-API-Operation	ARN-Eigenschaft
DescribeEventSubscriptions	EventSubscriptionArn
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn

RDS-API-Operation	ARN-Eigenschaft
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Amazon-Aurora-Updates

Amazon Aurora veröffentlicht regelmäßig Updates. Aktualisierungen werden auf das Amazon-Aurora-DB-Cluster während des Wartungszeitraums angewendet. Zu welchem Zeitpunkt Updates angewandt werden, ist von den Einstellungen für die Region und den Wartungszeitraum für den DB-Cluster und auch vom Typ des Updates abhängig. Aktualisierungen erfordern einen Neustart der Datenbank, daher kommt es zu einem Ausfall von 20 bis 30 Sekunden. Danach können Sie wieder mit der Nutzung Ihres DB-Clusters oder der Cluster fortfahren. Sie können die Einstellungen für den Wartungszeitraum in der anzeigen und ändern AWS-Managementkonsole.

> **Note**
>
> Die Zeit, die zum Neustart Ihrer DB-Instance benötigt wird, hängt vom Absturz-Wiederherstellungsprozess, der Datenbankaktivität zum Zeitpunkt des Neustarts und dem Verhalten Ihrer spezifischen DB-Engine ab. Wir empfehlen, während eines Neustarts die Datenbankaktivitäten möglichst zu minimieren, um die Dauer des Neustarts zu verkürzen. Die Reduzierung der Datenbankaktivität reduziert die Rollback-Aktivität für übertragene Transaktionen.

Informationen zu Betriebssystemupdates für Amazon Aurora finden Sie unter Betriebssystemupdates für Aurora-DB-Cluster.

Einige Updates sind für eine Datenbank-Engine spezifisch, die von Aurora unterstützt wird. Weitere Informationen über Datenbank-Engine-Updates finden Sie in der folgenden Tabelle.

Datenbank-Engine	Aktualisierungen
Amazon Aurora MySQL	Siehe Datenbank-Engine-Updates für Amazon Aurora MySQL
Amazon Aurora PostgreSQL	Siehe Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL

Identifizieren Ihrer Amazon-Aurora-Version

Amazon Aurora umfasst bestimmte Funktionen, die allgemein für Aurora und für alle Aurora-DB-Cluster verfügbar sind. Amazon Aurora umfasst andere Funktionen, die spezifisch für eine bestimmte

Datenbank-Engine sind, die Aurora unterstützt. Diese Funktionen stehen nur denjenigen Aurora-DB-Clustern zur Verfügung, die diese Datenbank-Engine verwenden, wie z. B. Aurora PostgreSQL.

Eine Aurora-DB-Instance hat zwei Versionsnummern, die Aurora-Versionsnummer und die Versionsnummer der Aurora-Datenbankengine. Aurora-Versionsnummern verwenden das folgende Format.

```
<major version>.<minor version>.<patch version>
```

Um die Aurora-Versionsnummer aus einer Aurora-DB-Instance abzurufen, die eine bestimmte Datenbank-Engine verwendet, verwenden Sie eine der folgenden Abfragen.

Datenbank-Engine	Abfragen
Amazon Aurora MySQL	`SELECT AURORA_VERSION();` `SHOW @@aurora_version;`
Amazon Aurora PostgreSQL	`SELECT AURORA_VERSION();`

Amazon RDS Extended Support mit Amazon Aurora

RDS Extended Support ermöglicht es Ihnen, gegen einen Aufpreis eine Datenbank weiterhin auf einer Engine-Hauptversion auszuführen, auch wenn der Aurora-Standard-Support abgelaufen ist.

Sie können eine Datenbank nur in RDS Extended Support registrieren, wenn Sie beim ersten Erstellen oder Wiederherstellen einer DB-Instance „RDS Extended Support“ aktivieren. Sie können den Registrierungsstatus bestehender DB-Instances in RDS Extended Support nicht ändern, es sei denn, Sie stellen sie wieder her.

Wenn Sie RDS Extended Support während der Erstellung oder Wiederherstellung einer DB-Instance aktiviert haben, registriert Amazon Aurora diese DB-Instance nach Ablauf des Aurora-Standard-Supports automatisch in RDS Extended Support. Die automatische Registrierung in RDS Extended Support ändert nichts an der Datenbank-Engine und beeinträchtigt nicht die Verfügbarkeit oder Leistung Ihrer DB-Instance.

RDS Extended Support bietet die folgenden Updates und folgenden technischen Support:

- Sicherheitsupdates für kritische und hohe CVEs für Ihre DB-Instance oder Ihren DB-Cluster, einschließlich der Datenbank-Engine
- Bugfixes und Patches für kritische Probleme
- Die Möglichkeit, Supportanfragen zu stellen und im Rahmen des standardmäßigen Service Level Agreements von Amazon RDS Hilfe bei Bugfixes zu erhalten

Mit dieser kostenpflichtigen Option haben Sie mehr Zeit für ein Upgrade auf eine unterstützte Hauptversion der Engine. Das Enddatum des Standard-Supports für Aurora für Aurora MySQL Version 2 ist beispielsweise der 31. Oktober 2024. Sie sind jedoch nicht bereit, vor diesem Datum ein manuelles Upgrade auf Aurora MySQL Version 3 durchzuführen. In diesem Fall registriert Amazon Aurora Ihren Cluster am 31. Oktober 2024 automatisch in RDS Extended Support und Sie können Aurora MySQL Version 2 weiterhin ausführen. Ab dem 1. Dezember 2024 berechnet Ihnen Amazon Aurora dann automatisch Gebühren für RDS Extended Support.

RDS Extended Support ist bis zu 3 Jahre nach dem Datum des Lebensendes der Community für eine Engine-Hauptversion (3 Jahre und 4 Monate für Aurora MySQL Version 2) verfügbar. Wenn Sie Ihre Engine-Hauptversion nach Ablauf dieses Zeitraums nicht auf eine unterstützte Version aktualisiert haben, aktualisiert Amazon Aurora Ihre Engine-Hauptversion automatisch. Wir empfehlen, so schnell wie möglich auf eine unterstützte Haupt-Engine-Version zu aktualisieren.

Weitere Informationen zum Ende des Aurora-Standard-Supports und des erweiterten RDS-Supports finden Sie im Veröffentlichungskalender für Aurora-MySQL-Hauptversionen und im Veröffentlichungskalender für Aurora-PostgreSQL-Hauptversionen.

Themen

- Amazon RDS Extended Support im Überblick
- Gebühren für Amazon RDS Extended Support
- Versionen mit Amazon RDS Extended Support
- Aufgaben von Amazon Aurora und von Kunden hinsichtlich Amazon RDS Extended Support
- Erstellen eines Aurora-DB-Clusters oder eines globalen Clusters mit Amazon RDS Extended Support
- Anzeigen der Registrierung Ihrer Aurora-DB-Cluster oder globalen Cluster in Amazon RDS Extended Support
- Anzeigen der Support-Daten für Engine-Versionen in Amazon RDS Extended Support
- Wiederherstellen eines Aurora-DB-Clusters oder eines globalen Clusters mit Amazon RDS Extended Support

Amazon RDS Extended Support im Überblick

Nach Ablauf des Aurora-Standard-Supports gilt Folgendes: Wenn Sie beim Erstellen oder Wiederherstellen Ihrer DB-Instances RDS Extended Support nicht deaktiviert haben, registriert Amazon Aurora diese automatisch in RDS Extended Support. Aurora aktualisiert Ihre DB-Instance automatisch auf die letzte Nebenversion, die vor Ablauf des Aurora-Standard-Supports veröffentlicht wurde, sofern Sie diese Version nicht bereits ausführen. Amazon Aurora aktualisiert Ihre Nebenversion erst nach Ablauf des Aurora-Standard-Supports für Ihre Engine-Hauptversion.

Sie können neue Datenbanken mit Engine-Hauptversionen erstellen, die das Ende des Aurora-Standard-Supports erreicht haben. Aurora registriert diese neuen Datenbanken automatisch in RDS Extended Support und berechnet Ihnen für dieses Angebot Gebühren.

Wenn Sie vor Ablauf des Aurora-Standard-Supports ein Upgrade auf eine Engine durchführen, für die der Aurora-Standard-Support noch gilt, registriert Amazon Aurora Ihre Engine nicht in RDS Extended Support.

Wenn Sie versuchen, einen Snapshot einer Datenbank wiederherzustellen, die mit einer Engine kompatibel ist, für die der Aurora-Standard-Support abgelaufen ist, die aber nicht in RDS Extended

Support registriert ist, versucht Amazon Aurora, den Snapshot so zu aktualisieren, dass er mit der neuesten Engine-Version kompatibel ist, für die der Aurora-Standard-Support noch gilt. Wenn die Wiederherstellung fehlschlägt, registriert Amazon Aurora Ihre Engine automatisch in RDS Extended Support – mit einer Version, die mit dem Snapshot kompatibel ist.

Sie können die Registrierung in RDS Extended Support jederzeit beenden. Um die Registrierung zu beenden, aktualisieren Sie jede registrierte Engine auf eine neuere Engine-Version, für die noch der Aurora-Standard-Support gilt. Die Registrierung in RDS Extended Support endet an dem Tag, an dem Sie ein Upgrade auf eine neuere Engine-Version ausführen, für die der Aurora-Standard-Support noch gilt.

Weitere Informationen zum Ende des Aurora-Standard-Supports und des erweiterten RDS-Supports finden Sie im Veröffentlichungskalender für Aurora-MySQL-Hauptversionen und im Veröffentlichungskalender für Aurora-PostgreSQL-Hauptversionen.

Gebühren für Amazon RDS Extended Support

Ab dem Tag nach Ablauf des Standardsupports für Aurora fallen Gebühren für alle Engines an, die bei RDS Extended Support registriert sind. Informationen zum Datum, an dem der Standardsupport für Aurora endet, finden Sie unter Amazon-Aurora-Hauptversionen.

Die zusätzliche Gebühr für RDS Extended Support endet automatisch, wenn Sie eine der folgenden Maßnahmen ergreifen:

- Durchführen eines Upgrades auf eine Engine-Version, die vom Standardsupport abgedeckt ist.
- Löschen der Datenbank, auf der eine Hauptversion nach Ablauf des Standardsupports für Aurora ausgeführt wird.

Die Gebühren werden wieder erhoben, wenn die Version der Ziel-Engine zukünftig in den RDS Extended Support aufgenommen wird.

Zum Beispiel wird 11 am 1. März 2024 in den Extended Support aufgenommen, die Gebühren werden jedoch erst ab 1. April 2024 erhoben. Am 30. April 2024 führen Sie ein Upgrade der Datenbank von Aurora PostgreSQL 11 auf 12 durch. Ihnen werden nur 30 Tage für den Extended Support von Aurora PostgreSQL 11 in Rechnung gestellt. Sie führen Aurora PostgreSQL 12 weiterhin auf dieser DB-Instance aus, nachdem der RDS-Standardsupport am 28. Februar 2025 abgelaufen ist. Für Ihre Datenbank fallen ab dem 1. März 2025 wieder Gebühren für RDS Extended Support an.

Weitere Informationen finden Sie unter Preise für Amazon Aurora.

Vermeiden von Gebühren für Amazon RDS Extended Support

Sie können Gebühren für RDS Extended Support umgehen, indem Sie verhindern, dass Aurora nach dem Ablauf des Aurora-Standardsupports einen Aurora-DB-Cluster oder einen globalen Cluster erstellt oder wiederherstellt. Sie können dazu die AWS CLI oder die RDS-API verwenden.

Geben Sie in der AWS CLI `open-source-rds-extended-support-disabled` für die Option `--engine-lifecycle-support` an. Geben Sie in der RDS-API `open-source-rds-extended-support-disabled` für den Parameter `LifeCycleSupport` an. Weitere Informationen finden Sie unter Erstellen eines Aurora-DB-Clusters oder eines globalen Clusters oder Wiederherstellen eines Aurora-DB-Clusters oder eines globalen Clusters.

Versionen mit Amazon RDS Extended Support

RDS Extended Support ist für Aurora MySQL und für Aurora PostgreSQL verfügbar. Weitere Informationen finden Sie unter Amazon-Aurora-Hauptversionen.

RDS Extended Support ist nur für bestimmte Nebenversionen verfügbar. Nebenversionen unterstützen RDS Extended Support gemäß Angabe erst, nachdem Hauptversionen das Community-Ablaufdatum erreicht haben. Weitere Informationen finden Sie im Veröffentlichungskalender für Aurora MySQL in den Versionshinweisen zu Aurora MySQL und im Veröffentlichungskalender für Aurora PostgreSQL in den Versionshinweisen zu Aurora PostgreSQL.

RDS Extended Support ist nur für Aurora Serverless v2 verfügbar. Es ist nicht für Aurora Serverless v1 verfügbar.

Mithilfe der AWS CLI oder der RDS-API können Sie auch Informationen zu den Support-Terminen für die Engine-Versionen einsehen. Weitere Informationen finden Sie unter Anzeigen der Support-Daten für Engine-Versionen in Amazon RDS Extended Support.

Aufgaben von Amazon Aurora und von Kunden hinsichtlich Amazon RDS Extended Support

Im folgenden Inhalt wird beschrieben, welche Aufgaben hinsichtlich RDS Extended Support Amazon Aurora hat, und welche Sie haben.

Themen

- Aufgaben von Amazon Aurora
- Ihre Aufgaben

Aufgaben von Amazon Aurora

Nach Ablauf des Aurora-Standard-Supports stellt Amazon Aurora Patches, Bugfixes und Upgrades für Engines bereit, die in RDS Extended Support registriert sind. Sie gelten bis zu 3 Jahre oder bis Sie die Engines nicht mehr verwenden – je nachdem, was zuerst eintritt.

Die Patches werden für kritische und hohe CVEs bereitgestellt, wie in den CVSS-Schweregraden der National Vulnerability Database (NVD) definiert. Weitere Informationen finden Sie unter Kennzahlen zu Schwachstellen.

Ihre Aufgaben

Sie sind verantwortlich für die Installation der Patches, Bugfixes und Upgrades, die für Aurora DB-Custer oder globale Cluster bereitgestellt werden, die in RDS Extended Support registriert sind. Amazon Aurora behält sich das Recht vor, solche Patches, Bugfixes und Upgrades jederzeit zu ändern, zu ersetzen oder zurückzuziehen. Wenn ein Patch erforderlich ist, um Sicherheits- oder kritische Stabilitätsprobleme zu beheben, behält sich Amazon Aurora das Recht vor, Ihre Aurora-DB-Cluster oder globalen Cluster anhand des Patches zu aktualisieren oder Sie aufzufordern, den Patch selbst zu installieren.

Sie sind auch dafür verantwortlich, Ihre Engine vor Ablauf von RDS Extended Support auf eine neuere Engine-Version zu aktualisieren. Der erweiterte Support für RDS endet in der Regel 3 Jahre nach dem Ende des Lebenszyklus der Community. . Wann der erweiterte RDS-Support für die Engine-Hauptversion Ihrer Datenbank endet, erfahren Sie unter Amazon-Aurora-Hauptversionen.

Wenn Sie Ihre Engine nicht aktualisieren, versucht Amazon Aurora nach Ablauf von RDS Extended Support, Ihre Engine auf eine neuere Engine-Version zu aktualisieren, die vom Aurora-Standard-Support unterstützt wird. Schlägt das Upgrade fehl, behält sich Amazon Aurora das Recht vor, den Aurora-DB-Cluster oder den globalen Cluster, auf dem(der) die Engine läuft, nach Ablauf des Aurora-Standard-Supports zu löschen. Vorher speichert Amazon Aurora jedoch Ihre Daten aus dieser Engine.

Erstellen eines Aurora-DB-Clusters oder eines globalen Clusters mit Amazon RDS Extended Support

Wenn Sie einen Aurora-DB-Cluster oder einen globalen Cluster erstellen, wählen Sie in der Konsole die Option Erweiterte RDS-Unterstützung aktivieren aus oder verwenden Sie die Option „Erweiterte Unterstützung“ in der AWS CLI oder den Parameter in der RDS-API. Wenn Sie einen Aurora-DB-Cluster oder einen globalen Cluster bei Amazon RDS Extended Support registrieren, ist diese(r) für die gesamte Lebensdauer des Aurora-DB-Clusters oder des globalen Clusters dauerhaft im RDS Extended Support registriert.

Wenn Sie die Konsole verwenden, müssen Sie Erweiterte RDS-Unterstützung aktivieren auswählen. Die Einstellung ist standardmäßig nicht ausgewählt.

Wenn Sie die AWS CLI oder die RDS-API verwenden und die Einstellung für RDS Extended Support nicht festlegen, aktiviert Amazon RDS den RDS Extended Support standardmäßig. Wenn Sie den Schritt mithilfe von CloudFormation oder anderen Diensten automatisieren, sorgt dieses Standardverhalten dafür, dass Ihre Datenbank auch nach dem Ende des Standard-Supports von Aurora verfügbar bleibt.

Sie können die Registrierung bei RDS Extended Support verhindern, indem Sie die AWS CLI oder die RDS-API verwenden, um einen Aurora-DB-Cluster oder einen globalen Cluster zu erstellen.

Themen

- Verhalten von RDS Extended Support
- Überlegungen zu RDS Extended Support
- Erstellen Sie einen Aurora-DB-Cluster oder einen globalen Cluster mit RDS Extended Support

Verhalten von RDS Extended Support

In der folgenden Tabelle wird zusammengefasst, was passiert, wenn eine Engine-Hauptversion das Ende der Standardunterstützung für Aurora erreicht.

Status von RDS Extended Support*	Behavior
Aktiviert	Amazon RDS stellt Ihnen RDS Extended Support in Rechnung.

Status von RDS Extended Support*	Behavior
Disabled	Amazon RDS aktualisiert Ihren Aurora-DB-Cluster oder Ihren globalen Cluster auf eine unterstützte Engine-Version. Dieses Upgrade findet am oder kurz nach dem Ende des Standard-Supports für Aurora statt.

* In der RDS-Konsole wird der Status von RDS Extended Support als „Ja“ oder „Nein“ angezeigt. In der AWS CLI oder RDS-API wird der Status von RDS Extended Support als `open-source-rds-extended-support` oder `open-source-rds-extended-support-disabled` angezeigt.

Überlegungen zu RDS Extended Support

Bevor Sie einen Aurora-DB-Cluster oder einen globalen Cluster erstellen, sollten Sie die folgenden Punkte berücksichtigen:

- Nach Ablauf des Standard-Supports für Aurora können Sie die Erstellung eines neuen Aurora-DB-Clusters oder eines neuen globalen Clusters verhindern und so die Gebühren für RDS Extended Support vermeiden. Sie können dazu die AWS CLI oder die RDS-API verwenden. Geben Sie in der AWS CLI `open-source-rds-extended-support-disabled` für die Option `--engine-lifecycle-support` an. Geben Sie in der RDS-API `open-source-rds-extended-support-disabled` für den Parameter `LifeCycleSupport` an. Wenn Sie `open-source-rds-extended-support-disabled` festlegen und der Standard-Supports für Aurora abgelaufen ist, schlägt die Erstellung eines Aurora-DB-Clusters oder eines globalen Clusters bei jedem Versuch fehl.
- RDS Extended Support wird auf Cluster-Ebene festgelegt. Mitglieder eines Clusters haben immer dieselbe Einstellung für RDS Extended Support in der RDS-Konsole, `--engine-lifecycle-support` in der AWS CLI und `EngineLifecycleSupport` in der RDS-API.

Weitere Informationen finden Sie unter Amazon-Aurora-Versionen.

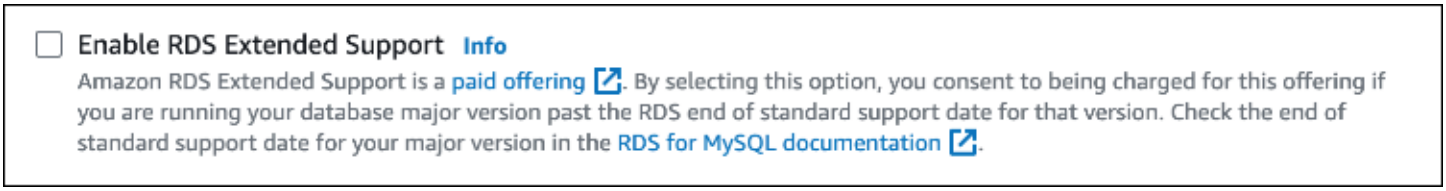
Erstellen Sie einen Aurora-DB-Cluster oder einen globalen Cluster mit RDS Extended Support

Sie können einen Aurora-DB-Cluster oder einen globalen Cluster mit einer Version von RDS Extended Support erstellen, indem Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API verwenden.

Konsole

Wenn Sie einen Aurora-DB-Cluster oder einen globalen Cluster erstellen, wählen Sie im Abschnitt Engine-Optionen die Option Erweiterte RDS-Unterstützung aktivieren aus. Diese Einstellung ist standardmäßig nicht ausgewählt.

Die folgende Abbildung zeigt die Einstellung Erweiterte RDS-Unterstützung aktivieren:



AWS CLI

Wenn Sie den AWS CLI-Befehl create-db-cluster oder create-global-cluster ausführen, wählen Sie „RDS Extended Support“ aus, indem Sie für die Option `--engine-lifecycle-support` `open-source-rds-extended-support` festlegen. Diese Option wird standardmäßig auf `open-source-rds-extended-support` festgelegt.

Um die Erstellung eines neuen Aurora-DB-Clusters oder eines neuen globalen Clusters nach dem Ende des Standard-Supports für Aurora zu verhindern, legen Sie für die Option `--engine-lifecycle-support` `open-source-rds-extended-support-disabled` fest. Auf diese Weise vermeiden Sie alle mit RDS Extended Support verbundenen Gebühren.

RDS-API

Wenn Sie die API-Operation CreateDBCluster oder CreateGlobalCluster von Amazon RDS verwenden, wählen Sie RDS Extended Support aus, indem Sie den Parameter `EngineLifecycleSupport` auf `open-source-rds-extended-support` festlegen. Dieser Parameter ist standardmäßig auf `open-source-rds-extended-support` festgelegt.

Um die Erstellung eines neuen Aurora-DB-Clusters oder eines neuen globalen Clusters nach dem Ende des Standard-Supports für Aurora zu verhindern, legen Sie für den Parameter

`EngineLifecycleSupport` `open-source-rds-extended-support-disabled` fest. Auf diese Weise vermeiden Sie alle mit RDS Extended Support verbundenen Gebühren.

Weitere Informationen finden Sie unter den folgenden Themen:

- Befolgen Sie die Anleitungen für Ihre DB-Engine unter Erstellen eines Amazon Aurora-DB Clusters, um einen Aurora-DB-Cluster zu erstellen.
- Befolgen Sie die Anleitungen für Ihre DB-Engine unter Erstellen einer globalen Datenbank von Amazon Aurora, um einen globalen Cluster zu erstellen.

Anzeigen der Registrierung Ihrer Aurora-DB-Cluster oder globalen Cluster in Amazon RDS Extended Support

Sie können die Registrierung Ihrer Aurora-DB-Cluster oder globalen Cluster in RDS Extended Support über die AWS-Managementkonsole, die AWS CLI oder RDS-API anzeigen.

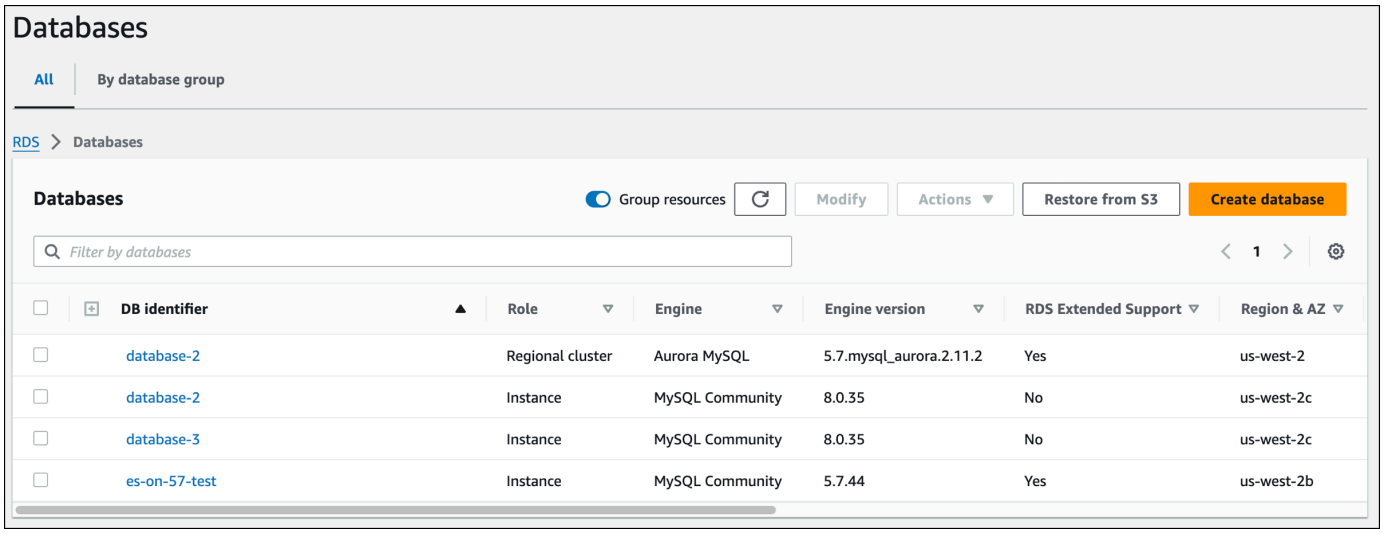
> **Note**
>
> Die Spalte RDS Extended Support in der Konsole, die Option `-engine-lifecycle-support` in der AWS CLI und der Parameter `EngineLifecycleSupport` in der RDS-API geben nur die Registrierung in RDS Extended Support an. Für RDS Extended Support fallen erst dann Gebühren an, wenn Aurora-Standard-Support für Ihre DB-Engine-Version abgelaufen ist. Weitere Informationen finden Sie unter Amazon-Aurora-Hauptversionen. Sie haben beispielsweise eine Datenbank des Typs Aurora PostgreSQL 11, die in RDS Extended Support registriert ist. Am 1. April 2024 begann Amazon RDS, Ihnen RDS Extended Support für diese Datenbank in Rechnung zu stellen. Am 31. Juli 2024 haben Sie diese Datenbank auf Aurora PostgreSQL 12 aktualisiert. Der Status von RDS Extended Support bleibt für diese Datenbank aktiviert. Für RDS Extended Support fallen jedoch keine Gebühren für diese Datenbank an, da der Aurora-Standard-Support für Aurora PostgreSQL 12 noch nicht abgelaufen ist. Amazon RDS berechnet Ihnen RDS Extended Support für diese Datenbank erst am 1. März 2026, wenn Aurora-Standard-Support für Aurora PostgreSQL 12 endet.

Konsole

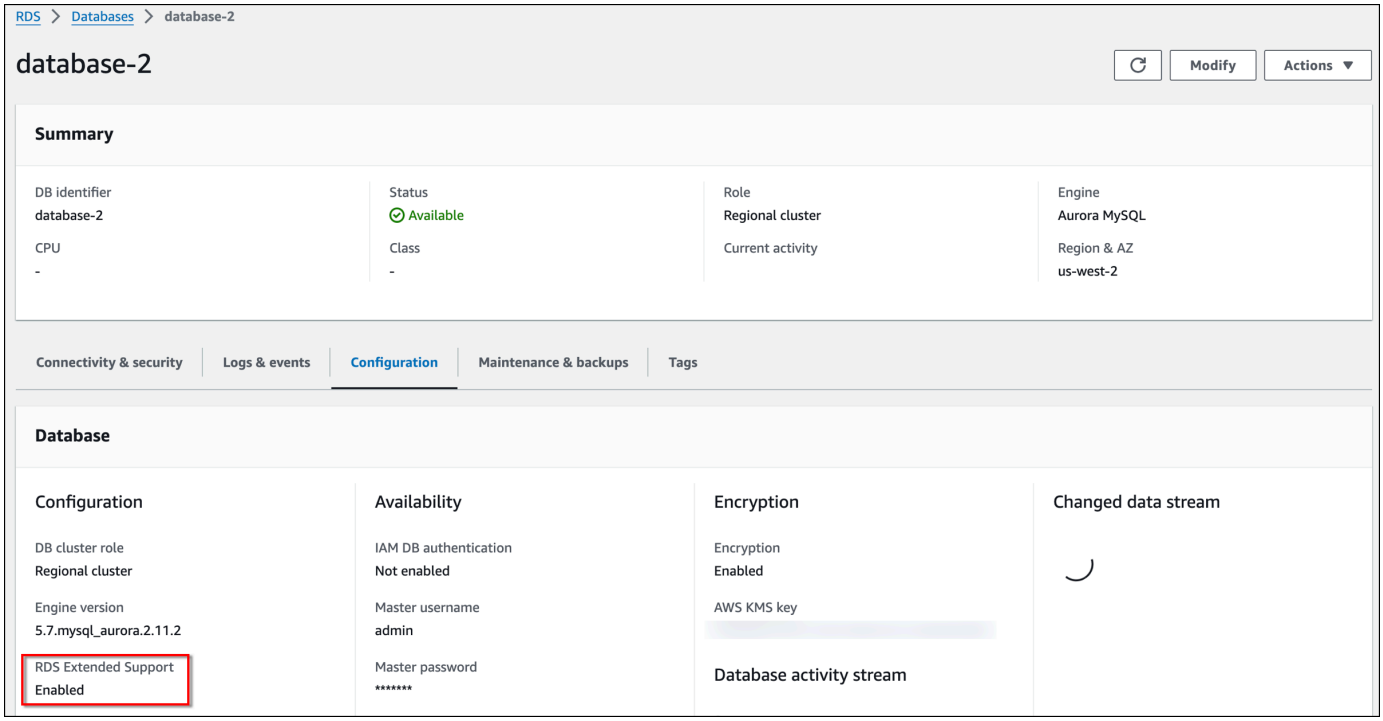
So zeigen Sie die Registrierung Ihrer Aurora-DB-Cluster oder globalen Cluster in Amazon RDS Extended Support an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus. Der Wert unter RDS Extended Support gibt an, ob ein Aurora-DB-Cluster oder ein globaler Cluster in RDS Extended Support registriert ist. Wenn kein Wert angezeigt wird, ist RDS Extended Support für Ihre Datenbank nicht verfügbar.

> **Tip**
>
> Wenn die Spalte RDS Extended Support nicht angezeigt wird, wählen Sie das Symbol Einstellungen und aktivieren Sie dann die Option RDS Extended Support.



3. Sie können die Registrierung auch auf der Registerkarte Konfiguration jeder Datenbank einsehen. Wählen Sie unter DB-Kennung eine Datenbank aus. Sehen Sie auf der Registerkarte Konfiguration unter Erweiterter Support nach, ob die Datenbank registriert ist oder nicht.



AWS CLI

Führen Sie den Befehl describe-db-clusters oder describe-global-clusters aus, um Informationen zur Registrierung Ihrer Datenbanken in RDS Extended Support über die AWS CLI anzeigen zu lassen.

Wenn RDS Extended Support für eine Datenbank verfügbar ist, enthält die Antwort den Parameter `EngineLifecycleSupport`. Der Wert `open-source-rds-extended-support` gibt an, dass ein Aurora-DB-Cluster oder ein globaler Cluster in RDS Extended Support registriert ist. Der Wert `open-source-rds-extended-support-disabled` gibt an, dass die Registrierung des Aurora-DB-Clusters oder des globalen Clusters in RDS Extended Support aufgehoben wurde.

Beispiel

Der folgende Befehl gibt Informationen für alle Ihre Aurora-DB-Cluster zurück:

```
aws rds describe-db-clusters
```

Die folgende Antwort zeigt, dass eine auf dem Aurora-DB-Cluster `database-1` ausgeführte Aurora-PostgreSQL-Engine in RDS Extended Support registriert ist:

```
{
    "DBClusterIdentifier": "database-1",
```

```
    ...
    "Engine": "aurora-postgresql",
    ...
    "EngineLifecycleSupport": "open-source-rds-extended-support"
}
```

RDS-API

Verwenden Sie die Operation DescribeDBClusters oder DescribeGlobalClusters, wenn Sie Informationen zur Registrierung Ihrer Datenbanken in RDS Extended Support über die Amazon-RDS-API anzeigen lassen möchten.

Wenn RDS Extended Support für eine Datenbank verfügbar ist, enthält die Antwort den Parameter `EngineLifecycleSupport`. Der Wert `open-source-rds-extended-support` gibt an, dass ein Aurora-DB-Cluster oder ein globaler Cluster in RDS Extended Support registriert ist. Der Wert `open-source-rds-extended-support-disabled` gibt an, dass die Registrierung des Aurora-DB-Clusters oder des globalen Clusters in RDS Extended Support aufgehoben wurde.

Anzeigen der Support-Daten für Engine-Versionen in Amazon RDS Extended Support

Sie können über die AWS CLI oder die RDS-API Informationen zu Support-Daten für Engine-Versionen für Ihre Aurora-DB-Cluster oder globalen Cluster in Amazon RDS Extended Support anzeigen. Diese Informationen können Ihnen bei der Planung von Upgrades helfen.

AWS CLI-Befehle und RDS-API-Operationen geben Start- und Enddaten zu Standard-Support und RDS Extended Support für Aurora zurück. Diese Daten finden Sie auch in den Tabellen zu den Engine-Hauptversionen. Weitere Informationen finden Sie unter Amazon-Aurora-Hauptversionen.

AWS CLI

Führen Sie den Befehl describe-db-major-engine-versions aus, um über die AWS CLI für Ihre Engine-Hauptversionen die Start- und Enddaten zu Standard-Support und RDS Extended Support für Aurora anzuzeigen.

Dieser Befehl gibt die folgenden relevanten Parameter zurück:

- `SupportedEngineLifecycles`: Dieser Parameter entspricht einer Reihe von Objekten, zu denen `LifecycleSupportName`, `LifecycleSupportStartDate` und `LifecycleSupportEndDate` gehören.

- `LifecycleSupportName`: Dieser Parameter gibt die Art des Supports an, die für die Engine-Version gilt: Aurora Standard-Support (`open-source-rds-standard-support`) oder RDS Extended Support (`open-source-rds-extended-support`).
- `LifecycleSupportStartDate`: Dieser Parameter listet das Startdatum von Aurora Standard-Support oder RDS Extended Support für die Engine-Hauptversion auf, abhängig vom Wert `LifecycleSupportName`.
- `LifecycleSupportEndDate`: Dieser Parameter listet das Enddatum von Aurora Standard-Support oder RDS Extended Support für die Engine-Hauptversion auf, abhängig vom Wert `LifecycleSupportName`.

Beispiel

Das Antwortbeispiel zeigt die Start- und Enddaten für die unterstützten Engine-Lebenszyklen `open-source-rds-standard-support` und `open-source-rds-extended-support` für Aurora MySQL Version 2 (MySQL 5.7). RDS Extended Support ist für Aurora MySQL Version 2 (MySQL 5.7) verfügbar.

```
{
    "DBMajorEngineVersions": [
        {
            "Engine": "aurora-mysql",
            "MajorEngineVersion": "5.7",
            "SupportedEngineLifecycles": [
                {
                    "LifecycleSupportName": "open-source-rds-standard-support",
                    "LifecycleSupportStartDate": "2018-02-06T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2024-10-31T23:59:59.999000+00:00"
                },
                {
                    "LifecycleSupportName": "open-source-rds-extended-support",
                    "LifecycleSupportStartDate": "2024-11-01T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2027-02-28T23:59:59.999000+00:00"
                }
            ]
        },
        ...
    ]
}
```

RDS-API

Verwenden Sie die Operation DescribeDBMajorEngineVersions, um über die RDS-API für Ihre Engine-Hauptversionen die Start- und Enddaten von Standard-Support und RDS Extended Support für Aurora anzuzeigen.

Wenn RDS Extended Support für eine Engine-Version verfügbar ist, enthält die Antwort den Parameter `SupportedEngineLifeCycles` als Array mit zwei Objekten. Ein Objekt enthält die Start- und Enddaten von Standard-Support für Aurora. Das zweite Objekt enthält die Start- und Enddaten von RDS Extended Support.

Wenn RDS Extended Support für eine Engine-Version nicht verfügbar ist, enthält die Antwort den Parameter `SupportedEngineLifeCycles` als Array mit einem einzigen Objekt. Dieses Objekt enthält die Start- und Enddaten von Aurora-Standard-Support.

Wiederherstellen eines Aurora-DB-Clusters oder eines globalen Clusters mit Amazon RDS Extended Support

Wenn Sie einen Aurora-DB-Cluster oder einen globalen Cluster wiederherstellen, wählen Sie in der Konsole die Option Erweiterte RDS-Unterstützung aktivieren aus oder verwenden Sie die Option „Erweiterte Unterstützung“ in der AWS CLI oder den Parameter in der RDS-API. Wenn Sie einen Aurora-DB-Cluster oder einen globalen Cluster bei RDS Extended Support registrieren, ist diese(r) für die gesamte Lebensdauer des Aurora-DB-Clusters oder des globalen Clusters dauerhaft im RDS Extended Support registriert.

Die Standardeinstellung für RDS Extended Support hängt davon ab, ob Sie zum Wiederherstellen Ihrer Datenbank die Konsole, die AWS CLI oder die RDS-API verwenden. Wenn Sie die Konsole verwenden, die Option Erweiterte RDS-Unterstützung aktivieren nicht auswählen und bei der Engine-Hauptversion, die Sie wiederherstellen, der Standard-Support für Aurora abgelaufen ist, aktualisiert Amazon Aurora Ihre DB-Instance automatisch auf eine neuere Engine-Version. Wenn Sie die AWS CLI oder die RDS-API verwenden und die Einstellung für RDS Extended Support nicht festlegen, aktiviert Amazon RDS den RDS Extended Support standardmäßig. Wenn Sie den Schritt mithilfe von CloudFormation oder anderen Diensten automatisieren, sorgt dieses Standardverhalten dafür, dass Ihre Datenbank auch nach dem Ende des Standard-Supports von Aurora verfügbar bleibt. Sie können RDS Extended Support über die AWS CLI oder die RDS-API deaktivieren.

Themen

- Verhalten von RDS Extended Support

- Überlegungen zu RDS Extended Support
- Stellen Sie einen Aurora-DB-Cluster oder einen globalen Cluster mit RDS Extended Support wieder her

Verhalten von RDS Extended Support

In der folgenden Tabelle wird zusammengefasst, was passiert, wenn die Engine-Hauptversion eines Aurora-DB-Clusters oder eines globalen Clusters, den(die) Sie wiederherstellen, das Ende des Standard-Supports für Aurora erreicht hat.

Status von RDS Extended Support*	Behavior
Aktiviert	Amazon RDS stellt Ihnen RDS Extended Support in Rechnung.
Disabled	Nach Abschluss der Wiederherstellung aktualisiert Amazon RDS Ihre Ihren Aurora-DB-Cluster oder Ihren globalen Cluster automatisch auf eine neuere Engine-Version (in einem zukünftigen Wartungsfenster).

* In der RDS-Konsole wird der Status von RDS Extended Support als „Ja“ oder „Nein“ angezeigt. In der AWS CLI oder RDS-API wird der Status von RDS Extended Support als `open-source-rds-extended-support` oder `open-source-rds-extended-support-disabled` angezeigt.

Überlegungen zu RDS Extended Support

Bevor Sie einen Aurora-DB-Cluster oder einen globalen Cluster wiederherstellen, sollten Sie die folgenden Punkte berücksichtigen:

- Möchten Sie nach Ablauf des Standard-Supports für Aurora einen Aurora-DB-Cluster oder einen globalen Cluster von Amazon S3 aus wiederherstellen, gelingt Ihnen dies nur mit der AWS CLI oder der RDS-API. Verwenden Sie die Option `--engine-lifecycle-support` im AWS CLI-Befehl restore-db-cluster-from-s3 oder den Parameter `EngineLifecycleSupport` in der RDS-API-Operation RestoreDBClusterFromS3.
- Wenn Sie verhindern möchten, dass Aurora Ihre Datenbanken auf Versionen von RDS Extended Support zurücksetzt, legen Sie in der AWS CLI oder der RDS-API `open-source-rds-extended-support-disabled` fest. Auf diese Weise vermeiden Sie alle mit RDS Extended Support verbundenen Gebühren.

Wenn Sie diese Einstellung festlegen, aktualisiert Amazon Aurora Ihre wiederhergestellte Datenbank automatisch auf eine neuere, unterstützte Hauptversion. Wenn das Upgrade die Pre-Upgrade-Prüfungen nicht bestanden hat, führt Amazon Aurora ein sicheres Rollback zur Engine-Version von RDS Extended Support durch. Diese Datenbank bleibt im Modus „RDS Extended Support“ und Amazon Aurora stellt Ihnen RDS Extended Support in Rechnung, bis Sie Ihre Datenbank manuell aktualisieren.

- RDS Extended Support wird auf Cluster-Ebene festgelegt. Mitglieder eines Clusters haben immer dieselbe Einstellung für RDS Extended Support in der RDS-Konsole, `--engine-lifecycle-support` in der AWS CLI und `EngineLifecycleSupport` in der RDS-API.

Weitere Informationen finden Sie unter Amazon-Aurora-Versionen.

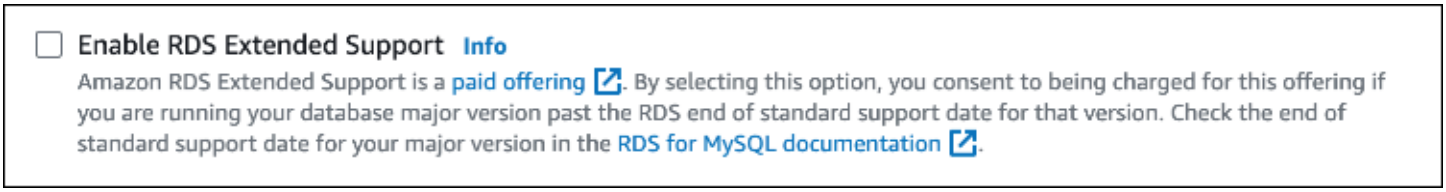
Stellen Sie einen Aurora-DB-Cluster oder einen globalen Cluster mit RDS Extended Support wieder her

Sie können einen Aurora-DB-Cluster oder einen globalen Cluster mit einer Version von RDS Extended Support wiederherstellen, indem Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API verwenden.

Konsole

Wenn Sie einen Aurora-DB-Cluster oder einen globalen Cluster wiederherstellen, wählen Sie im Abschnitt Engine-Optionen die Option Erweiterte RDS-Unterstützung aktivieren aus. Wenn Sie diese Einstellung nicht festlegen und bei der Engine-Hauptversion, die Sie wiederherstellen, der Standard-Support für Aurora abgelaufen ist, aktualisiert Amazon Aurora Ihren Aurora-DB-Cluster oder Ihren globalen Cluster automatisch auf eine Version mit Standard-Support für Aurora.

Die folgende Abbildung zeigt die Einstellung Erweiterte RDS-Unterstützung aktivieren:



AWS CLI

Wenn Sie den AWS CLI-Befehl restore-db-cluster-from-snapshot ausführen, wählen Sie RDS Extended Support aus, indem Sie für die Option `--engine-lifecycle-support` `open-source-rds-extended-support` angeben.

Wenn Sie Gebühren im Zusammenhang mit RDS Extended Support vermeiden möchten, setzen Sie die Option `--engine-lifecycle-support` auf `open-source-rds-extended-support-disabled`. Diese Option wird standardmäßig auf `open-source-rds-extended-support` festgelegt.

Sie können diesen Wert auch mit den folgenden AWS CLI-Befehlen angeben:

- restore-db-cluster-from-s3
- restore-db-cluster-to-point-in-time

RDS-API

Wenn Sie die API-Operation RestoreDBClusterFromSnapshot von Amazon RDS verwenden, wählen Sie RDS Extended Support aus, indem Sie den Parameter `EngineLifecycleSupport` auf `open-source-rds-extended-support` setzen.

Wenn Sie Gebühren im Zusammenhang mit RDS Extended Support vermeiden möchten, setzen Sie den Parameter `EngineLifecycleSupport` auf `open-source-rds-extended-support-disabled`. Dieser Parameter ist standardmäßig auf `open-source-rds-extended-support` festgelegt.

Sie können diesen Wert auch mittels der folgenden RDS-API-Operationen angeben:

- RestoreDBClusterFromS3
- RestoreDBClusterToPointInTime

Weitere Informationen zum Wiederherstellen eines Aurora-DB-Clusters finden Sie in den Anleitungen für Ihre DB-Engine unter Sichern und Wiederherstellen eines Amazon-Aurora-DB-Clusters.

Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates

Bei einer blue/green Bereitstellung wird eine Produktionsdatenbankumgebung in eine separate, synchronisierte Staging-Umgebung kopiert. Mithilfe von Amazon Aurora Blue/Green Deployments können Sie Änderungen an der Datenbank in der Staging-Umgebung vornehmen, ohne die Produktionsumgebung zu beeinträchtigen. Sie können beispielsweise die Haupt- oder Nebenversion der DB-Engine aktualisieren oder Datenbankparameter in der Staging-Umgebung ändern. Wenn Sie bereit sind, können Sie die Staging-Umgebung zur neuen Produktionsdatenbankumgebung hochstufen, wobei die Ausfallzeit in der Regel unter einer Minute liegt.

Amazon Aurora erstellt die Staging-Umgebung, indem das zugrunde liegende Aurora-Speichervolume in der Produktionsumgebung geklont wird. Das Cluster-Volume in der Staging-Umgebung speichert nur inkrementelle Änderungen, die in dieser Umgebung vorgenommen wurden.

> **Note**
>
> Derzeit werden Blue/Green Bereitstellungen nur für Aurora MySQL, Aurora PostgreSQL und Aurora Global Database. Informationen zur Verfügbarkeit der Amazon-RDS-Engine finden Sie im Amazon-RDS-Benutzerhandbuch unter Verwenden von Amazon-RDS-Blau/Grün-Bereitstellungen für Datenbankaktualisierungen.

Themen

- Überblick über Aurora Blue/Green Aurora-Bereitstellungen
- Eine blue/green Bereitstellung in — Amazon Aurora erstellen
- Anzeigen einer Blau/Grün-Bereitstellung in Amazon Aurora
- Wechseln einer blue/green Bereitstellung in Amazon Aurora
- Löschen einer Blau/Grün-Bereitstellung in Amazon Aurora

Überblick über Aurora Blue/Green Aurora-Bereitstellungen

Mithilfe von Amazon Aurora Blue/Green Deployments können Sie Datenbankänderungen vornehmen und testen, bevor Sie sie in einer Produktionsumgebung implementieren. Mit einer Blau/Grün-Bereitstellung wird eine Staging-Umgebung erstellt, die die Produktionsumgebung kopiert. In einer Blau/Grün-Umgebung ist die blaue Umgebung die aktuelle Produktionsumgebung. Die grüne Umgebung ist die Staging-Umgebung und bleibt mit der aktuellen Produktionsumgebung synchronisiert.

Sie können Änderungen dem Aurora-DB-Cluster in der grünen Umgebung vornehmen, ohne die Produktions-Workloads zu beeinträchtigen. Sie können beispielsweise die Haupt- oder Nebenversion der DB-Engine aktualisieren, oder Datenbankparameter in der Staging-Umgebung ändern. Sie können Änderungen in der grünen Umgebung gründlich testen. Wenn Sie bereit sind, können Sie die Umgebungen umstellen, um die grüne Umgebung zur neuen Produktionsumgebung zu machen. Die Umstellung dauert in der Regel weniger als eine Minute, ohne dass Daten verloren gehen und Anwendungsänderungen erforderlich sind.

Da die grüne Umgebung eine Kopie der Topologie der Produktionsumgebung ist, werden der DB-Cluster und alle seine DB-Instances in der Bereitstellung kopiert. Die grüne Umgebung umfasst auch die vom DB-Cluster verwendeten Funktionen wie DB-Cluster-Snapshots, Performance Insights, verbesserte Überwachung und Aurora Serverless v2.

> **Note**
>
> Blue/Green-Bereitstellungen werden für Aurora MySQL, Aurora PostgreSQL und Aurora Global Database unterstützt. Informationen zur Verfügbarkeit von Amazon RDS finden Sie unter Überblick über Amazon Blue/Green RDS-Bereitstellungen im Amazon RDS-Benutzerhandbuch.

Themen

- Verfügbarkeit von Regionen und Versionen
- Vorteile der Verwendung von Amazon Blue/Green RDS-Bereitstellungen
- Ablauf einer blue/green Bereitstellung
- Autorisieren des Zugriffs auf Aurora blue/green Aurora-Bereitstellungsvorgänge
- Einschränkungen und Überlegungen für Aurora blue/green Aurora-Bereitstellungen
- Bewährte Methoden für Aurora blue/green Aurora-Bereitstellungen

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen finden Sie unter the section called “Blau/Grün-Bereitstellungen”.

Vorteile der Verwendung von Amazon Blue/Green RDS-Bereitstellungen

Durch die Verwendung von Amazon RDS Blue/Green Deployments können Sie über Sicherheitspatches auf dem Laufenden bleiben, die Datenbankleistung verbessern und neuere Datenbankfunktionen mit kurzen, vorhersehbaren Ausfallzeiten einführen. Blue/green Bereitstellungen reduzieren die Risiken und Ausfallzeiten bei Datenbank-Updates, wie z. B. bei Upgrades größerer oder kleinerer Engine-Versionen.

Blau/Grün-Bereitstellungen bieten die folgenden Vorteile:

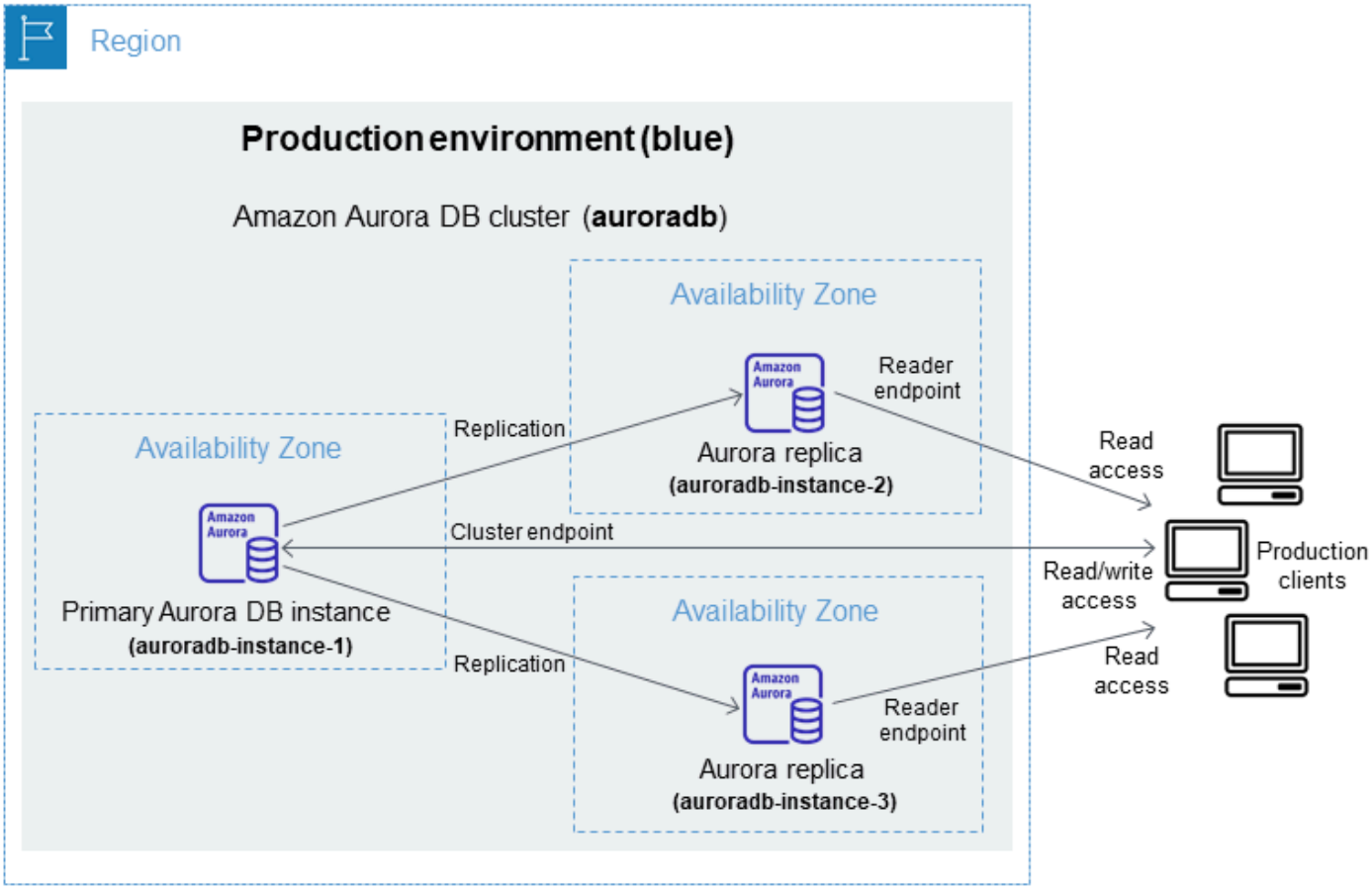
- Sie können ganz einfach eine produktionsreife Staging-Umgebung erstellen.
- Sie können Datenbankänderungen von der Produktionsumgebung in die Staging-Umgebung automatisch replizieren.
- Sie können Datenbankänderungen in einer sicheren Staging-Umgebung testen, ohne die Produktionsumgebung zu beeinträchtigen.
- Sie bleiben mit Datenbank-Patches und Systemaktualisierungen auf dem Laufenden.
- Sie können neue Datenbankfunktionen implementieren und testen.
- Sie können Ihre Staging-Umgebung auf die neue Produktionsumgebung umstellen, ohne Änderungen an Ihrer Anwendung vorzunehmen.
- Die Umstellung erfolgt durch den Einsatz des integrierten Integritätsschutzes völlig sicher.
- Es treten keine Datenverluste während der Umstellung auf.
- Die Umstellung erfolgt schnell, in der Regel in weniger als einer Minute, abhängig von Ihrer Workload.

Ablauf einer blue/green Bereitstellung

Führen Sie die folgenden wichtigen Schritte aus, wenn Sie eine blue/green Bereitstellung für Aurora-DB-Cluster-Updates verwenden.

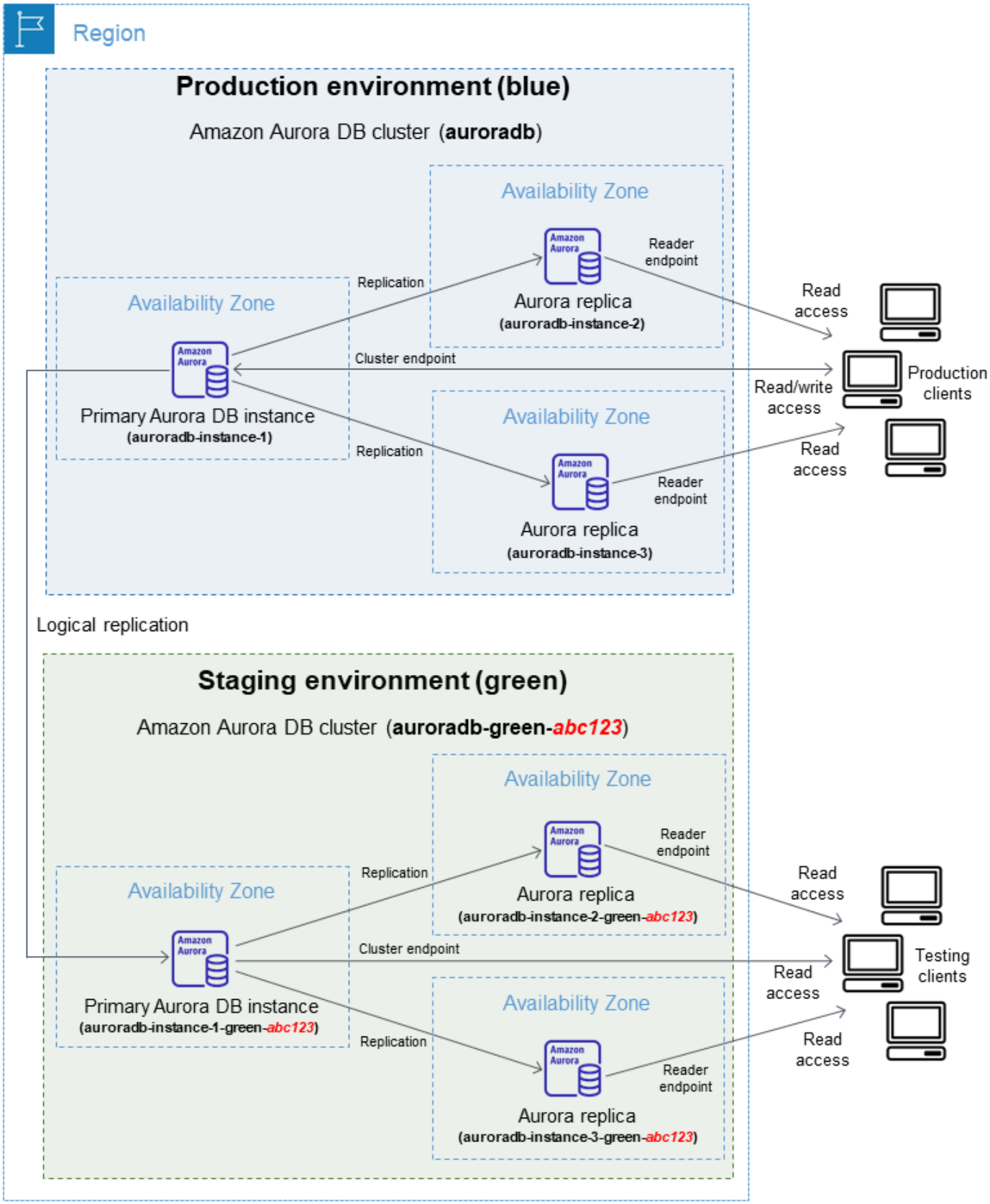
1. Identifizieren Sie einen Produktions-DB-Cluster, der aktualisiert werden muss.

Die folgende Abbildung zeigt ein Beispiel für einen Produktions-DB-Cluster.



2. Erstellen Sie die blue/green Bereitstellung. Detaillierte Anweisungen finden Sie unter Eine blue/green Bereitstellung in — Amazon Aurora erstellen.

   Die folgende Abbildung zeigt ein Beispiel für eine blue/green Bereitstellung der Produktionsumgebung aus Schritt 1. Bei der Erstellung der blue/green Bereitstellung kopiert RDS die komplette Topologie und Konfiguration des Aurora-DB-Clusters, um eine grüne Umgebung zu schaffen. Den Namen des kopierten DB-Clusters und der DB-Instances wird `-green-`*`random-characters`* angehängt. Die Staging-Umgebung im Image enthält den DB-Cluster (auroradb-green-). *`abc123`* Es enthält auch die drei DB-Instances im DB-Cluster (auroradb-instance1-green-, auroradb-instance2-green- und auroradb-instance3-green-). *`abc123 abc123 abc123`*


Region
Production environment (blue)
Amazon Aurora DB cluster (auroradb)
Availability Zone
Aurora replica
(auroradb-instance-2)
Reader endpoint
Replication
Availability Zone
Primary Aurora DB instance
(auroradb-instance-1)
Cluster endpoint
Read access
Production clients
Read/write access
Availability Zone
Aurora replica
(auroradb-instance-3)
Replication
Reader endpoint
Read access
Logical replication
Staging environment (green)
Amazon Aurora DB cluster (auroradb-green-abc123)
Availability Zone
Aurora replica
(auroradb-instance-2-green-abc123)
Reader endpoint
Replication
Availability Zone
Primary Aurora DB instance
(auroradb-instance-1-green-abc123)
Cluster endpoint
Read access
Testing clients
Read access
Availability Zone
Aurora replica
(auroradb-instance-3-green-abc123)
Replication
Reader endpoint
Read access

Wenn Sie die blue/green Bereitstellung erstellen, können Sie eine höhere DB-Engine-Version und eine andere DB-Cluster-Parametergruppe für den DB-Cluster in der grünen Umgebung angeben. Außerdem können Sie eine andere DB-Parametergruppe für die DB-Instances im DB-Cluster angeben.

RDS konfiguriert auch die Replikation von der primären DB-Instance in der blauen Umgebung zur primären DB-Instance in der grünen Umgebung.

> **Important**
>
> Für Aurora MySQL Version 3 erlaubt der DB-Cluster in der grünen Umgebung nach der Erstellung der blue/green Bereitstellung standardmäßig keine Schreibvorgänge. Dies gilt jedoch nicht für Benutzer, die über die `CONNECTION_ADMIN`-Berechtigung verfügen, darunter der Aurora-Masterbenutzer. Benutzer mit dieser Berechtigung können das Verhalten `read_only` außer Kraft setzen. Weitere Informationen finden Sie unter Rollenbasiertes Berechtigungsmodell.

3. Nehmen Sie Änderungen an der Staging-Umgebung vor.

   Sie können beispielsweise die DB-Instance-Klasse ändern, die von einer oder mehreren DB-Instances in der grünen Umgebung verwendet wird.

   Weitere Informationen zum Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

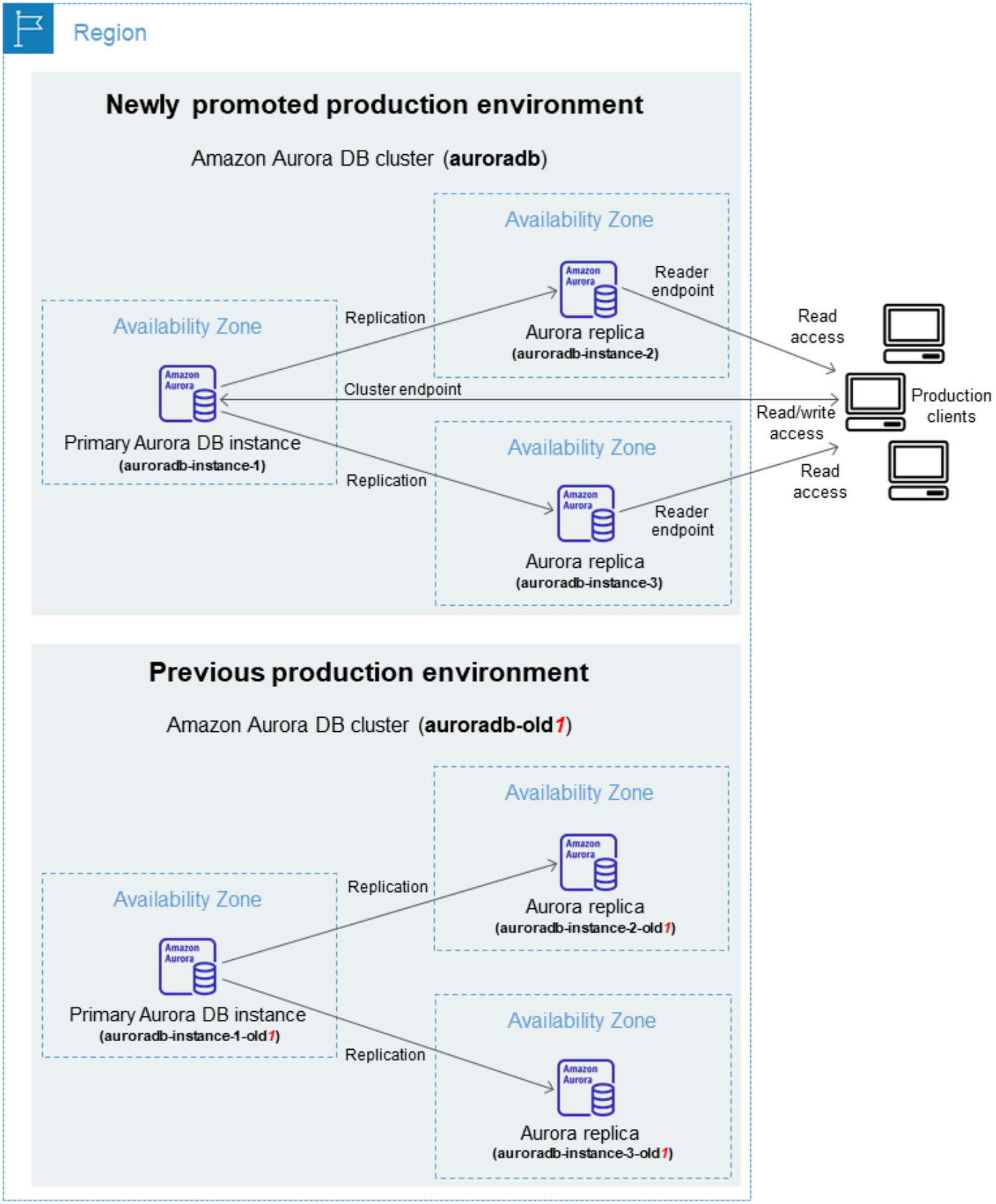
4. Testen Sie Ihre Staging-Umgebung.

   Während des Testens empfehlen wir, dass Sie Ihre Datenbanken in der grünen Umgebung schreibgeschützt lassen. Aktivieren Sie Schreibvorgänge in der grünen Umgebung mit Vorsicht, da sie zu Replikationskonflikten führen können. Sie können auch zu ungewollten Daten in den Produktionsdatenbanken nach der Umstellung führen. Um Schreibvorgänge für Aurora MySQL zu aktivieren, legen Sie den Parameter `read_only` auf `0` fest und starten Sie dann die DB-Instance neu. Legen Sie für Aurora PostgreSQL den Parameter `default_transaction_read_only` auf Sitzungsebene auf `off` fest.

5. Wenn Sie bereit sind, können Sie die Umstellung vornehmen und die Staging-Umgebung zur neuen Produktionsumgebung machen. Detaillierte Anweisungen finden Sie unter Wechseln einer blue/green Bereitstellung in Amazon Aurora.

Die Umstellung führt zu Ausfallzeiten. Die Ausfallzeit beträgt normalerweise weniger als eine Minute, kann aber je nach Workload länger sein.

Die folgende Abbildung zeigt die DB-Cluster nach der Umstellung.


Region
Newly promoted production environment
Amazon Aurora DB cluster (auroradb)
Availability Zone
Amazon Aurora
Reader endpoint
Availability Zone
Replication
Aurora replica
(auroradb-instance-2)
Read access
Cluster endpoint
Production clients
Read/write access
Primary Aurora DB instance
(auroradb-instance-1)
Availability Zone
Replication
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Previous production environment
Amazon Aurora DB cluster (auroradb-old1)
Availability Zone
Availability Zone
Replication
Aurora replica
(auroradb-instance-2-old1)
Primary Aurora DB instance
(auroradb-instance-1-old1)
Availability Zone
Replication
Aurora replica
(auroradb-instance-3-old1)

Nach der Umstellung wird der Aurora-DB-Cluster in der grünen Umgebung zum neuen Produktions-DB-Cluster. Die Namen und Endpunkte in der aktuellen Produktionsumgebung werden der neuen Produktionsumgebung zugewiesen, sodass keine Änderungen an Ihrer Anwendung erforderlich sind. Infolgedessen fließt Ihr Produktionsdatenverkehr jetzt in die neue Produktionsumgebung. Der DB-Cluster und die DB-Instances in der blauen Umgebung werden umbenannt, indem `-old`*n* an den aktuellen Namen angehängt wird, wobei *n* eine Zahl ist. Angenommen, der Name der DB-Instance in der blauen Umgebung lautet `auroradb-instance-1`. Nach der Umstellung könnte der Name der DB-Instance `auroradb-instance-1-old1` lauten.

In dem Beispiel in der Abbildung werden bei der Umstellung die folgenden Änderungen vorgenommen:

- Der DB-Cluster in der grünen Umgebung `auroradb-green-abc123` wird zum Produktions-DB-Cluster und erhält den Namen `auroradb`.
- Die DB-Instance in der grünen Umgebung `auroradb-instance1-green-abc123` wird zur Produktions-DB-Instance und erhält den Namen `auroradb-instance1`.
- Die DB-Instance in der grünen Umgebung `auroradb-instance2-green-abc123` wird zur Produktions-DB-Instance und erhält den Namen `auroradb-instance2`.
- Die DB-Instance in der grünen Umgebung `auroradb-instance3-green-abc123` wird zur Produktions-DB-Instance und erhält den Namen `auroradb-instance3`.
- Der DB-Cluster in der blauen Umgebung `auroradb` wird in `auroradb-old1` umbenannt.
- Die DB-Instance in der blauen Umgebung `auroradb-instance1` wird in `auroradb-instance1-old1` umbenannt.
- Die DB-Instance in der blauen Umgebung `auroradb-instance2` wird in `auroradb-instance2-old1` umbenannt.
- Die DB-Instance in der blauen Umgebung `auroradb-instance3` wird in `auroradb-instance3-old1` umbenannt.

6. Wenn Sie eine blue/green Bereitstellung nicht mehr benötigen, können Sie sie löschen. Detaillierte Anweisungen finden Sie unter Löschen einer Blau/Grün-Bereitstellung in Amazon Aurora.

   Nach der Umstellung wird die vorherige Produktionsumgebung nicht gelöscht, sodass Sie sie bei Bedarf für Regressionstests verwenden können.

Autorisieren des Zugriffs auf Aurora blue/green Aurora-Bereitstellungsvorgänge

Benutzer müssen über die erforderlichen Berechtigungen verfügen, um Operationen im Zusammenhang mit Blau/Grün-Bereitstellungen ausführen zu können. Sie können IAM-Richtlinien erstellen, die Benutzern und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Sie können diese Richtlinien dann den IAM-Berechtigungssätzen oder -Rollen zuordnen, die diese Berechtigungen benötigen. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora.

Der Benutzer, der eine blue/green Bereitstellung erstellt, muss über die erforderlichen Berechtigungen verfügen, um die folgenden RDS-Operationen durchzuführen:

- `rds:CreateBlueGreenDeployment`
- `rds:AddTagsToResource`
- `rds:CreateDBCluster`
- `rds:CreateDBInstance`
- `rds:CreateDBClusterEndpoint`

Der Benutzer, der zu einer blue/green Bereitstellung wechselt, muss über die erforderlichen Berechtigungen verfügen, um die folgenden RDS-Vorgänge auszuführen:

- `rds:SwitchoverBlueGreenDeployment`
- `rds:ModifyDBCluster`
- `rds:PromoteReadReplicaDBCluster`

Der Benutzer, der eine blue/green Bereitstellung löscht, muss über Berechtigungen verfügen, um die folgenden RDS-Vorgänge auszuführen:

- `rds:DeleteBlueGreenDeployment`
- `rds:DeleteDBCluster`
- `rds:DeleteDBInstance`
- `rds:DeleteDBClusterEndpoint`

Aurora stellt für Sie Ressourcen in der Staging-Umgebung bereit und ändert sie. Zu diesen Ressourcen gehören DB-Instances, die eine intern definierte Namenskonvention verwenden. Daher dürfen angehängte IAM-Richtlinien keine unvollständigen Muster für Ressourcennamen enthalten, wie `my-db-prefix-*`. Nur Platzhalter (*) werden unterstützt. Im Allgemeinen empfehlen wir, zur Steuerung des Zugriffs auf diese Ressourcen Ressourcen-Tags und andere unterstützte Attribute anstelle von Platzhaltern zu verwenden. Weitere Informationen finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für Amazon RDS.

Zusätzliche Berechtigungen für Aurora Global Database Blue/Green Deployments

Beim Erstellen von blue/green Bereitstellungen für Aurora Global Database-Cluster benötigen Benutzer zusätzlich zu den oben aufgeführten Berechtigungen die folgenden Berechtigungen, um Operationen zur Verwaltung der globalen Cluster-Topologie durchzuführen.

Der Benutzer, der eine blue/green Bereitstellung erstellt, muss über Berechtigungen verfügen, um die folgenden RDS-Operationen durchzuführen:

- `rds:CreateGlobalCluster`

Der Benutzer, der zu einer blue/green Bereitstellung wechselt, muss über die erforderlichen Berechtigungen verfügen, um die folgenden RDS-Vorgänge auszuführen:

- `rds:ModifyGlobalCluster`
- `rds:PromoteReadReplicaDBCluster`

Der Benutzer, der eine blue/green Bereitstellung löscht, muss über Berechtigungen verfügen, um die folgenden RDS-Vorgänge auszuführen:

- `rds:DeleteGlobalCluster`

Einschränkungen und Überlegungen für Aurora blue/green Aurora-Bereitstellungen

Blau/Grün-Bereitstellungen in Amazon RDS erfordern eine sorgfältige Abwägung von Faktoren wie Replikations-Slots, Ressourcenmanagement, Instance-Größe und potenziellen Auswirkungen auf die Datenbankleistung. In den folgenden Abschnitten finden Sie Anleitungen zur Optimierung Ihrer

Bereitstellungsstrategie, um minimale Ausfallzeiten, reibungslose Übergänge und eine effektive Verwaltung Ihrer Datenbankumgebung sicherzustellen.

Themen

- Einschränkungen für Bereitstellungen blue/green
- Einschränkungen der Aurora Global Database für blue/green Bereitstellungen
- Überlegungen zu Bereitstellungen blue/green

Einschränkungen für Bereitstellungen blue/green

Die folgenden Einschränkungen gelten für blue/green Bereitstellungen.

Themen

- Allgemeine Einschränkungen für Blau/Grün-Bereitstellungen
- Einschränkungen von Aurora MySQL für blue/green Bereitstellungen

Allgemeine Einschränkungen für Blau/Grün-Bereitstellungen

Die folgenden allgemeinen Einschränkungen gelten für blue/green Bereitstellungen:

- Sie können einen Cluster, der Teil einer blue/green Bereitstellung ist, nicht beenden und starten.
- Blau/Grün-Bereitstellungen unterstützen nicht die Verwaltung von Master-Benutzerpasswörtern mit AWS Secrets Manager.
- Wenn Sie versuchen, einen Backtrack auf dem blauen DB-Cluster zu erzwingen, wird die blue/green Bereitstellung unterbrochen und der Switchover wird blockiert.
- Während der Umstellung sind für die blauen und grünen Umgebungen keine Null-ETL-Integrationen mit Amazon Redshift möglich. Sie müssen zuerst die Integration löschen und umstellen. Anschließend erstellen Sie die Integration neu.
- Der Event Scheduler (`event_scheduler`Parameter) muss in der grünen Umgebung deaktiviert sein, wenn Sie eine Bereitstellung erstellen. blue/green Dadurch wird verhindert, dass Ereignisse in der grünen Umgebung generiert werden und zu Inkonsistenzen führen.
- Auto-Scaling-Richtlinien, die im blauen DB-Cluster konfiguriert sind, werden nicht in die grüne Umgebung übernommen. Sie müssen sie nach der Umstellung neu konfigurieren, unabhängig davon, ob sie ursprünglich in der blauen oder grünen Umgebung eingerichtet wurden.

- Sie können einen unverschlüsselten DB-Cluster nicht in einen verschlüsselten DB-Cluster ändern. Zusätzlich können Sie einen verschlüsselten DB-Cluster nicht in einen unverschlüsselten DB-Cluster ändern.
- Sie können einen blauen DB-Cluster nicht auf eine höhere Engine-Version als die des entsprechenden grünen DB-Clusters ändern.
- Die Ressourcen in der blauen und der grünen Umgebung müssen sich in demselben AWS-Konto befinden.
- Blau/Grün-Bereitstellungen werden für die folgenden Funktionen nicht unterstützt:
  - Amazon-RDS-Proxy
  - Regionsübergreifende Lesereplikate
  - Aurora Serverless v1-DB-Cluster
  - CloudFormation

Einschränkungen von Aurora MySQL für blue/green Bereitstellungen

Die folgenden Einschränkungen gelten für blue/green Bereitstellungen von Aurora MySQL :

- Der Quell-DB-Cluster kann keine Datenbanken mit dem Namen `tmp` enthalten. Datenbanken mit diesem Namen werden nicht in die grüne Umgebung kopiert.
- Der blaue DB-Cluster kann kein externes Binlog-Replikat sein.
- Wenn für den Quell-DB-Cluster die Rückverfolgung aktiviert ist, wird der grüne DB-Cluster ohne Unterstützung für die Rückverfolgung erstellt. Das liegt daran, dass Backtracking bei der Replikation von Binärprotokollen (Binlog), die für blue/green Bereitstellungen erforderlich ist, nicht funktioniert. Weitere Informationen finden Sie unter the section called “Rückverfolgen eines DB-Clusters”.
- Blaue/grüne Bereitstellungen unterstützen den AWS JDBC-Treiber für MySQL nicht. Weitere Informationen finden Sie unter Bekannte Einschränkungen von. GitHub

Die folgenden Einschränkungen gelten für Blau/Grün-Bereitstellungen für Aurora PostgreSQL, .

- Nicht protokollierte Tabellen werden nicht in die grüne Umgebung repliziert, es sei denn, der Parameter `rds.logically_replicate_unlogged_tables` ist im blauen DB-Cluster auf `1` gesetzt. Ändern Sie diesen Parameterwert nicht, nachdem Sie eine blue/green Bereitstellung erstellt haben, um mögliche Replikationsfehler bei Tabellen ohne Protokollierung zu vermeiden.

- Bei dem blauen DB-Cluster kann es sich nicht um eine logische Quelle (Publisher) oder ein Replikat (Subscriber) handeln.
- Wenn der blaue DB-Cluster als fremder Server einer FDW-Erweiterung (Foreign Data Wrapper) konfiguriert ist, müssen Sie den Endpunktnamen des -Clusters anstelle von IP-Adressen verwenden. Dadurch kann die Konfiguration auch nach der Umstellung funktionsfähig bleiben.
- In einer blue/green Bereitstellung benötigt jede Datenbank einen logischen Replikationsslot. Mit zunehmender Anzahl von Datenbanken nimmt der Ressourcenaufwand zu, was möglicherweise zu Verzögerungen bei der Replikation führen kann, insbesondere wenn der DB-Cluster nicht ausreichend skaliert ist. Die Auswirkungen hängen von Faktoren wie dem Datenbank-Workload und der Anzahl der Verbindungen ab. Um dies zu vermeiden, sollten Sie erwägen, Ihre DB-Instance-Klasse zu skalieren oder die Anzahl der Datenbanken im Quellcluster zu verringern.
- Blau/Grün-Bereitstellungen werden für Babelfish für Aurora PostgreSQL nur für Version 15.7 und höhere 15-Versionen sowie für Version 16.3 und höhere 16-Versionen unterstützt.
- Wenn Sie Ausführungspläne in Aurora Replicas erfassen möchten, müssen Sie beim Aufrufen der `apg_plan_mgmt.create_replica_plan_capture`-Funktion den Endpunkt des blauen DB-Clusters angeben. Dadurch wird sichergestellt, dass die Planerfassungen nach der Umstellung weiterhin funktionieren. Weitere Informationen finden Sie unter the section called “Erfassung von Aurora-PostgreSQL-Ausführungsplänen”.
- Der Prozess zum Anwenden der logischen Replikation in der grünen Umgebung erfolgt über einen einzigen Thread. Wenn die blaue Umgebung ein hohes Volumen an Schreibverkehr generiert, kann die grüne Umgebung möglicherweise nicht Schritt halten. Dies kann zu Verzögerungen oder Fehlern bei der Replikation führen, insbesondere bei Workloads, die einen kontinuierlich hohen Schreibdurchsatz erzeugen. Stellen Sie sicher, dass Sie Ihre Workloads gründlich testen. Für Szenarien, die größere Versions-Upgrades und die Verarbeitung umfangreicher Schreib-Workloads erfordern, sollten Sie alternative Ansätze in Betracht ziehen, wie die Verwendung von AWS Database Migration Service (AWS DMS) oder selbstverwalteter logischer Replikation.
- Das Erstellen neuer Partitionen in partitionierten Tabellen wird bei Blau/Grün-Bereitstellungen für Aurora PostgreSQL nicht unterstützt. Das Erstellen neuer Partitionen umfasst Data Definition Language (DDL)-Operationen wie `CREATE TABLE`, die nicht von der blauen Umgebung in die grüne Umgebung repliziert werden. Bestehende partitionierte Tabellen und ihre Daten werden jedoch in die grüne Umgebung repliziert.
- Die folgenden Einschränkungen gelten für PostgreSQL-Erweiterungen:
  - Die `pg_partman` Erweiterung muss in der blauen Umgebung deaktiviert sein, wenn Sie eine blue/green Bereitstellung erstellen. Die Erweiterung führt DDL-Operationen wie etwa `CREATE TABLE` durch, die die logische Replikation von der blauen in die grüne Umgebung unterbrechen.

- Die `pg_cron` Erweiterung muss nach der Erstellung der blue/green Bereitstellung in allen grünen Datenbanken deaktiviert bleiben. Die Erweiterung verfügt über Hintergrund-Worker, die als Superuser ausgeführt werden und die Schreibschutzeinstellung der grünen Umgebung umgehen, was zu Replikationskonflikten führen kann.
- Für die `apg_plan_mgmt`-Erweiterung muss der `apg_plan_mgmt.capture_plan_baselines`-Parameter für alle grünen Datenbanken auf `off` gesetzt sein, um Primärschlüsselkonflikte zu vermeiden, wenn ein identischer Plan in der blauen Umgebung erfasst wird. Weitere Informationen finden Sie unter the section called “Übersicht über die Abfrageplanverwaltung in Aurora PostgreSQL”.
- Die `pgactive` Erweiterungen `pglogical` und müssen in der blauen Umgebung deaktiviert sein, wenn Sie eine blue/green Bereitstellung erstellen. Nachdem Sie die grüne Umgebung zur neuen Produktionsumgebung umgestellt haben, können Sie die Erweiterungen wieder aktivieren. Dazu kann die blaue Datenbank kein logischer Subscriber einer externen Instance sein.
- Wenn Sie die `pgAudit`-Erweiterung verwenden, muss sie in den gemeinsam genutzten Bibliotheken (`shared_preload_libraries`) der benutzerdefinierten DB-Parametergruppen sowohl für die blaue als auch für die grüne DB-Instance verbleiben. Weitere Informationen finden Sie unter the section called “Einrichten der pgAudit-Erweiterung”.

Spezifische Einschränkungen für Bereitstellungen im Zusammenhang mit der logischen Replikation blue/green

PostgreSQL hat bestimmte Einschränkungen in Bezug auf die logische Replikation, die sich in Einschränkungen bei der Erstellung von blue/green Bereitstellungen für Aurora PostgreSQL-DB-Cluster, für PostgreSQL-DB-Instances niederschlagen.

In der folgenden Tabelle werden die Einschränkungen der logischen Replikation beschrieben, die für Blau/Grün-Bereitstellungen für Aurora PostgreSQL gelten. Weitere Informationen finden Sie in der Dokumentation zur logischen Replikation in PostgreSQL.

Einschränkung	Erklärung
Data Definition Language (DDL)-Anweisungen wie `CREATE TABLE` und `CREATE`	Wenn Aurora eine DDL-Änderung in der blauen Umgebung erkennt, gehen die grünen Datenbanken in den Status Replikation herabgestuft über. Sie müssen die blue/green Bereitstellung und alle grünen Datenbanken löschen und sie dann neu erstellen.

Einschränkung	Erklärung
`SCHEMA` werden nicht von der blauen in die grüne Umgebung repliziert.	
Data Control Language (DCL)-Anweisungen wie `GRANT` und `REVOKE` werden nicht von der blauen in die grüne Umgebung repliziert.	Wenn Aurora einen Versuch erkennt, eine DCL-Anweisung in der blauen Umgebung auszuführen, wird eine Warnmeldung angezeigt. Es ist keine Konfiguration oder API verfügbar, um dieses Verhalten zu ändern, da dies eine Einschränkung des blue/green Bereitstellungsprozesses darstellt.
`NEXTVAL`-Operationen an Sequenzobjekten werden nicht zwischen der blauen und der grünen Umgebung synchronisiert.	Während der Umstellung erhöht Aurora die Sequenzwerte in der grünen Umgebung so, dass sie denen in der blauen Umgebung entsprechen. Wenn Sie Tausende von Sequenzen haben, kann dies die Umstellung verzögern.

Einschränkung	Erklärung
Große Objekte in der blauen Umgebung werden nicht in die grüne Umgebung repliziert. Dies gilt sowohl für bestehende große Objekte als auch für alle während des blue/green Bereitstellungsprozesses neu erstellten oder geänderten großen Objekte.	Wenn Aurora die Erstellung oder Änderung großer Objekte in der blauen Umgebung erkennt, die in der `pg_largeobject` -Systemtabelle gespeichert sind, gehen die grünen Datenbanken in den Status Replikation herabgestuft über. Sie müssen die blue/green Bereitstellung und alle grünen Datenbanken löschen und sie dann neu erstellen.
Durch das Aktualisieren materialisierter Ansichten wird die Replikation unterbrochen.	Das Aktualisieren materialisierter Sichten in der blauen Umgebung unterbricht die Replikation in die grüne Umgebung. Vermeiden Sie das Aktualisieren materialisierter Ansichten in der blauen Umgebung. Nach einer Umstellung können Sie sie manuell mit dem Befehl REFRESH MATERIALIZED VIEW aktualisieren oder eine Aktualisierung planen.
UPDATE- und DELETE-Operationen sind für Tabellen, die keinen Primärschlüssel haben, nicht zulässig.	Bevor Sie eine blue/green Bereitstellung erstellen, stellen Sie sicher, dass alle Tabellen über einen Primärschlüssel oder eine Verwendung `REPLICA IDENTITY FULL` verfügen. Verwenden Sie `REPLICA IDENTITY FULL` jedoch nur, wenn kein primärer oder eindeutiger Schlüssel vorhanden ist, da dies die Replikationsleistung beeinträchtigt. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation.

Einschränkungen der Aurora Global Database für blue/green Bereitstellungen

Zusätzlich zu den oben genannten allgemeinen und Engine-spezifischen Einschränkungen gelten die folgenden Einschränkungen für blue/green Bereitstellungen für Aurora Global Database:

- Alle Operationen müssen von derselben Region aus initiiert werden wie der Writer-Cluster der globalen Datenbank.
- Die Durchführung eines globalen Switchovers oder eines globalen Failovers führt dazu, dass die aktive blue/green Bereitstellung ungültig wird. Die blaugrüne Bereitstellung muss gelöscht und in der neuen primären Region neu erstellt werden.
- Wenn Sie für Aurora PostgreSQL die globale Schreibweiterleitung in Ihrer Produktionsumgebung aktiviert haben und eine blue/green Bereitstellung erstellen, ist die Schreibweiterleitung auf dem grünen Cluster deaktiviert. In der grünen Umgebung wird sie erst nach dem blue/green Switchover aktiviert, wenn die grüne Umgebung zur neuen Produktionsumgebung wird. Nach dem Switchover ist die Schreibweiterleitung auf dem `-old1` Cluster deaktiviert.
- Änderungen an der Topologie der globalen Datenbank nach der Erstellung der blue/green Einrichtung haben zur Folge, dass die aktive blue/green Bereitstellung ungültig wird. Die blaugrüne Bereitstellung müsste gelöscht und in der neuen Primärregion neu erstellt werden.
- Automatisierte Snapshots werden innerhalb der Backup-Aufbewahrungstage aufbewahrt, die ursprünglich in der alten blauen Umgebung konfiguriert waren. Automatisierte Snapshots aus dem alten blauen Cluster werden nicht in den grünen Cluster kopiert.
- Globales Failover wird während eines blue/green Switchovers unterstützt, aber ein globaler Switchover wird während eines Switchovers nicht unterstützt. blue/green
- Stellen Sie sicher, dass DB-Cluster und DB-Parametergruppen für die grüne Umgebung in allen sekundären Regionen mit identischen Namen vorhanden sind. Wenn die Parametergruppe in einer Region nicht verfügbar ist, wird die Standardparametergruppe in den Regionen verwendet.
- Vermeiden Sie die Verwendung von RDS Proxy für alle Mitglieder der globalen Datenbank während des blue/green Bereitstellungsswitchovers.

Überlegungen zu Bereitstellungen blue/green

Amazon RDS verfolgt Ressourcen in blue/green Bereitstellungen mit dem `DbiResourceId` und `DbClusterResourceId` jeder Ressource. Diese Ressourcen-ID ist eine AWS-Region eindeutige, unveränderliche Kennung für die Ressource.

Die Ressourcen-ID ist getrennt von der DB-Cluster-ID: Jede ist in der Datenbankkonfiguration in der RDS-Konsole aufgeführt.

Der Name (Cluster-ID) einer Ressource ändert sich, wenn Sie zu einer blue/green Bereitstellung wechseln, aber jede Ressource behält dieselbe Ressourcen-ID. Eine DB-Cluster-ID in der blauen Umgebung lautete beispielsweise `mycluster`. Nach der Umstellung könnte dieser DB-Cluster in `mycluster-old1` umbenannt sein. Die Ressourcen-ID des DB-Clusters ändert sich während der Umstellung jedoch nicht. Wenn Sie also die grünen Ressourcen auf die neuen Produktionsressourcen umstellen, stimmen ihre Ressourcen IDs nicht mit den blauen Ressourcen überein IDs , die zuvor in Produktion waren.

Nachdem Sie eine blue/green Bereitstellung umgestellt haben, sollten Sie erwägen, die Ressource IDs auf die Ressourcen der neu umgestellten Produktionsressourcen für integrierte Funktionen und Dienste zu aktualisieren, die Sie mit den Produktionsressourcen verwendet haben. Berücksichtigen Sie insbesondere die folgenden Aktualisierungen:

- Wenn Sie die Filterung mithilfe der RDS-API und der RDS-Ressource durchführen IDs, passen Sie die für die Filterung IDs verwendete Ressource nach dem Switchover an.
- Wenn Sie Ressourcen CloudTrail für die Überwachung verwenden, passen Sie die Benutzer von so an, CloudTrail dass sie die neue Ressource IDs nach dem Switchover verfolgen. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora-API-Aufrufen in AWS CloudTrail.
- Wenn Sie Datenbank-Aktivitätsstreams für Ressourcen in der blauen Umgebung verwenden, passen Sie Ihre Anwendung an, um die Datenbankereignisse für den neuen Stream nach der Umstellung zu überwachen. Weitere Informationen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Datenbankaktivitäts-Streams.
- Wenn Sie die Performance Insights Insights-API verwenden, passen Sie die Ressource IDs in API-Aufrufen nach dem Switchover an. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .

  Sie können eine Datenbank mit demselben Namen nach der Umstellung überwachen, diese enthält jedoch nicht die Daten, die vor der Umstellung vorhanden waren.
- Wenn Sie Ressourcen IDs in IAM-Richtlinien verwenden, stellen Sie sicher, dass Sie bei Bedarf die Ressource IDs der neu übertragenen Ressourcen hinzufügen. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora.
- Wenn Sie Ihrem DB-Cluster IAM-Rollen zugeordnet haben, stellen Sie sicher, dass Sie diese nach der Umstellung erneut zuordnen. Angehängte Rollen werden nicht automatisch in die grüne Umgebung kopiert.

- Wenn Sie sich mithilfe der IAM-Datenbankauthentifizierung bei Ihrem DB-Cluster authentifizieren, stellen Sie sicher, dass in der für den Datenbankzugriff verwendeten IAM-Richtlinie sowohl die blauen als auch die grünen Datenbanken unter dem Element `Resource` der Richtlinie aufgeführt sind. Dies ist erforderlich, um nach der Umstellung eine Verbindung mit der grünen Datenbank herzustellen. Weitere Informationen finden Sie unter the section called “Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff”.
- Wenn Sie einen manuellen DB-Cluster-Snapshot für einen DB-Cluster wiederherstellen möchten, der Teil einer blue/green Bereitstellung war, stellen Sie sicher, dass Sie den richtigen DB-Cluster-Snapshot wiederherstellen, indem Sie den Zeitpunkt überprüfen, zu dem der Snapshot erstellt wurde. Weitere Informationen finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.
- Nach dem Umschalten AWS Database Migration Service (AWS DMS) können die Replikationsaufgaben nicht fortgesetzt werden, da der Checkpoint aus der blauen Umgebung in der grünen Umgebung ungültig ist. Sie müssen die DMS-Aufgabe mit einem neuen Checkpoint neu erstellen, um die Replikation fortzusetzen.
- Amazon Aurora erstellt die Grün-Umgebung, indem das zugrunde liegende Aurora-Speichervolume in der Blau-Umgebung geklont wird. Das grüne Cluster-Volume speichert nur inkrementelle Änderungen, die in der Grün-Umgebung vorgenommen wurden. Wenn Sie den DB-Cluster in der Blau-Umgebung löschen, wächst die Größe des zugrunde liegenden Aurora-Speichervolumens in der Grün-Umgebung auf die volle Größe an. Weitere Informationen finden Sie unter the section called “Klonen eines Volumes für einen Aurora-DB-Cluster”.
- Wenn Sie dem DB-Cluster in der grünen Umgebung einer Blau/Grün-Bereitstellung eine DB-Instance hinzufügen, ersetzt die neue DB-Instance bei der Umstellung keine DB-Instance in der blauen Umgebung. Die neue DB-Instance wird jedoch im DB-Cluster beibehalten und wird in der neuen Produktionsumgebung zu einer DB-Instance.
- Wenn Sie eine DB-Instance im DB-Cluster in der grünen Umgebung einer blue/green deployment, you can't create a new DB instance to replace it in the blue/green Bereitstellung löschen.

  Wenn Sie eine neue DB-Instance mit demselben Namen und ARN wie die gelöschte DB-Instance erstellen, hat sie eine andere `DbiResourceId`. Sie ist folglich nicht Teil der grünen Umgebung.

  Das folgende Verhalten ergibt sich, wenn Sie eine DB-Instance im DB-Cluster der grünen Umgebung löschen:

  - Wenn die DB-Instance in der blauen Umgebung mit dem gleichen Namen vorhanden ist, wird sie nicht auf die DB-Instance in der grünen Umgebung umgestellt. Diese DB-Instance wird nicht umbenannt, indem dem DB-Instance-Namen `-old`*n* angefügt wird.

  - Jede Anwendung, die auf die DB-Instance in der blauen Umgebung verweist, verwendet nach der Umstellung weiterhin dieselbe DB-Instance.
- Wenn Sie Ressourcen-Tags für die Zugriffskontrolle oder das Betriebsmanagement verwenden, müssen Sie sich darüber im Klaren sein, dass Tag-Änderungen erst beim Switchover zwischen blauen und grünen Umgebungen synchronisiert werden. Wenn Sie eine blue/green Bereitstellung erstellen, werden Tags aus der blauen Umgebung in die grüne Umgebung kopiert. Nach der Erstellung werden alle Tag-Änderungen, die Sie an einer der Umgebungen vornehmen, nicht automatisch synchronisiert. Beim Switchover ersetzen blaue Umgebungs-Tags alle Tags in der grünen Umgebung. Wenden Sie alle erforderlichen Tags auf die blaue Umgebung an, bevor Sie die blue/green Bereitstellung erstellen, oder wenden Sie die erforderlichen Tags nach dem Switchover erneut auf die neue Produktionsumgebung an. Weitere Informationen zu Tags erhalten Sie unter Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen.

Bewährte Methoden für Aurora blue/green Aurora-Bereitstellungen

Im Folgenden finden Sie bewährte Methoden für blue/green Bereitstellungen.

Themen

- Allgemeine bewährte Methoden für Bereitstellungen blue/green
- — Bewährte Methoden für blue/green Bereitstellungen
- Bewährte Methoden für Bereitstellungen von Aurora PostgreSQL blue/green
- Bewährte Methoden Aurora Global Database

Allgemeine bewährte Methoden für Bereitstellungen blue/green

Beachten Sie die folgenden allgemeinen bewährten Methoden, wenn Sie eine Blau/Grün-Bereitstellung erstellen.

- Testen Sie den Aurora-DB-Cluster in der grünen Umgebung gründlich, bevor Sie umstellen.
- Halten Sie Ihre Datenbanken in der grünen Umgebung schreibgeschützt. Wir empfehlen, Schreibvorgänge in der grünen Umgebung mit Vorsicht zu aktivieren, da sie zu Replikationskonflikten führen können. Sie können auch zu ungewollten Daten in den Produktionsdatenbanken nach der Umstellung führen.
- Wenn Sie eine blue/green Bereitstellung verwenden, um Schemaänderungen zu implementieren, nehmen Sie nur replikationskompatible Änderungen vor.

Sie können beispielsweise am Ende einer Tabelle neue Spalten hinzufügen, ohne die Replikation von der blauen zur grünen Bereitstellung zu unterbrechen. Schemaänderungen, wie das Umbenennen von Spalten oder Tabellen, führen jedoch dazu, dass die Replikation zur Grün-Bereitstellung unterbrochen wird.

Weitere Informationen zu replikationskompatiblen Änderungen finden Sie in der MySQL-Dokumentation unter Replikation mit unterschiedlichen Tabellendefinitionen auf Quelle und Replikat sowie unter Einschränkungen in der Dokumentation zur logischen Replikation in PostgreSQL.

- Verwenden Sie den Cluster-Endpunkt, den Reader-Endpunkt oder den benutzerdefinierten Endpunkt für alle Verbindungen in beiden Umgebungen. Verwenden Sie keine Instance-Endpunkte oder benutzerdefinierten Endpunkte mit statischen oder Ausschlusslisten.
- Wenn Sie zu einer blue/green Bereitstellung wechseln, befolgen Sie die bewährten Methoden für den Switchover. Weitere Informationen finden Sie unter the section called “Bewährte Methoden für die Umstellung”.

— Bewährte Methoden für blue/green Bereitstellungen

Beachten Sie die folgenden bewährten Methoden, wenn Sie eine blue/green Bereitstellung aus einem Aurora MySQL-DB-Cluster einer erstellen.

- Wenn es in der grünen Umgebung zu Replikatverzögerungen kommt, sollten Sie die folgenden Schritte ausführen:
  - Deaktivieren Sie die Aufbewahrung von Binärprotokollen, falls sie nicht benötigt wird, oder deaktivieren Sie sie vorübergehend, bis die Replikation abgeschlossen ist. Setzen Sie dazu den `binlog_format`-DB-Cluster-Parameter wieder auf `0` und starten Sie die grüner Schreiber-DB-Instance neu.
  - Setzen Sie den Parameter `innodb_flush_log_at_trx_commit` in der grünen DB-Parametergruppe vorübergehend auf `0`. Wenn die Replikation aufgeholt hat, kehren Sie zum Standardwert 1 vor der Umstellung zurück. Wenn der temporäre Parameterwert unerwartet heruntergefahren wird oder es zu einem Absturz kommt, erstellen Sie die grüne Umgebung neu, um unentdeckte Datenbeschädigungen zu vermeiden. Weitere Informationen finden Sie unter the section called “Konfigurieren, wie oft der Protokollpuffer geleert wird”.

Bewährte Methoden für Bereitstellungen von Aurora PostgreSQL blue/green

Beachten Sie die folgenden bewährten Methoden, wenn Sie eine blue/green Bereitstellung aus einem Aurora PostgreSQL-DB-Cluster erstellen.

- Überwachen Sie den Write-Through-Cache für die logische Replikation in Aurora PostgreSQL und nehmen Sie gegebenenfalls Anpassungen am Cache-Puffer vor. Weitere Informationen finden Sie unter the section called “Überwachung des Write-Through-Cache für die logische Replikation”.
- Erhöhen Sie den Wert des `logical_decoding_work_mem`-DB-Parameters in der blauen Umgebung. Dadurch muss auf der Festplatte weniger dekodiert werden und stattdessen wird Arbeitsspeicher beansprucht. Weitere Informationen finden Sie unter the section called “Arbeitsspeicher für logische Dekodierung anpassen”.
  - Mithilfe der Metrik können Sie den Transaktionsüberlauf überwachen, der auf die Festplatte geschrieben wird. `ReplicationSlotDiskUsage` CloudWatch Diese Metrik bietet Einblicke in die Festplattennutzung von Replikations-Slots und hilft so, zu erkennen, wann Transaktionsdaten die Speicherkapazität überschreiten und auf der Festplatte gespeichert werden. Mit der Metrik können Sie den `FreeableMemory` CloudWatch freien Speicher überwachen. Weitere Informationen finden Sie unter the section called “Metriken auf Instance-Ebene für Amazon Aurora”.
  - In Aurora PostgreSQL Version 14 und höher können Sie die Größe von logischen Überlaufdateien mithilfe der Systemansicht `pg_stat_replication_slots` überwachen.
- Aktualisieren Sie alle Ihre PostgreSQL-Erweiterungen auf die neueste Version, bevor Sie ein blue/green Deployment erstellen. Weitere Informationen finden Sie unter the section called “Aktualisieren von PostgreSQL-Erweiterungen”.
- Wenn Sie die `aws_s3`-Erweiterung verwenden, gewähren Sie dem grünen DB-Cluster über eine IAM-Rolle Zugriff auf Amazon S3, nachdem die grüne Umgebung erstellt wurde. Dadurch können die Import- und Exportbefehle auch nach der Umstellung weiter funktionieren. Detaillierte Anleitungen finden Sie unter the section called “Einrichten des Zugriffs auf einen Amazon-S3-Bucket”.
- Wenn Sie eine höhere Engine-Version für die grüne Umgebung angeben, führen Sie den `ANALYZE`-Vorgang für alle Datenbanken aus, um die `pg_statistic`-Tabelle zu aktualisieren. Optimizer-Statistiken werden während eines Hauptversions-Upgrades nicht übertragen, daher müssen Sie alle Statistiken neu generieren, um Leistungsprobleme zu vermeiden. Weitere bewährte Methoden bei Upgrades von Hauptversionen finden Sie unter the section called “Durchführen eines Hauptversions-Upgrades”.

- Vermeiden Sie es, Trigger als `ENABLE REPLICA` oder `ENABLE ALWAYS` zu konfigurieren, wenn der Trigger auf der Quelle zur Datenmanipulation verwendet wird. Andernfalls überträgt das Replikationssystem die Änderungen und führt den Trigger aus, was zu Duplikaten führt.
- Transaktionen mit langer Laufzeit können zu erheblichen Verzögerungen bei der Replikation führen. Um die Replikationsverzögerung zu verringern, sollten Sie Folgendes in Betracht ziehen:
  - Reduzieren Sie lang andauernde Transaktionen und Untertransaktionen, die verzögert werden können, bis die grüne Umgebung die blaue Umgebung eingeholt hat.
  - Reduzieren Sie Massenoperationen in der blauen Umgebung, bis die grüne Umgebung die blaue Umgebung eingeholt hat.
  - Initiieren Sie vor der Erstellung der Bereitstellung einen manuellen Vakuum-Freeze-Vorgang für ausgelastete Tabellen. blue/green Eine Anleitung dazu finden Sie unter Durchführen einer manuellen Bereinigungseinfrierung.
  - Deaktivieren Sie in PostgreSQL Version 12 und höher den Parameter `index_cleanup` für große oder stark frequentierte Tabellen, um die Effizienz der regelmäßigen Wartung von blauen Datenbanken zu verbessern. Weitere Informationen finden Sie unter Möglichst schnelles Bereinigen einer Tabelle.

    > **Note**
    >
    > Wenn die Indexbereinigung während des Bereinigens regelmäßig übersprungen wird, kann dies zu einer Überlastung des Index führen, wodurch die Scanleistung beeinträchtigt werden kann. Es hat sich bewährt, diesen Ansatz nur während einer blue/green Bereitstellung zu verwenden. Sobald die Bereitstellung abgeschlossen ist, empfehlen wir, die regelmäßige Indexwartung und -bereinigung wieder aufzunehmen.

  - Replikatverzögerungen können auftreten, wenn die blauen und grünen DB-Instances für die Workload zu klein sind. Stellen Sie sicher, dass Ihre DB-Instances ihre Ressourcengrenzen für den Instance-Typ nicht erreichen. Weitere Informationen finden Sie unter the section called “Analysieren Sie die Ressourcennutzung mit Metriken CloudWatch ”.
- Eine langsame Replikation kann dazu führen, dass Sender und Empfänger häufig neu gestartet werden, was die Synchronisierung verzögert. Um sicherzustellen, dass sie aktiv bleiben, deaktivieren Sie Timeouts, indem Sie den Parameter `wal_sender_timeout` in der blauen Umgebung auf `0` und den Parameter `wal_receiver_timeout` in der grünen Umgebung auf `0` setzen.
- Überprüfen Sie die Leistung Ihrer UPDATE- und DELETE-Anweisungen und bewerten Sie, ob das Erstellen eines Indexes für die in der WHERE-Klausel verwendete Spalte diese Abfragen

optimieren kann. Dies kann die Leistung verbessern, wenn die Vorgänge in der grünen Umgebung erneut ausgeführt werden. Weitere Informationen finden Sie unter the section called “Überprüfen Sie Prädikatfilter auf Abfragen, die Wartezeiten generieren”.

- Wenn Sie Trigger verwenden, stellen Sie sicher, dass sie das Erstellen, Aktualisieren und Löschen von `pg_catalog.pg_publication`-, `pg_catalog.pg_subscription`- und `pg_catalog.pg_replication_slots`-Objekten, deren Namen mit „rds“ beginnen, nicht beeinträchtigen.
- Wenn Babelfish auf dem Quell-DB-Cluster aktiviert ist, müssen die folgenden Parameter in der Ziel-DB-Cluster-Parametergruppe für die grüne Umgebung dieselben Einstellungen haben wie in der Parametergruppe des Quell-DB-Clusters:
  - rds.babelfish_status
  - babelfishpg_tds.tds_default_numeric_precision
  - babelfishpg_tds.tds_default_numeric_scale
  - babelfishpg_tsql.default_locale
  - babelfishpg_tsql.migration_mode
  - babelfishpg_tsql.server_colation_name

  Weitere Informationen zu diesen Parametern finden Sie unter the section called “Einstellungen der DB-Cluster-Parametergruppe für Babelfish ”.

Bewährte Methoden Aurora Global Database

Zusätzlich zu den oben aufgeführten allgemeinen und Engine-spezifischen Best Practices sollten Sie die folgenden Best Practices für die Aurora Global Database berücksichtigen.

- Überwachen Sie die folgenden CloudWatch Kennzahlen, um Perioden geringer Aktivität in Ihrer Produktionsumgebung zu identifizieren:
  - `DatabaseConnections`
  - `ActiveTransactions`

  Planen Sie den blue/green Switchover während Ihres geplanten Wartungsfensters oder während einer Phase geringer Aktivität.
- Blue/Green switchover duration varies based on your workload and the number of secondary regions. When you initiate a blue/greenSwitchover: Der Service wartet, bis die Replikatverzögerung

Null erreicht hat, bevor er fortfährt. Es wird empfohlen, die Replikatverzögerung zu überprüfen, bevor Sie einen Switchover einleiten.

- Wenn Sie beabsichtigen, für Ihre grüne Umgebung einen anderen DB-Parameter oder eine andere DB-Cluster-Parametergruppe als die Standardparametergruppe zu verwenden, erstellen Sie die gewünschte Parametergruppe mit demselben Namen in allen sekundären Regionen, bevor Sie die Bereitstellung initiieren. blue/green

Eine blue/green Bereitstellung in — Amazon Aurora erstellen

RDS kopiert die Topologie und die Funktionen der blauen Umgebung in einen Staging-Bereich. Wenn die blaue DB-Instance Lesereplikate hat, werden diese als Replikate der grünen Instance kopiert. Der zugewiesene Speicher aller grünen Replikate entspricht der grünen Primär-Instancer, während andere Speicherparameter von den blauen Replikaten übernommen werden.

Wenn Sie eine blue/green Bereitstellung erstellen, geben Sie den DB-Cluster an, der in die Bereitstellung kopiert werden soll. Der DB-Cluster, den Sie auswählen, ist der Produktions-DB-Cluster und wird in der blauen Umgebung zum DB-Cluster. RDS kopiert die Topologie der blauen Umgebung zusammen mit den konfigurierten Funktionen in einen Staging-Bereich. Der DB-Cluster wird in die grüne Umgebung kopiert und RDS konfiguriert die Replikation vom DB-Cluster in der blauen Umgebung zum DB-Cluster in der grünen Umgebung. RDS kopiert auch alle DB-Instances im DB-Cluster.

Themen

- Vorbereiten einer Blau-Grün-Bereitstellung
- Angabe von Änderungen bei der Erstellung einer blue/green Bereitstellung
- Erstellen einer Blau/Grün-Bereitstellung
- Einstellungen für die Erstellung von Blau/Grün-Bereitstellungen

Vorbereiten einer Blau-Grün-Bereitstellung

Abhängig von der Engine, auf der Ihre ausgeführt wird, müssen Sie bestimmte Schritte ausführen, bevor Sie eine blue/green Bereitstellung erstellen.

Themen

- Einen Aurora MySQL-DB-Cluster für eine blue/green Bereitstellung vorbereiten
- Vorbereiten eines Aurora PostgreSQL-DB-Clusters für eine Bereitstellung blue/green

- Vorbereiten eines Aurora Global Database-DB-Clusters für eine blue/green Bereitstellung

Einen Aurora MySQL-DB-Cluster für eine blue/green Bereitstellung vorbereiten

Bevor Sie eine blue/green Bereitstellung für einen Aurora MySQL-DB-Cluster erstellen, muss der Cluster mit einer benutzerdefinierten DB-Cluster-Parametergruppe verknüpft werden, bei der die Binärprotokollierung (`binlog_format`) aktiviert ist. Für die Replikation von der blauen Umgebung in die grüne Umgebung ist die binäre Protokollierung erforderlich. Obwohl jedes Binlog-Format funktioniert, empfehlen wir `ROW`, um das Risiko von Replikationsinkonsistenzen zu verringern. Weitere Informationen über das Erstellen einer benutzerdefinierten DB-Parametergruppe und das Festlegen von Parametern finden Sie unter the section called “DB-Cluster-Parametergruppen”.

> **Note**
>
> Durch die Aktivierung der binären Protokollierung wird die Anzahl der I/O Festplattenschreibvorgänge im DB-Cluster erhöht. Sie können die IOPS-Nutzung mit der `VolumeWriteIOPs` CloudWatch Metrik überwachen.

Nachdem Sie die binäre Protokollierung aktiviert haben, stellen Sie sicher, dass Sie den DB-Cluster neu starten, damit Ihre Änderungen wirksam werden. Blue/green Bereitstellungen erfordern, dass die Writer-Instance mit der DB-Cluster-Parametergruppe synchronisiert ist, andernfalls schlägt die Erstellung fehl. Weitere Informationen finden Sie unter Neustarten einer DB-Instance in einem Aurora Cluster.

Darüber hinaus empfehlen wir, den Aufbewahrungszeitraum für Binärprotokolle auf einen anderen Wert als `NULL` zu ändern, um zu verhindern, dass binäre Protokolldateien gelöscht werden. Weitere Informationen finden Sie unter the section called “Festlegen und Anzeigen der Konfiguration des Binärprotokolls”.

Vorbereiten eines Aurora PostgreSQL-DB-Clusters für eine Bereitstellung blue/green

Bevor Sie eine blue/green Bereitstellung für einen Aurora PostgreSQL-DB-Cluster erstellen, stellen Sie sicher, dass Sie Folgendes tun.

- Ordnen Sie den Cluster einer benutzerdefinierten DB-Cluster-Parametergruppe zu, für die die logische Replikation (`rds.logical_replication`) aktiviert ist. Für die Replikation von der blauen zur grünen Umgebung ist die logische Replikation erforderlich.

Wenn Sie die logische Replikation aktivieren, müssen Sie auch bestimmte Cluster-Parameter wie `max_replication_slots`, `max_logical_replication_workers` und `max_worker_processes` optimieren. Anweisungen zum Aktivieren der logischen Replikation und zum Anpassen dieser Parameter finden Sie unter the section called “Einrichten der logischen Replikation”.

Stellen Sie außerdem sicher, dass der `synchronous_commit`-Parameter auf on gesetzt ist.

Nachdem Sie die erforderlichen Parameter konfiguriert haben, starten Sie den DB-Cluster neu, damit Ihre Änderungen wirksam werden. Blue/green Bereitstellungen erfordern, dass die Writer-Instance mit der DB-Cluster-Parametergruppe synchronisiert ist, andernfalls schlägt die Erstellung fehl. Weitere Informationen finden Sie unter Neustarten einer DB-Instance in einem Aurora Cluster.

- Vergewissern Sie sich, dass auf Ihrem DB-Cluster eine Version von Aurora PostgreSQL ausgeführt wird, die mit Blue/Green Deployments kompatibel ist. Eine Tabelle mit kompatiblen Versionen finden Sie unter the section called “Blau/Grün-Bereitstellungen mit Aurora PostgreSQL”.
- Stellen Sie sicher, dass alle Tabellen im DB-Cluster einen Primärschlüssel haben. Die logische PostgreSQL-Replikation lässt keine UPDATE- oder DELETE-Operationen mit Tabellen zu, die keinen Primärschlüssel haben.

Vorbereiten eines Aurora Global Database-DB-Clusters für eine blue/green Bereitstellung

Bevor Sie eine blue/green Bereitstellung für Ihren Aurora Global Database DB-Cluster erstellen, beachten Sie die folgenden Punkte:

- Alle Operationen müssen von derselben Region aus initiiert werden wie der Writer-Cluster der globalen Datenbank.
- Konfiguration der Parametergruppen:
  - Die grüne Umgebung verwendet entweder eine neue Parametergruppe, die Sie angeben, oder dieselbe Parametergruppe wie der blaue Cluster (Standard).
  - Benutzerdefinierte Parametergruppen werden in die grüne Umgebung kopiert.
  - Wenn eine angegebene Parametergruppe in der sekundären Region nicht vorhanden ist, wird die Standardparametergruppe in der sekundären Region für die grüne Umgebung verwendet.

Angabe von Änderungen bei der Erstellung einer blue/green Bereitstellung

Sie können die folgenden Änderungen am in der grünen Umgebung vornehmen, wenn Sie die blue/green Bereitstellung erstellen.

Sie können nach der Bereitstellung weitere Änderungen am DB-Cluster und seinen DB-Instances in der grünen Umgebung vornehmen. Sie können beispielsweise eine höhere Engine-Version oder eine andere Parametergruppe angeben.

Weitere Informationen zum Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Themen

- Angeben einer höheren Engine-Version
- Geben Sie eine andere DB-Parametergruppe an

Angeben einer höheren Engine-Version

Sie können eine höhere Engine-Version angeben, wenn Sie ein DB-Engine-Upgrade testen möchten. Bei der Umstellung wird die Datenbank auf die von Ihnen angegebene Haupt- oder Unterversion der DB-Engine aktualisiert.

Geben Sie eine andere DB-Parametergruppe an

Sie können eine andere DB-Cluster-Parametergruppe angeben, als vom DB-Cluster verwendet wird. Sie können testen, wie sich Parameteränderungen auf den DB-Cluster in der grünen Umgebung auswirken, oder im Falle eines Upgrades eine Parametergruppe für eine neue Hauptversion der DB-Engine angeben.

Wenn Sie eine andere DB-Cluster-Parametergruppe festlegen, wird die angegebene DB-Parametergruppe dem DB-Cluster in der grünen Umgebung zugeordnet. Wenn Sie eine andere DB-Cluster-Parametergruppe festlegen, wird der DB-Cluster in der grünen Umgebung derselben Parametergruppe zugeordnet wie der blaue DB-Cluster.

Erstellen einer Blau/Grün-Bereitstellung

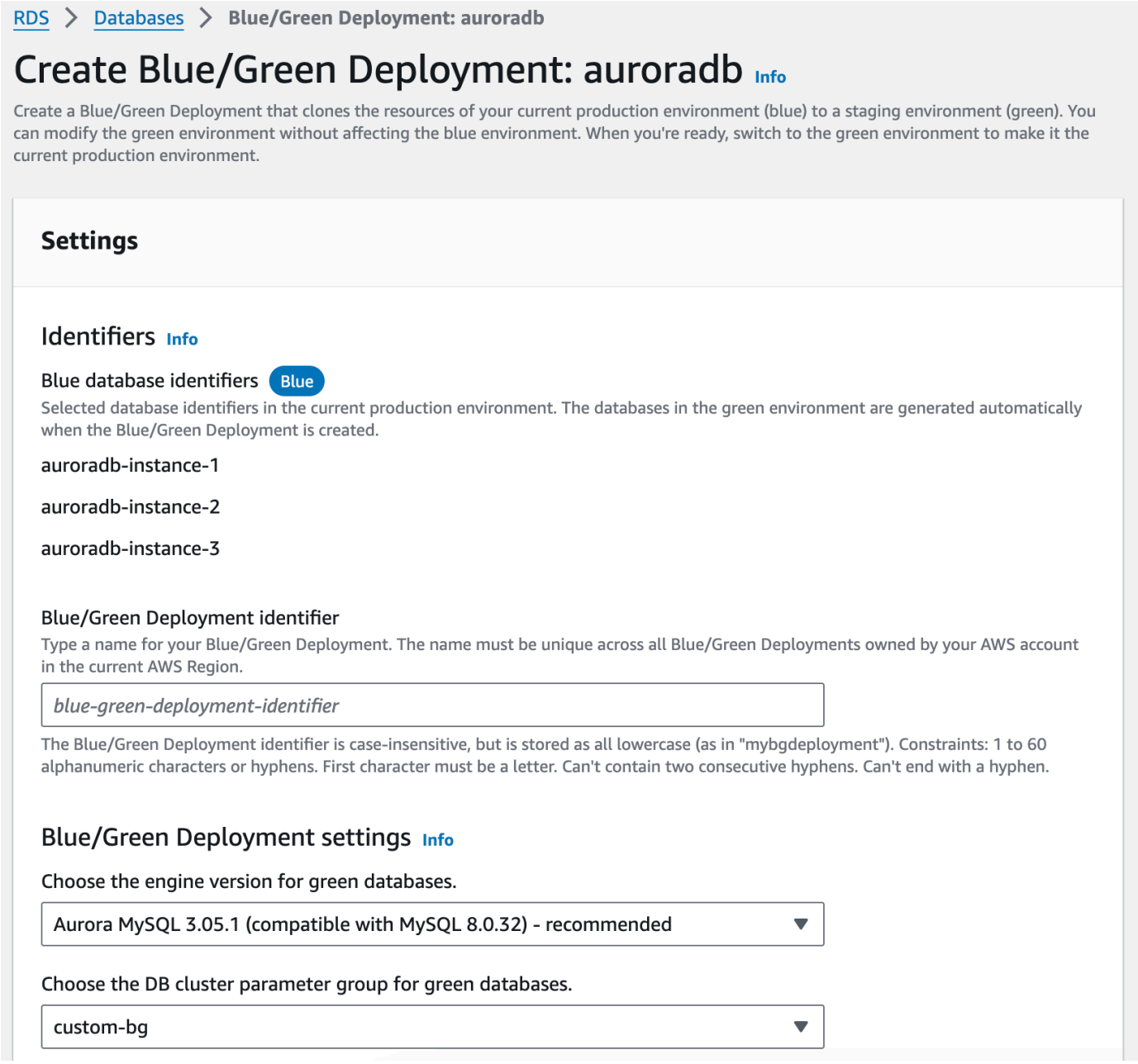
Sie können eine blue/green Bereitstellung mithilfe der AWS-Managementkonsole, der oder der AWS CLI RDS-API erstellen.

Konsole

Um eine blue/green Bereitstellung zu erstellen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den DB-Cluster zum Kopieren in eine grüne Umgebung aus.
3. Wählen Sie für Aktionen die Option Blau/Grün-Bereitstellung erstellen aus.

   Die Seite „ blue/green Bereitstellung erstellen“ wird angezeigt.



4. Überprüfen Sie die IDs der blauen Datenbanken. Stellen Sie sicher, dass sie mit den DB-Instances übereinstimmen, die Sie in der blauen Umgebung erwarten. Wenn dies nicht der Fall ist, wählen Sie Cancel (Abbrechen) aus.
5. Geben Sie für den blauen/grünen Bereitstellungsnamen einen Namen für Ihre blue/green Bereitstellung ein.
6. Geben Sie in den restlichen Abschnitten die Einstellungen für die grüne Umgebung an. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter the section called “Verfügbare Einstellungen”.

Sie können nach der Bereitstellung weitere Änderungen an den Datenbanken in der grünen Umgebung vornehmen.

7. Wählen Sie Erstellen aus.

AWS CLI

Verwenden Sie den create-blue-green-deploymentBefehl AWS CLI, um eine blue/green Bereitstellung mit dem zu erstellen. Weitere Informationen zu allen verfügbaren Aktionen finden Sie unter the section called “Verfügbare Einstellungen”.

Example

Für Linux, macOS oder Unix:

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name aurora-blue-green-deployment \
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb \
    --target-engine-version 8.0 \
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

Für Windows:

```
aws rds create-blue-green-deployment ^
    --blue-green-deployment-name aurora-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb ^
    --target-engine-version 8.0 ^
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

RDS-API

Verwenden Sie den `CreateBlueGreenDeployment`Vorgang, um eine blue/green Bereitstellung mithilfe der Amazon RDS-API zu erstellen. Informationen zu den jeweiligen Optionen finden Sie unter the section called “Verfügbare Einstellungen”.

Einstellungen für die Erstellung von Blau/Grün-Bereitstellungen

In der folgenden Tabelle werden die Einstellungen erläutert, die Sie bei der Erstellung einer blue/green Bereitstellung auswählen können. Weitere Informationen zu den AWS CLI Optionen finden Sie

unter create-blue-green-deployment. Weitere Informationen zu den RDS-API-Parametern finden Sie unter CreateBlueGreenDeployment.

Konsolene instellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter

Blau/Grün-Bereitstellungs-ID	Ein Name für die blue/green Bereitstellung.	CLI-Option:
		`--blue-green-deployment-name`
		API-Parameter:
		`BlueGreenDeploymentName`

Blaue Datenbank kennung	Die Kennung des Clusters, das Sie in die grüne Umgebung kopieren möchten. Wenn Sie die CLI oder API verwenden, geben Sie den Amazon-Ressourcennamen (ARN) des Clusters an.	CLI-Option:
		`--source`
		API-Parameter:
		`Source`

DB-Cluster-Parameter gruppe für grüne Datenbanken	Eine Parametergruppe, die den Datenbanken in der grünen Umgebung zugeordnet werden soll.	CLI-Option:
		`--target-db-cluster-parameter-group-name`
		API-Parameter:
		`TargetDBClusterParameterGroupName`

Engine-Version für grüne Datenbanken	Aktualisieren Sie das Cluster in der grünen Umgebung zur angegebenen DB-Engine-Version.
	Wenn Sie sich für einen Aurora-PostgreSQL-DB-Cluster entscheid
		CLI-Option:
		`--target-engine-version`
		RDS-API-Parameter:
		`TargetEngineVersion`

Konsolene instellung	Beschreibung der Einstellung	CLI-Option und RDS-API-Parameter
	en, überprüfen und bestätigen Sie die Einschränkungen der logischen Replikation. Weitere Informationen finden Sie unter the section called “Spezifische Einschränkungen für Bereitstellungen im Zusammenhang mit der logischen Replikation blue/green ”.	

Anzeigen einer Blau/Grün-Bereitstellung in Amazon Aurora

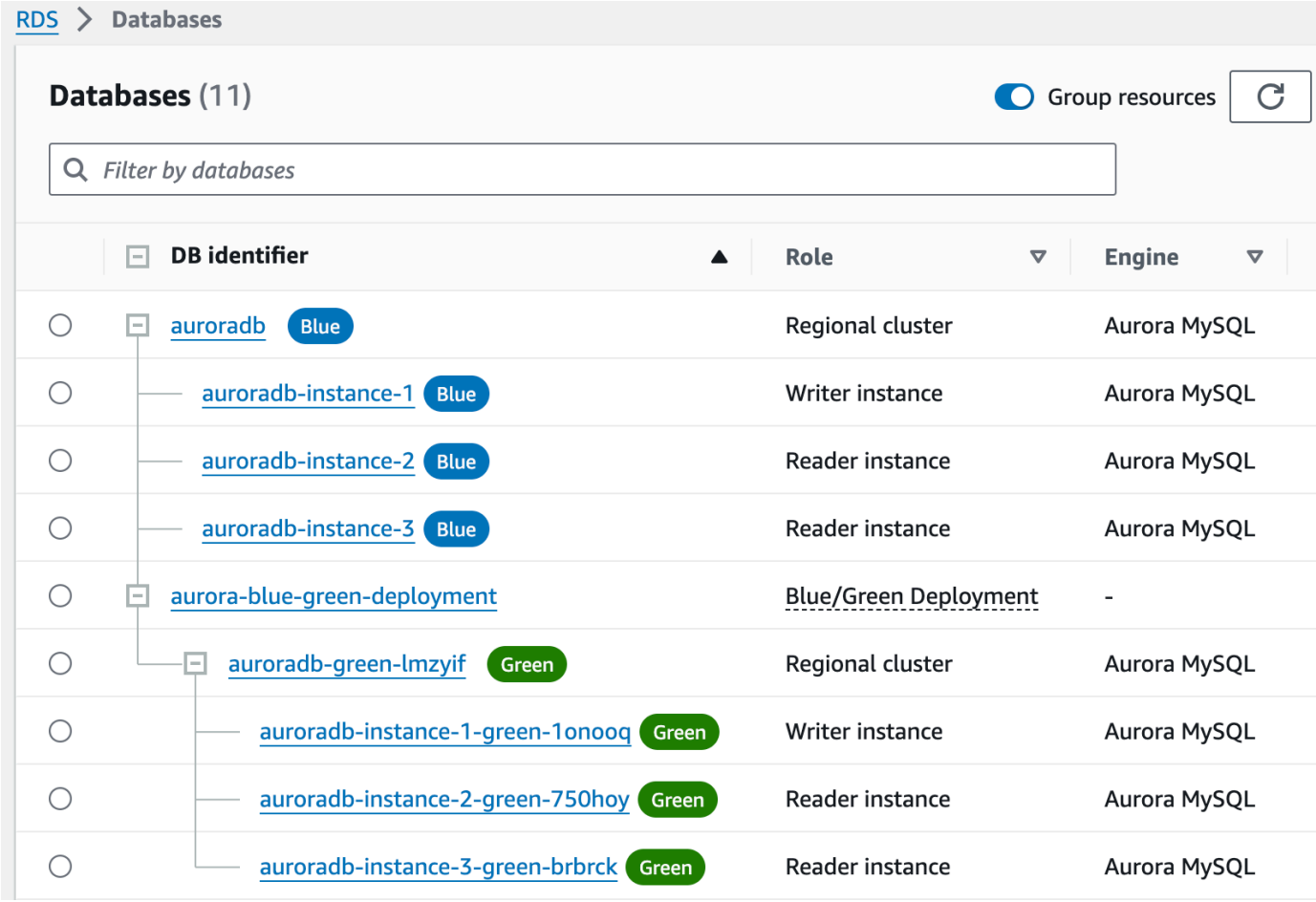
Sie können die Details einer Blau/Grün-Bereitstellung mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API anzeigen.

Außerdem können Sie Ereignisse anzeigen und abonnieren, um Informationen über eine Blau/Grün-Bereitstellung zu erhalten. Weitere Informationen finden Sie unter Blau/Grün-Bereitstellungsereignisse.

Konsole

So zeigen Sie Details über eine Blau/Grün-Bereitstellung an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus und suchen Sie dann in der Liste nach der Blau/Grün-Bereitstellung.

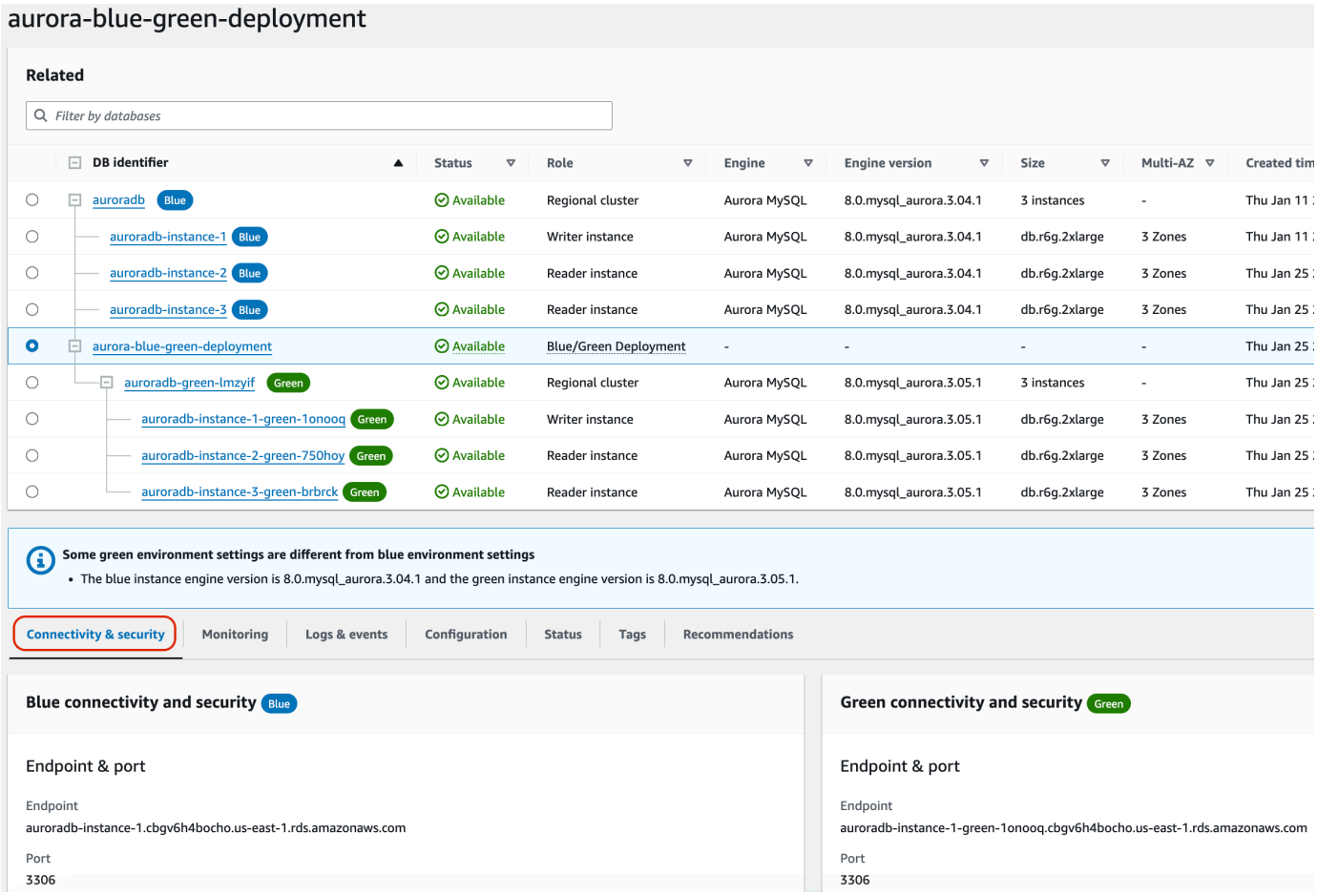


Der Wert Role (Rolle) für die Blau/Grün-Bereitstellung ist Blue/Green Deployment (Blau/Grün-Bereitstellung).

3. Wählen Sie den Namen der Blau/Grün-Bereitstellung aus, die Sie anzeigen möchten, um die Details anzeigen zu lassen.

   Jede Registerkarte verfügt über einen Abschnitt für die blaue und einen Abschnitt für die grüne Bereitstellung. Auf der Registerkarte Konfiguration kann sich die DB-Engine-Version beispielsweise in der blauen Umgebung und in der grünen Umgebung unterscheiden, wenn Sie die DB-Engine-Version in der grünen Umgebung aktualisieren.

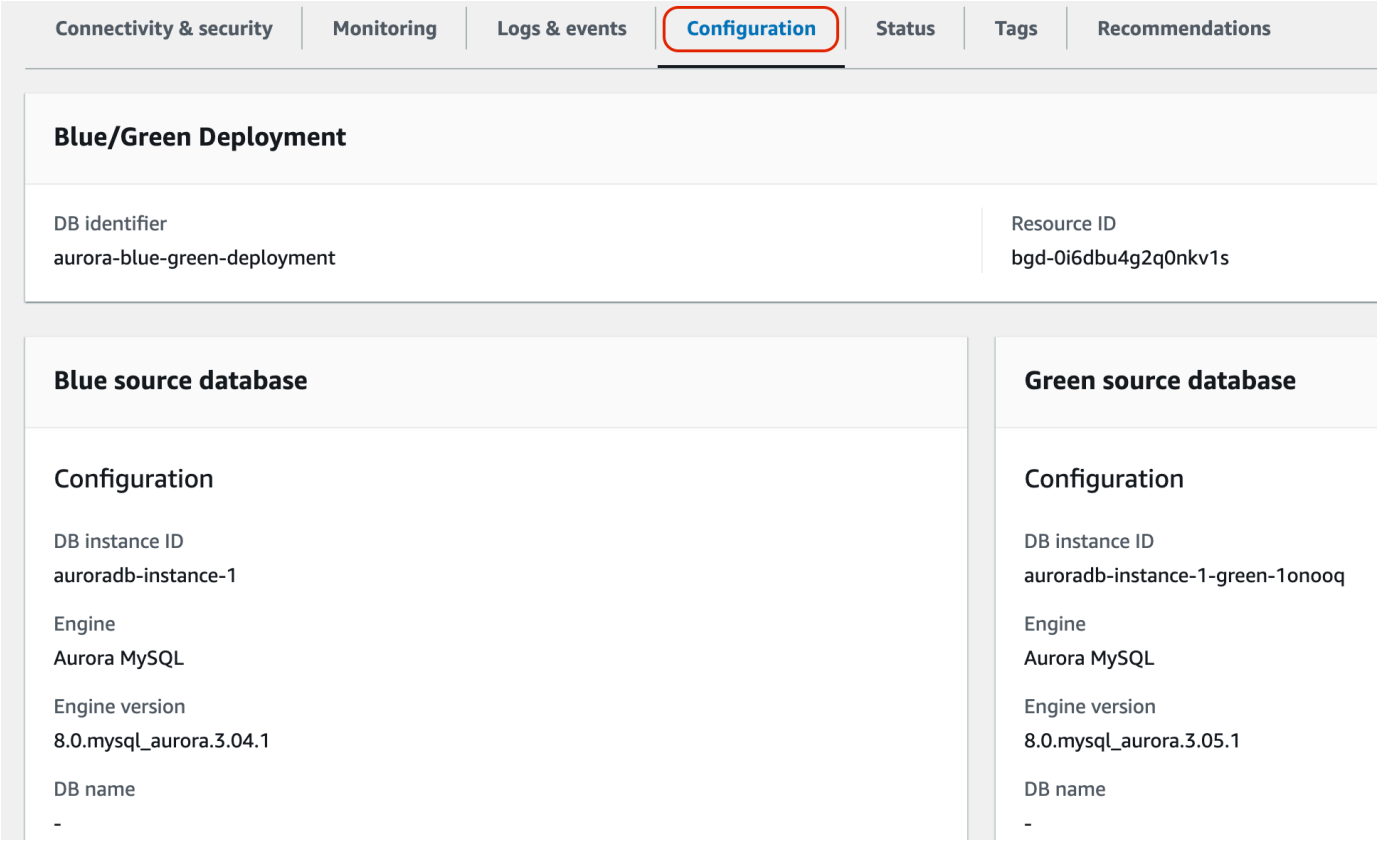
   In der folgenden Abbildung sehen Sie ein Beispiel für die Registerkarte Konnektivität und Sicherheit.



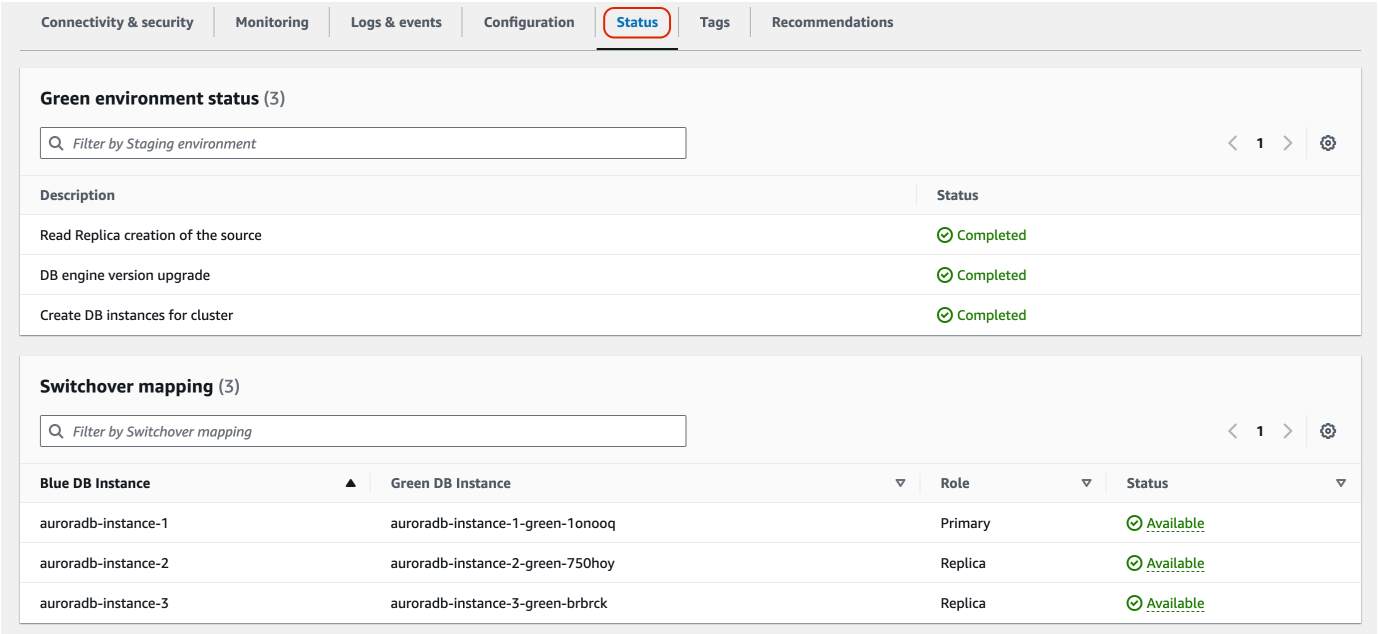
Die Registerkarte Konnektivität und Sicherheit enthält auch den Abschnitt Replikation, in dem der aktuelle Status der Replikation und die Replikatverzögerung zwischen den blauen und grünen Umgebungen angezeigt werden. Wenn der Replikationsstatus `Replicating` lautet, wird die Blau/Grün-Bereitstellung erfolgreich repliziert.

Bei Blau/Grün-Bereitstellungen von Aurora PostgreSQL kann sich der Replikationsstatus in `Replication degraded` ändern, wenn Sie in der blauen Umgebung nicht unterstützte DDL-Änderungen vornehmen oder große Objekte ändern. Weitere Informationen finden Sie unter the section called “Spezifische Einschränkungen für Bereitstellungen im Zusammenhang mit der logischen Replikation blue/green ”.

In der folgenden Abbildung sehen Sie ein Beispiel für die Registerkarte Konfiguration.



Die folgende Abbildung enthält ein Beispiel für die Registerkarte Status.



AWS CLI

Verwenden Sie den Befehl describe-blue-green-deployments, um die Details einer Blau/Grün-Bereitstellung mithilfe der AWS CLI anzuzeigen.

Example Anzeigen der Details einer Blau/Grün-Bereitstellung durch Filtern nach ihrem Namen

Wenn Sie den Befehl describe-blue-green-deployments verwenden, können Sie nach `--blue-green-deployment-name` filtern.

Das folgende Beispiel zeigt die Details für eine Blau/Grün-Bereitstellung mit dem Namen *`my-blue-green-deployment`*.

```
aws rds describe-blue-green-deployments \
  --filters Name=blue-green-deployment-name,Values=my-blue-green-deployment
```

Example Anzeigen der Details einer Blau/Grün-Bereitstellung durch Angabe der entsprechenden ID

Wenn Sie den Befehl describe-blue-green-deployments verwenden, können Sie die Option `--blue-green-deployment-identifier` angeben.

Das folgende Beispiel zeigt die Details für eine Blau/Grün-Bereitstellung mit der ID *`bgd-1234567890abcdef`*.

```
aws rds describe-blue-green-deployments \
  --blue-green-deployment-identifier bgd-1234567890abcdef
```

RDS-API

Wenn Sie die Details einer Blau/Grün-Bereitstellung mithilfe der Amazon-RDS-API anzeigen möchten, verwenden Sie den `DescribeBlueGreenDeployments`-Vorgang und geben Sie `BlueGreenDeploymentIdentifier` an.

Wechseln einer blue/green Bereitstellung in Amazon Aurora

Bei einer Umstellung wird der DB-Cluster einschließlich seiner DB-Instances in der grünen Umgebung zum Produktions-DB-Cluster gemacht. Vor der Umstellung wird der Produktionsdatenverkehr an den Cluster in der blauen Umgebung weitergeleitet. Nach der

Umstellung wird der Produktionsdatenverkehr an den DB-Cluster in der grünen Umgebung weitergeleitet.

Das Umschalten einer blue/green Bereitstellung ist nicht dasselbe wie das Heraufstufen des grünen innerhalb der blue/green Bereitstellung. Wenn Sie den DB-Cluster der grünen manuell heraufstufen, indem Sie im Menü Aktionen die Option Heraufstufen wählen, wird die Replikation zwischen der blauen und der grünen Umgebung unterbrochen und die blue/green Bereitstellung wechselt in den Status Ungültige Konfiguration.

Themen

- Umstellungs-Timeout
- Integrationsschutz der Umstellung
- Umstellungsaktionen
- Bewährte Methoden für die Umstellung
- CloudWatch Überprüfen der Metriken vor dem Switchover
- Überwachung der Replikatverzögerung vor der Umstellung
- Eine blue/green Bereitstellung umschalten
- Nach der Umstellung

Umstellungs-Timeout

Sie können einen Umstellungs-Timeout zwischen 30 Sekunden und 3 600 Sekunden (eine Stunde) festlegen. Wenn die Umstellung länger als angegeben dauert, werden alle Änderungen rückgängig gemacht und es werden keine Änderungen an einer der Umgebungen vorgenommen. Der Standardwert für den Timeout beträgt 300 Sekunden (fünf Minuten).

Integrationsschutz der Umstellung

Wenn Sie eine Umstellung starten, führt Amazon RDS einige grundlegende Prüfungen durch, um zu testen, ob die blaue und die grüne Umgebung für die Umstellung bereit sind. Diese Prüfungen werden als Integrationsschutz der Umstellung. Dieser Integrationsschutz verhindert eine Umstellung, wenn die Umgebungen dafür nicht bereit sind. Mit diesem Schutz werden längere Ausfallzeiten als erwartet vermieden und Datenverluste zwischen der blauen und der grünen Umgebung verhindern, die sich ergeben könnten, wenn die Umstellung gestartet wird.

Amazon RDS führt die folgenden Integrationsschutzprüfungen in der grünen Umgebung durch:

- Zustand der Replikation – Es wird geprüft, ob der Replikationsstatus des grünen DB-Clusters fehlerfrei ist. Der grüne DB-Cluster ist ein Replikat des blauen DB-Clusters.
- Replikationsverzögerung – Es wird geprüft, ob die Replikatverzögerung des grünen DB-Clusters innerhalb der für die Umstellung zulässigen Grenzwerte liegt. Die zulässigen Grenzwerte basieren auf dem angegebenen Timeout-Zeitraum. Die Replikatverzögerung gibt an, wie weit der grüne DB-Cluster hinter seinem blauen DB-Cluster zurückbleibt. Weitere Informationen finden Sie unter the section called “Überwachung der Replikatverzögerung vor der Umstellung”.
- Aktive Schreibvorgänge – Es wird sichergestellt, dass es auf dem grünen DB-Cluster keine aktiven Schreibvorgänge gibt.

Amazon RDS führt die folgenden Integrationsschutzprüfungen in der blauen Umgebung durch:

- Externe Replikation: stellt für Aurora PostgreSQL sicher, dass es sich bei der blauen Umgebung nicht um eine selbstverwaltete logische Quelle (Publisher) oder ein Replikat (Subscriber) handelt. Ist dies der Fall, empfehlen wir, die selbstverwalteten Replikations-Slots und Abonnements für alle Datenbanken in der blauen Umgebung zu löschen, mit der Umstellung fortzufahren und sie dann neu zu erstellen, um die Replikation fortzusetzen. Prüft für Aurora MySQL, ob die blaue Datenbank kein externes Binlog-Replikat ist. Ist dies der Fall, stellen Sie sicher, dass sie nicht aktiv repliziert wird.
- Lang andauernde aktive Schreibvorgänge – Es wird sichergestellt, dass auf dem blauen DB-Cluster keine lang andauernden aktiven Schreibvorgänge vorhanden sind, da diese die Replikatverzögerung erhöhen können.
- Lang andauernde DDL-Anweisungen – Es wird sichergestellt, dass auf dem blauen DB-Cluster keine lang andauernden DDL-Anweisungen vorhanden sind, da diese die Replikatverzögerung erhöhen können.
- Nicht unterstützte PostgreSQL-Änderungen — Bei Aurora PostgreSQL RDS for blue/green PostgreSQL-Bereitstellungen in der blauen Umgebung keine DDL-Änderungen und keine Hinzufügungen oder Änderungen großer Objekte vorgenommen wurden. Weitere Informationen finden Sie unter the section called “Spezifische Einschränkungen für Bereitstellungen im Zusammenhang mit der logischen Replikation blue/green ”.

  Wenn Amazon RDS nicht unterstützte PostgreSQL-Änderungen erkennt, ändert es den Replikationsstatus auf `Replication degraded` und benachrichtigt Sie, dass Switchover für die Bereitstellung nicht verfügbar ist. blue/green Um mit dem Switchover fortzufahren, empfehlen wir, die Bereitstellung und alle grünen Datenbanken zu löschen und neu zu erstellen. blue/green Wählen Sie dazu Aktionen, Mit grünen Datenbanken löschen aus.

Umstellungsaktionen

Wenn Sie zu einer blue/green Bereitstellung wechseln, führt RDS die folgenden Aktionen aus:

1. Es führt Integritätsschutzprüfungen durch, um zu überprüfen, ob die blaue und die grüne Umgebung bereit für die Umstellung sind.
2. Es stoppt neue Schreibvorgänge auf dem DB-Cluster in beiden Umgebungen.
3. Es trennt Verbindungen mit den DB-Instances in beiden Umgebungen und erlaubt keine neuen Verbindungen.
4. Es wartet, bis die Replikation in der grünen Umgebung aufgeholt hat, sodass die grüne Umgebung mit der blauen Umgebung synchron ist.
5. Es benennt den DB-Cluster und die DB-Instances in beiden Umgebungen um.

   RDS benennt den DB-Cluster und die DB-Instances in der grünen Umgebung um, sodass sie dem jeweiligen DB-Cluster und den DB-Instances in der blauen Umgebung entsprechen. Angenommen, der Name einer DB-Instance in der blauen Umgebung lautet `mydb`. Nehmen wir außerdem an, dass der Name der entsprechenden DB-Instance in der grünen Umgebung `mydb-green-abc123` lautet. Während der Umstellung wird der Name der DB-Instance in der grünen Umgebung in `mydb` geändert.

   RDS benennt den DB-Cluster und die DB-Instances in der blauen Umgebung um, indem `-old`*n* an den aktuellen Namen angehängt wird, wobei *n* eine Zahl ist. Angenommen, der Name einer DB-Instance in der blauen Umgebung lautet `mydb`. Nach der Umstellung könnte der Name der DB-Instance `mydb-old1` lauten.

   RDS benennt auch die Endpunkte in der grünen Umgebung um, sodass sie mit den entsprechenden Endpunkten in der blauen Umgebung übereinstimmen und keine Anwendungsänderungen erforderlich sind.
6. Erlaubt Verbindungen mit Datenbanken in beiden Umgebungen.
7. Erlaubt neue Schreibvorgänge auf dem DB-Cluster in der neuen Produktionsumgebung.

   Nach der Umstellung erlaubt der DB-Cluster der vorherigen Produktion nur Lesevorgänge. Selbst wenn Sie Schreibvorgänge auf dem DB-Cluster aktivieren, bleibt dieser schreibgeschützt, bis Sie die blue/green Bereitstellung löschen.

Sie können den Status einer Umstellung mit Amazon EventBridge überwachen. Weitere Informationen finden Sie unter the section called “Blau/Grün-Bereitstellungsereignisse”.

Beim Switchover ersetzen Tags aus der blauen Umgebung alle Tags auf Ressourcen in der grünen Umgebung. Alle Tags, die Sie direkt zu Ressourcen der grünen Umgebung hinzugefügt haben, werden während dieses Vorgangs überschrieben. Weitere Informationen zu Tags erhalten Sie unter Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen.

Wenn die Umstellung startet und aus irgendeinem Grund vorzeitig beendet wird, werden alle Änderungen rückgängig gemacht und es werden keine Änderungen an einer der Umgebungen vorgenommen.

Bewährte Methoden für die Umstellung

Wir empfehlen Ihnen dringend, sich an bewährte Methoden zu halten und vor der Umstellung die folgenden Aufgaben auszuführen:

- Testen Sie die Ressourcen in der grünen Umgebung gründlich. Stellen Sie sicher, dass sie ordnungsgemäß und effizient funktionieren.
- Überwachen Sie relevante CloudWatch Amazon-Metriken. Weitere Informationen finden Sie unter the section called “ CloudWatch Überprüfen der Metriken vor dem Switchover”.
- Ermitteln Sie den besten Zeitpunkt für die Umstellung.

  Während der Umstellung werden Schreibvorgänge in beiden Umgebungen von den Datenbanken abgeschnitten. Ermitteln Sie einen Zeitpunkt, an dem der Datenverkehr in Ihrer Produktionsumgebung am niedrigsten ist. Transaktionen mit langer Laufzeit, z. B. aktive DDLs, können Ihre Umstellungszeit verlängern, was zu längeren Ausfallzeiten Ihrer Produktionsworkloads führt.

  Wenn es eine große Anzahl von Verbindungen auf Ihren und Ihren DB-Instances gibt, sollten Sie erwägen, diese manuell auf das für Ihre Anwendung erforderliche Minimum zu reduzieren, bevor Sie die Bereitstellung umstellen. blue/green Eine Möglichkeit, dies zu erreichen, besteht darin, ein Skript zu erstellen, das den Status der blue/green Bereitstellung überwacht und mit der Bereinigung der Verbindungen beginnt, wenn es feststellt, dass sich der Status zu geändert hat`SWITCHOVER_IN_PROGRESS`.
- Stellen Sie sicher, dass der DB-Cluster und die DB-Instances in beiden Umgebungen den Status `Available` aufweisen.
- Stellen Sie sicher, dass der DB-Cluster in der grünen Umgebung fehlerfrei ist und repliziert wird.
- Stellen Sie sicher, dass Ihre Netzwerk- und Client-Konfigurationen den DNS-Cache Time-To-Live (TTL) nicht über fünf Sekunden hinaus erhöhen. Dies ist die Standardeinstellung für Aurora .

Andernfalls senden Anwendungen nach der Umstellung weiterhin Schreibdatenverkehr an die blaue Umgebung.

- Gehen Sie für Aurora blue/green PostgreSQL-Bereitstellungen, wie folgt vor:
  - Überprüfen Sie vor der Umstellung die Einschränkungen der logischen Replikation und ergreifen Sie alle erforderlichen Maßnahmen. Weitere Informationen finden Sie unter the section called “Spezifische Einschränkungen für Bereitstellungen im Zusammenhang mit der logischen Replikation blue/green ”.
  - Führen Sie die `ANALYZE`-Operation aus, um die Tabelle `pg_statistics` zu aktualisieren. Dadurch verringert sich das Risiko, dass nach der Umstellung Leistungsprobleme auftreten.

> **Note**
>
> Während einer Umstellung können Sie keinen der DB-Cluster in der Umstellung ändern.

CloudWatch Überprüfen der Metriken vor dem Switchover

Bevor Sie zu einer blue/green Bereitstellung wechseln, empfehlen wir Ihnen, die Werte der folgenden Kennzahlen bei Amazon zu überprüfen CloudWatch.

- `DatabaseConnections`— Verwenden Sie diese Kennzahl, um den Grad der Aktivität bei der blue/green Bereitstellung abzuschätzen und sicherzustellen, dass der Wert für Ihre Bereitstellung akzeptabel ist, bevor Sie umsteigen. Wenn Performance Insights aktiviert ist, ist `DBLoad` eine genauere Metrik.
- `ActiveTransactions` – Wenn `innodb_monitor_enable` in der DB-Parametergruppe für eine Ihrer DB-Instances auf `all` gesetzt ist, verwenden Sie diese Metrik, um festzustellen, ob es eine hohe Anzahl aktiver Transaktionen gibt, die die Umstellung blockieren könnten.

Weitere Informationen finden Sie unter the section called “CloudWatch Metriken für Aurora”.

Überwachung der Replikatverzögerung vor der Umstellung

Stellen Sie vor dem Umstieg auf eine blue/green Bereitstellung sicher, dass die Replikatverzögerung nahezu Null ist, um Ausfallzeiten zu reduzieren.

Aurora MySQL

Überprüfen Sie bei MySQL die `AuroraBinlogReplicaLag` CloudWatch Metrik in der grünen Umgebung, um die aktuelle Replikatverzögerung zu ermitteln. Weitere Informationen finden Sie unter the section called “Diagnose und Lösung bei Verzögerungen zwischen Read Replicas (Lesereplikaten)”.

Aurora PostgreSQL

Überprüfen Sie bei blue/green PostgreSQL-Bereitstellungen, die `OldestReplicationSlotLag` CloudWatch Metrik in der blauen Umgebung, um die aktuelle Replikatverzögerung zu ermitteln. Weitere Informationen finden Sie unter the section called “Metriken auf Instance-Ebene für Amazon Aurora”.

Darüber hinaus können Sie die folgende SQL-Abfrage in der blauen Umgebung ausführen:

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
WHERE slot_type = 'logical';

slot_name        |    flushed    | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

`confirmed_flush_lsn` stellt die letzte Log-Sequenznummer (LSN) dar, die an das Replikat gesendet wurde. `pg_current_wal_lsn` stellt dar, wo sich die Datenbank jetzt befindet. Ist `lsn_distance` 0, bedeutet das, dass das Replikat keine Verzögerung aufweist.

Eine blue/green Bereitstellung umschalten

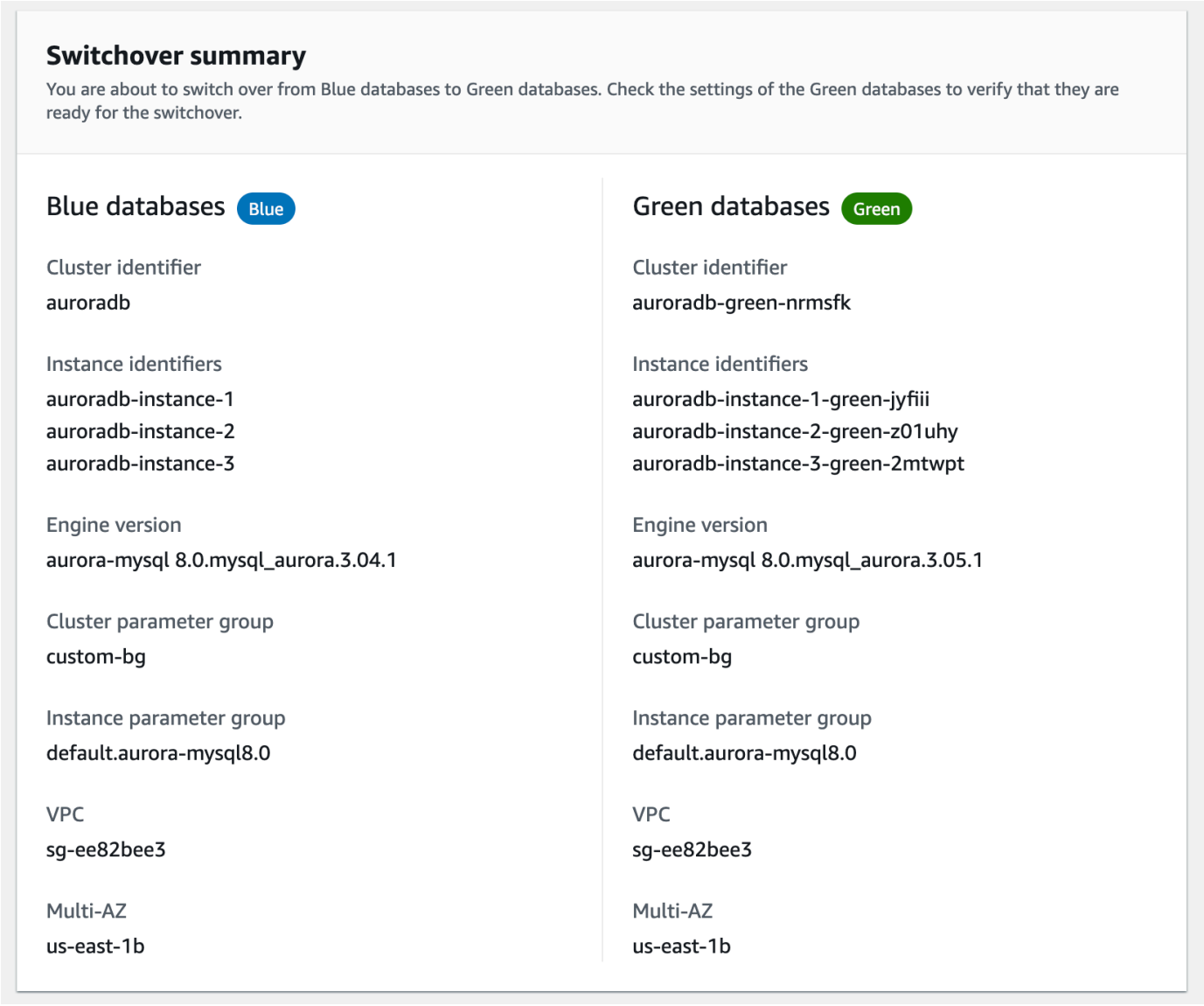
Sie können eine blue/green Bereitstellung mithilfe der AWS-Managementkonsole, der oder der AWS CLI RDS-API umschalten.

Konsole

Um eine blue/green Bereitstellung umzuschalten

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken und dann die blue/green Bereitstellung aus, zu der Sie wechseln möchten.
3. Wählen Sie unter Actions (Aktionen) die Option Switch over (Umstellen) aus.

   Die Seite Switch Over (Umstellen) wird angezeigt.



4. Sehen Sie sich auf der Seite Switchover (Umstellen) die Umstellungszusammenfassung an. Stellen Sie sicher, dass die Ressourcen in beiden Umgebungen Ihren Erwartungen entsprechen. Wenn dies nicht der Fall ist, wählen Sie Cancel (Abbrechen) aus.
5. Geben Sie unter Timeout-Einstellungen das Zeitlimit für die Umstellung ein.
6. Wenn in Ihrem Cluster Aurora PostgreSQL ausgeführt wird, überprüfen und bestätigen Sie die vor der Umstellung zu berücksichtigenden Empfehlungen. Weitere Informationen finden Sie unter the section called “Spezifische Einschränkungen für Bereitstellungen im Zusammenhang mit der logischen Replikation blue/green ”.
7. Wählen Sie Switch over (Umstellen) aus.

AWS CLI

Um eine blue/green Bereitstellung mithilfe von umzuschalten AWS CLI, verwenden Sie den switchover-blue-green-deploymentBefehl mit den folgenden Optionen:

- `--blue-green-deployment-identifier`— Geben Sie die Ressourcen-ID der blue/green Bereitstellung an.
- `--switchover-timeout` – Geben Sie das Zeitlimit für die Umstellung in Sekunden an. Der Standardwert ist 300.

Example Wechseln Sie zu einer blue/green Bereitstellung

Für Linux, macOS oder Unix:

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

Für Windows:

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

RDS-API

Um eine blue/green Bereitstellung mithilfe der Amazon RDS-API umzuschalten, verwenden Sie den SwitchoverBlueGreenDeploymentVorgang mit den folgenden Parametern:

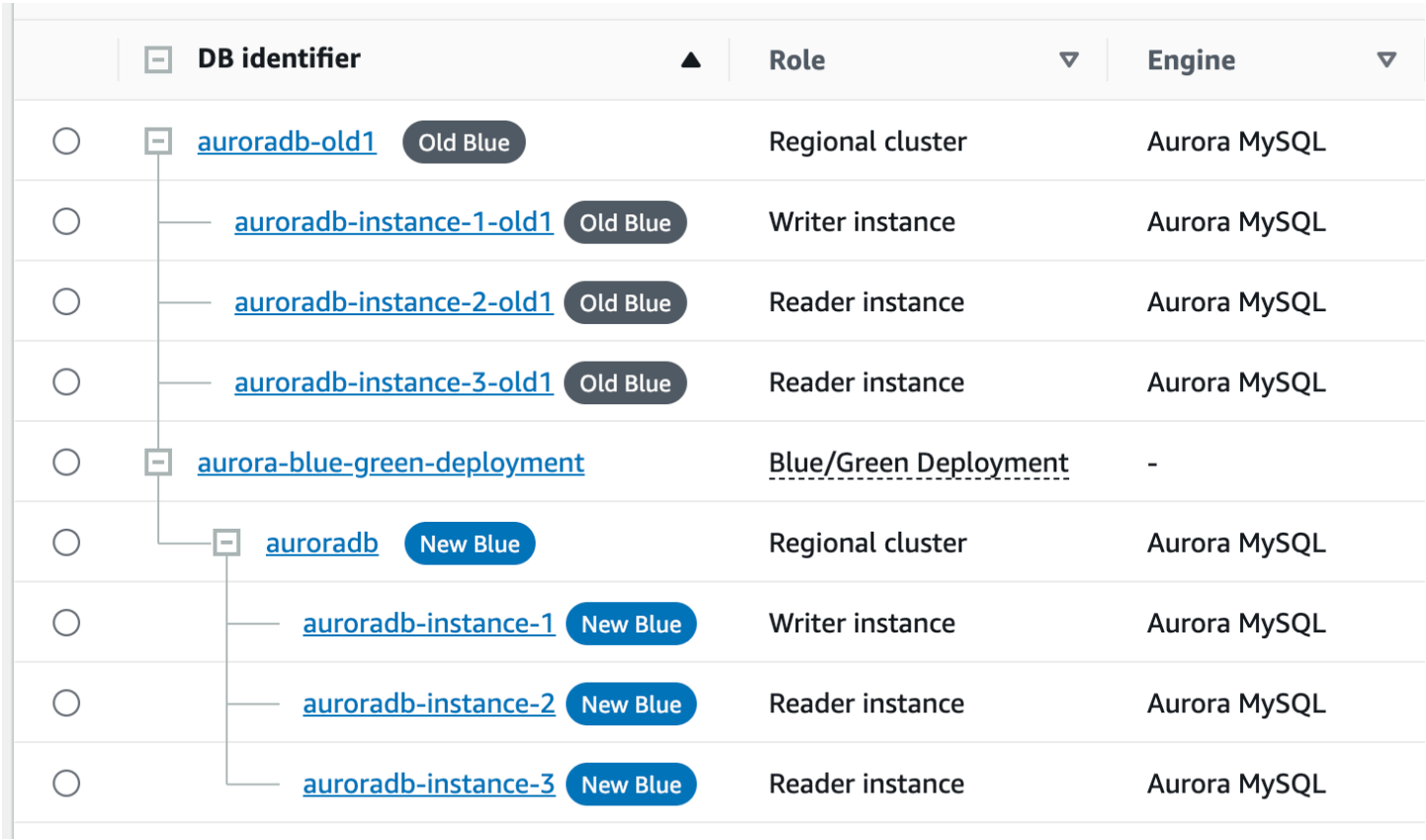
- `BlueGreenDeploymentIdentifier`— Geben Sie die Ressourcen-ID der blue/green Bereitstellung an.
- `SwitchoverTimeout` – Geben Sie das Zeitlimit für die Umstellung in Sekunden an. Der Standardwert ist 300.

Nach der Umstellung

Nach einer Umstellung werden der DB-Cluster und die DB-Instances in der vorherigen blauen Umgebung beibehalten. Für diese Ressourcen fallen die Standardkosten an. Die Replikation und die binäre Protokollierung zwischen der blauen und der grünen Umgebung werden gestoppt.

RDS benennt den DB-Cluster und die DB-Instances in der blauen Umgebung um, indem `-old`*n* an den aktuellen Namen angehängt wird, wobei *n* eine Zahl ist. Der blaue DB-Cluster wird in einen schreibgeschützten Zustand versetzt. Selbst wenn Sie Schreibvorgänge aktivieren, bleibt sie schreibgeschützt, bis Sie die blue/green Bereitstellung löschen. RDS nennt den DB-Cluster und die DB-Instances in der grünen Umgebung `-new`*n*.

Wenn Sie die blue/green Bereitstellungsressource löschen, behält RDS die Ressourcen `-old`*n* und `-new`*n* bei.



Aktualisieren des übergeordneten Knotens für Verbraucher

RDS bietet vollständig verwaltete Lesereplikate. Es bietet jedoch auch die Möglichkeit, selbstverwaltete Replikate einzurichten, die auch als externe Replikate bezeichnet werden. Mit externen Replikaten können Sie Ressourcen von Drittanbietern als Replikationsziele verwenden.

Wenn Sie eine umgestellt haben und der DB-Cluster der blauen vor dem Switchover externe Replikate oder Binärprotokollverbraucher hatte, müssen Sie deren übergeordneten Knoten nach dem Switchover aktualisieren, um die Kontinuität der Replikation aufrechtzuerhalten.

So ändern Sie den übergeordneten Knoten

1. Nach der Umstellung gibt die Writer-DB-Instance, die sich zuvor in der grünen Umgebung befand, ein Ereignis aus, das den Namen der Master-Protokolldatei und die Position des Master-Protokolls enthält. Navigieren Sie zur RDS-Konsole und wählen Sie im linken Navigationsbereich die Option Ereignisse aus, um das Ereignis ausfindig zu machen.
2. Filtern Sie vor der Umstellung nach Ereignissen, bei denen die Quelle der Name der alten grünen Writer-DB-Instance ist.
3. Suchen Sie das Ereignis, das die Koordinaten des Binärprotokolls enthält. Die Ereignismeldung sieht ungefähr so aus: `Binary log coordinates in green environment after switchover: file mysql-bin-changelog.`*`000003`*` and position `*`40134574`*`.`
4. Stellen Sie sicher, dass der Consumer oder das Replikat alle Binärprotokolle aus der alten blauen Umgebung übernommen hat. Verwenden Sie dann die bereitgestellten Koordinaten des Binärprotokolls, um die Replikation auf den Consumern fortzusetzen. Wenn Sie beispielsweise ein MySQL-Replikat auf ausführen EC2, können Sie die folgenden Befehle verwenden:

   MySQL 8.0.22 und niedrigere Haupt- und Nebenversionen

   ```
   CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-
   changelog.000003', MASTER_LOG_POS=40134574;
   ```

   MySQL 8.0.23 und höhere Haupt- und Nebenversionen

   ```
   CHANGE REPLICATION SOURCE TO SOURCE_HOST='{new-writer-endpoint}',
    SOURCE_LOG_FILE='mysql-bin-changelog.000003', SOURCE_LOG_POS=40134574;
   ```

Löschen einer Blau/Grün-Bereitstellung in Amazon Aurora

Sie können eine Blau/Grün-Bereitstellung vor oder nach der Umstellung löschen.

Wenn Sie eine Blau/Grün-Bereitstellung löschen, bevor Sie sie umstellen, löscht Amazon RDS optional die/den Cluster in der Grün-Umgebung:

- Wenn Sie den DB-Cluster in der grünen Umgebung (`--delete-target`) löschen möchten, stellen Sie sicher, dass sein Löschschutz nicht aktiviert ist.
- Wenn Sie die/den DB- Cluster in der Grün-Umgebung (`--no-delete-target`) nicht löschen, werden/wird der Cluster beibehalten, er sind/ist jedoch nicht länger Teil einer Blau-/Grün-Bereitstellung. Für Aurora MySQL wird die Replikation zwischen den Umgebungen fortgesetzt. Für Aurora PostgreSQL wird die grüne Umgebung zu einer eigenständigen Umgebung hochgestuft, sodass die Replikation gestoppt wird.

Die Option zum Löschen der grünen Datenbanken ist in der Konsole nach der Umstellung nicht verfügbar. Beim Löschen von Blau/Grün-Bereitstellungen über die AWS CLI können Sie die Option `--delete-target` nicht angeben, wenn der Bereitstellungsstatus `SWITCHOVER_COMPLETED` lautet.

> **Important**
>
> Nachdem Sie eine Blau/Grün-Bereitstellung gelöscht haben, entfernt RDS den Schreibschutz aus dem vorherigen Produktions-DB-Cluster. Wenn der Parameter `read_only` für den DB-Cluster deaktiviert ist, werden Schreibvorgänge wieder zugelassen.

Sie können eine Blau/Grün-Bereitstellung mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API löschen.

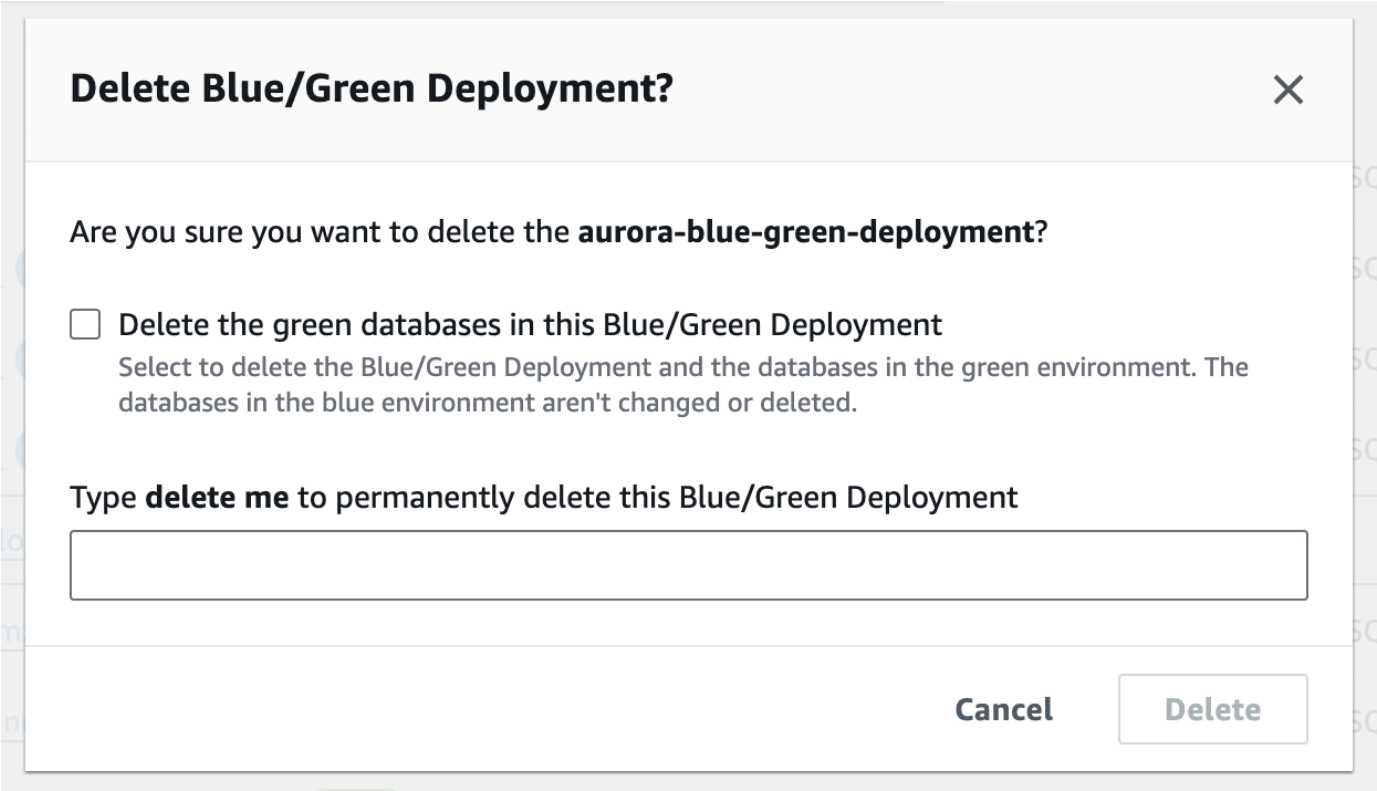
Konsole

So löschen Sie eine Blau/Grün-Bereitstellung

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die Blau/Grün-Bereitstellung aus, die Sie löschen möchten.

3. Klicken Sie bei Actions auf Delete.

   Das Fenster Blau/Grün-Bereitstellung löschen? wird angezeigt.



   Um die grünen Datenbanken zu löschen, wählen Sie Die grünen Datenbanken in dieser Blau/Grün-Bereitstellung löschen aus.
4. Geben Sie **`delete me`** in das Feld ein.
5. Wählen Sie Delete (Löschen).

AWS CLI

Verwenden Sie den Befehl delete-blue-green-deployment mit den folgenden Optionen, um eine Blau/Grün-Bereitstellung mit der AWS CLI zu löschen:

- `--blue-green-deployment-identifier` – Die Ressourcen-ID der Blau/Grün-Bereitstellung, die gelöscht werden soll.
- `--delete-target` – Gibt an, dass der DB-Cluster in der grünen Umgebung gelöscht wird. Sie können diese Option nicht angeben, wenn die Blau/Grün-Bereitstellung den Status `SWITCHOVER_COMPLETED` hat.
- `--no-delete-target` – Gibt an, dass der DB-Cluster in der grünen Umgebung beibehalten wird.

Example Löschen einer Blau/Grün-Bereitstellung und des DB-Clusters in der grünen Umgebung

Für Linux, macOS oder Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

Für Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --delete-target
```

Example Löschen einer Blau/Grün-Bereitstellung unter Beibehaltung des DB-Clusters in der grünen Umgebung

Für Linux, macOS oder Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

Für Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

RDS-API

Verwenden Sie die `DeleteBlueGreenDeployment`-Operation mit den folgenden Parametern, um mithilfe der Amazon-RDS-API eine Blau/Grün-Bereitstellung zu löschen:

- `BlueGreenDeploymentIdentifier` – Die Ressourcen-ID der Blau/Grün-Bereitstellung, die gelöscht werden soll.
- `DeleteTarget` – Geben Sie `TRUE` an, um den DB-Cluster in der grünen Umgebung zu löschen, oder `FALSE`, um ihn beizubehalten. Darf nicht `TRUE` sein, wenn die Blau/Grün-Bereitstellung den Status `SWITCHOVER_COMPLETED` hat.

Sichern und Wiederherstellen eines Amazon-Aurora-DB-Clusters

Diese Themen enthalten Informationen über Backup und Wiederherstellung von DB-Clustern von Amazon Aurora.

> **Tip**
>
> Die Hochverfügbarkeitsfunktionen und automatischen Sicherungsfunktionen von Aurora helfen, Ihre Daten zu schützen, ohne eine umfangreiche Einrichtung zu erfordern. Informieren Sie sich vor der Implementierung einer Sicherungsstrategie darüber, wie Aurora mehrere Kopien Ihrer Daten verwaltet und Ihnen hilft, über verschiedene DB-Instances und AWS-Regionen hinweg auf diese zuzugreifen. Details hierzu finden Sie unter Hohe Verfügbarkeit für Amazon Aurora.

Themen

- Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters
- Aufbewahren automatisierter Backups
- Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora
- Erstellen eines DB-Cluster-Snapshots
- Wiederherstellen aus einem DB-Cluster-Snapshot
- Kopieren eines DB-Cluster-Snapshots
- Freigeben eines DB-Cluster-Snapshots
- Exportieren von DB-Cluster-Daten nach Amazon S3
- Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3
- Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit
- Löschen eines DB-Cluster-Snapshots
- Tutorial: Wiederherstellen eines DB-Clusters von Amazon Aurora aus einem DB-Cluster-Snapshot

Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters

In den folgenden Themen werden Aurora-Backups und das Verfahren zum Wiederherstellen Ihres Aurora-DB-Clusters beschrieben.

Inhalt

- Backups
  - Benutzen AWS Backup
- Backup window
- Wiederherstellen von Daten
- Klonen von Datenbanken für Aurora
- Backtrack

Backups

Aurora sichert Ihr Cluster-Volume automatisch und bewahrt die Wiederherstellungsdaten für die Länge des Aufbewahrungszeitraums für Backups auf. Automatische Aurora-Backups sind kontinuierlich und inkrementell, sodass Sie schnell eine Wiederherstellung auf einen beliebigen Zeitpunkt im Aufbewahrungszeitraum für Backups durchführen können. Es gibt keine Auswirkungen auf die Leistungsfähigkeit oder Unterbrechung von Datenbank-Services, während Backup-Daten geschrieben werden. Sie können einen Aufbewahrungszeitraum für Backups von 1 bis 35 Tagen festlegen, wenn Sie einen DB-Cluster erstellen oder ändern. Automatische Aurora-Backups werden in Amazon S3 gespeichert. Weitere Informationen zur Beibehaltung automatisierter Backups finden Sie unter Aufbewahren automatisierter Backups.

Wenn Sie ein Backup nach dem eingestellten Aufbewahrungszeitraum behalten möchten, können Sie auch einen Snapshot der Daten Ihres Cluster-Volumes erstellen. Snapshots von Aurora-DB-Clustern laufen nicht ab. Sie können ein neues DB-Cluster aus dem Snapshot erstellen. Weitere Informationen finden Sie unter Erstellen eines DB-Cluster-Snapshots.

> **Note**
>
> - Für Amazon Aurora-DB-Cluster beträgt der Standard-Aufbewahrungszeitraum für Backups einen Tag, unabhängig davon, wie der DB-Cluster erstellt wurde.

- Automatisierte Sicherungen können in Aurora nicht deaktiviert werden. Die Aufbewahrungszeitraum von Sicherungen für Aurora wird vom DB-Cluster verwaltet.

Die Kosten für Sicherungsspeicher basieren auf dem Aufbewahrungszeitraum und der Menge der Aurora-Sicherungs- und -Snapshot-Daten. Weitere Informationen über den Speicher, der Aurora-Sicherungen und -Snapshots zugeordnet ist, finden Sie unter Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora. Weitere Informationen zu den Preisen für Aurora-Sicherungsspeicher finden Sie unter Amazon RDS für Aurora: Preise. Nachdem der Aurora-Cluster, der einem Snapshot zugeordnet ist, gelöscht wurde, werden für die Speicherung dieses Snapshots die Standardgebühren für Sicherungsspeicher für Aurora berechnet.

Benutzen AWS Backup

Sie können es verwenden AWS Backup , um Backups von Amazon Aurora Aurora-DB-Clustern zu verwalten.

Snapshots, die von verwaltet werden, AWS Backup gelten als manuelle DB-Cluster-Snapshots, werden aber nicht auf das DB-Cluster-Snapshot-Kontingent für Aurora angerechnet. Snapshots, die mit erstellt wurden, AWS Backup haben Namen mit. `awsbackup:job-`*`AWS-Backup-job-number`* Weitere Informationen AWS Backup dazu finden Sie im AWS Backup Developer Guide.

Sie können es auch AWS Backup zur Verwaltung automatisierter Backups von Amazon Aurora Aurora-DB-Clustern verwenden. Wenn Ihr DB-Cluster mit einem Backup-Plan verknüpft ist AWS Backup, können Sie diesen Backup-Plan für die point-in-time Wiederherstellung verwenden. Automatisierte (kontinuierliche) Backups, die von AWS Backup verwaltet werden, haben Namen mit `continuous:cluster-`*`AWS-Backup-job-number`*. Weitere Informationen finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit mit AWS Backup.

Backup window

Automatisierte Backups werden täglich während des bevorzugten Zeitfensters für das Backup durchgeführt. Wenn die Sicherung mehr Zeit als im Zeitfenster angegeben benötigt, wird sie nach dem Ende des Fensters weiter ausgeführt, bis sie abgeschlossen ist. Das Backup-Fenster kann sich nicht mit dem wöchentlichen Wartungsfenster für den DB-Cluster überschneiden.

Automatische Aurora-Backups sind kontinuierlich und inkrementell. Das Backup-Fenster wird aber verwendet, um ein tägliches System-Backup zu erstellen, das innerhalb des

Aufbewahrungszeitraums für Backups gespeichert wird. Sie können das Backup kopieren, um es über den Aufbewahrungszeitraum hinaus zu behalten.

> **Note**
>
> Wenn Sie mit dem einen DB-Cluster erstellen AWS-Managementkonsole, können Sie kein Backup-Fenster angeben. Sie können aber ein Backup-Fenster angeben, wenn Sie einen DB-Cluster mit der RDS-API oder AWS CLI erstellen.

Wenn Sie beim Erstellen des DB-Clusters kein bevorzugtes Backup-Fenster angeben, weist Aurora ein standardmäßiges 30-minütiges Backup-Fenster zu. Dieses Fenster wird nach dem Zufallsprinzip aus einem Zeitblock von jeweils AWS-Region 8 Stunden ausgewählt. In der folgenden Tabelle sind die Zeitblöcke für jeden AWS-Region Block aufgeführt, denen die Standard-Backup-Fenster zugewiesen werden.

Name der Region	Region	Zeitblock
USA Ost (Nord-Virginia)	us-east-1	03:00 bis 11:00 Uhr UTC
USA Ost (Ohio)	us-east-2	03:00 - 11:00 UTC
USA West (Nordkalifornien)	us-west-1	06:00 - 14:00 UTC
USA West (Oregon)	us-west-2	06:00 - 14:00 UTC
Africa (Cape Town)	af-south-1	03:00 - 11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00 - 14:00 UTC
Asien-Pazifik (Hyderabad)	ap-south-2	06:30 – 14:30 Uhr UTC
Asien-Pazifik (Jakarta)	ap-southeast-3	08:00–16:00 Uhr UTC

Name der Region	Region	Zeitblock
Asien-Pazifik (Malaysia)	ap-southeast-5	09:00–17:00 Uhr UTC
Asien-Pazifik (Melbourne)	ap-southeast-4	11:00–19:00 Uhr UTC
Asien-Pazifik (Mumbai)	ap-south-1	16:30 - 00:30 UTC
Asien-Pazifik (Neuseeland)	ap-southeast-6	13:00 - 21:00 UTC
Asia Pacific (Osaka)	ap-northeast-3	00:00 - 08:00 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00 - 21:00 UTC
Asien-Pazifik (Singapur)	ap-southeast-1	14:00 - 22:00 UTC
Asien-Pazifik (Sydney)	ap-southeast-2	12:00 - 20:00 UTC
Asien-Pazifik (Taipeh)	ap-east-2	9:00 bis 17:00 Uhr UTC
Asien-Pazifik (Thailand)	ap-southeast-7	08:00 bis 16:00 Uhr UTC
Asien-Pazifik (Tokio)	ap-northeast-1	13:00 - 21:00 UTC
Canada (Central)	ca-central-1	03:00 bis 11:00 Uhr UTC
Kanada West (Calgary)	ca-west-1	18:00 - 02:00 UTC
China (Beijing)	cn-north-1	06:00 - 14:00 UTC
China (Ningxia)	cn-northwest-1	06:00 - 14:00 UTC

Name der Region	Region	Zeitblock
Europe (Frankfurt)	eu-central-1	20:00 - 04:00 UTC
Europa (Irland)	eu-west-1	22:00 - 06:00 UTC
Europe (London)	eu-west-2	22:00 bis 06:00 Uhr UTC
Europa (Milan)	eu-south-1	02:00 - 10:00 UTC
Europa (Paris)	eu-west-3	07:29 - 14:29 UTC
Europa (Spain)	eu-south-2	02:00 - 10:00 UTC
Europe (Stockholm)	eu-north-1	23:00 - 07:00 UTC
Europa (Zürich)	eu-central-2	02:00 - 10:00 UTC
Israel (Tel Aviv)	il-central-1	03:00 bis 11:00 Uhr UTC
Mexiko (Zentral)	mx-central-1	19:00 - 03:00 UTC
Naher Osten (Bahrain)	me-south-1	06:00 - 14:00 UTC
Naher Osten (VAE)	me-central-1	05:00–13:00 UHR UTC
Südamerika (São Paulo)	sa-east-1	23:00 - 07:00 UTC
AWS GovCloud (US-Ost)	us-gov-east-1	17:00 - 01:00 UTC
AWS GovCloud (US-West)	us-gov-west-1	06:00 - 14:00 UTC

Wiederherstellen von Daten

Sie können Ihre Daten wiederherstellen, indem Sie einen neuen Aurora-DB-Cluster aus den von Aurora beibehaltenen Backup-Daten, aus einem von Ihnen gespeicherten DB-Cluster-Snapshot

oder aus einem beibehaltenen, automatisierten Backup erstellen. Sie können schnell eine neue Kopie eines DB-Clusters wiederherstellen, die aus den Sicherungsdaten von irgendeinem Zeitpunkt der Aufbewahrungsfrist für Backups erstellt wurde. Da Aurora-Backups während des Aufbewahrungszeitraums für Backups fortlaufend und ansteigend sind, müssen Sie keine häufigen Snapshots Ihrer Daten machen, um die Wiederherstellungszeiten zu optimieren.

Der späteste wiederherstellbare Zeitpunkt für einen DB-Cluster ist der aktuelle Punkt, an dem Sie Ihr DB-Cluster wiederherstellen können. Dieser liegt im Normalfall 5 Minuten vor dem aktuellen Zeitpunkt für einen aktiven DB-Cluster oder 5 Minuten nach der Cluster-Löschzeit für ein beibehaltenes, automatisiertes Backup.

Der früheste wiederherstellbare Zeitpunkt gibt an, wie weit Sie zeitlich innerhalb des Aufbewahrungszeitraums für Backups zurückgehen können, um Ihr Cluster-Volume an einem vergangen Zeitpunkt wiederherstellen zu können.

Sehen Sie sich die Werte `Latest restorable time` oder `Earliest restorable time` in der RDS-Konsole an, um den spätesten und frühesten wiederherstellbaren Zeitpunkt für einen DB-Cluster zu bestimmen. Weitere Informationen zur Anzeige dieser Werte finden Sie unter Anzeigen beibehaltener automatisierter Backups für Amazon Aurora.

Sie können sehen, wann ein Wiederherstellungsvorgang eines DB-Clusters abgeschlossen ist, indem Sie `Latest restorable time` und `Earliest restorable time`-Werte überprüfen. Die Werte geben NULL zurück, bis der Wiederherstellungsvorgang abgeschlossen ist. Sie können kein Backup oder einen Wiederherstellungsvorgang anfordern, wenn `Latest restorable time` oder `Earliest restorable time` NULL zurückgibt.

Informationen zum Wiederherstellen eines DB-Clusters zu einem bestimmten Zeitpunkt finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit.

Klonen von Datenbanken für Aurora

Sie können das Klonen von Datenbanken auch verwenden, um die Datenbanken Ihres Aurora DB-Clusters auf einen neuen DB-Cluster zu klonen, anstatt einen DB-Cluster-Snapshot wiederherzustellen. Die Klondatenbanken beanspruchen bei der Erstellung nur eine kleine Menge an zusätzlichem Speicherplatz. Die Daten werden nur als Datenänderungen entweder in den Quelldatenbanken oder in den Klondatenbanken kopiert. Sie können mehrere Klone vom selben DB-Cluster anfertigen oder zusätzliche Klone von anderen Klonen erstellen. Weitere Informationen finden Sie unter Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster.

Backtrack

Aurora MySQL unterstützt jetzt das "Zurückspulen" eines DB-Clusters auf einen bestimmten Zeitpunkt, ohne dass die Daten aus einer Sicherung wiederhergestellt werden müssen. Weitere Informationen finden Sie unter Rückverfolgen eines Aurora-DB-Clusters.

Aufbewahren automatisierter Backups

Automatisierte Backups können beim Löschen eines bereitgestellten oder Aurora Serverless v2-DB-Clusters beibehalten werden. Auf diese Weise können Sie einen DB-Cluster zu einem bestimmten Zeitpunkt innerhalb des Backup-Aufbewahrungszeitraums wiederherstellen, auch nachdem der Cluster gelöscht wurde.

Beibehaltene, automatisierte Backups enthalten System-Snapshots und Transaktionsprotokolle aus einem DB-Cluster. Sie enthalten auch DB-Cluster-Eigenschaften wie die DB-Instance-Klasse, die für die Wiederherstellung in einen aktiven Cluster erforderlich sind.

Sie können aufbewahrte Backups mit der AWS-Managementkonsole, der RDS-API und der AWS CLI wiederherstellen oder entfernen.

> **Note**
>
> Sie können keine automatisierten Backups für Aurora Serverless v1-DB-Cluster beibehalten.

Themen

- Aufbewahrungszeitraum
- Aufbewahrungskosten
- Verhindern des Löschens von automatisierten Backups
- Einschränkungen
- Anzeigen beibehaltener automatisierter Backups für Amazon Aurora
- Löschen beibehaltener automatisierter Backups für Amazon Aurora

Aufbewahrungszeitraum

Die System-Snapshots und Transaktionsprotokolle in einem beibehaltenen, automatisierten Backup laufen auf dieselbe Art wie für den Quell-DB-Cluster ab. Die Einstellungen für den Aufbewahrungszeitraum des Quell-Clusters gelten auch für die automatisierten Backups. Da keine neuen Snapshots oder Protokolle für diesen Cluster erstellt wurden, laufen die beibehaltenen, automatisierten Backups letztendlich vollständig ab. Nach Ablauf der Beibehaltungsfrist behalten Sie weiterhin manuelle DB-Cluster-Snapshots bei, aber alle automatisierten Backups laufen ab.

Sie können beibehaltene, automatisierte Backups mit der Konsole, AWS CLI oder RDS-API entfernen. Weitere Informationen finden Sie unter Löschen beibehaltener automatisierter Backups für Amazon Aurora.

Im Gegensatz zu einem beibehaltenen, automatisierten Backup läuft ein endgültiger Snapshot nicht ab. Wir empfehlen dringend das Erstellen eines finalen Snapshots, selbst wenn Sie automatische Backups aufbewahren, da automatische Backups letztendlich ablaufen.

Aufbewahrungskosten

Es fallen keine zusätzlichen Gebühren für Backup-Speicher von bis zu 100 % Ihres gesamten Aurora-Datenbankspeichers für jeden Aurora-DB-Cluster an. Außerdem fallen bis zu einem Tag keine zusätzlichen Kosten an, wenn Sie automatisierte Backups nach dem Löschen eines DB-Clusters beibehalten. Backups, die Sie länger als einen Tag beibehalten, werden in Rechnung gestellt.

Für Transaktionsprotokolle oder Instance-Metadaten wird keine zusätzliche Gebühr erhoben. Alle anderen Preisregeln für Backups gelten für wiederherstellbare Cluster. Weitere Informationen finden Sie auf der Seite Amazon-Aurora-Preise.

Verhindern des Löschens von automatisierten Backups

Amazon RDS löscht automatisierte Backups in verschiedenen Situationen:

- Am Ende des Aufbewahrungszeitraums.
- Wenn Sie einen DB-Cluster löschen.

Soll ein automatisches Backup länger aufbewahrt werden, erstellen Sie durch Kopieren einen manuellen Snapshot, der aufbewahrt wird, bis Sie ihn löschen. Amazon-RDS-Speicherkosten können für manuelle Snapshots anfallen, wenn sie Ihren Standardspeicherplatz überschreiten.

Weitere Informationen zum Kopieren eines DB-Cluster-Snapshots finden Sie unter Kopieren eines DB-Cluster-Snapshots.

Weitere Information zu Sicherungsspeicherkosten finden Sie unter Amazon RDS – Preise.

Einschränkungen

Die folgenden Einschränkungen gelten für aufbewahrte automatische Backups:

- Die maximale Anzahl der beibehaltenen automatischen Backups in einer AWS-Region ist 40. Dies ist nicht im Kontingent für DB-Cluster enthalten. Sie können bis zu 40 laufende DB-Cluster, 40 laufende DB-Instances und 40 beibehaltene, automatisierte Backups für DB-Cluster gleichzeitig haben.

  Weitere Informationen finden Sie unter Kontingente in Amazon Aurora.
- Aufbewahrte automatische Backups enthalten keine Informationen über Parameter oder Optionsgruppen.
- Sie können einen gelöschten Cluster zu jedem Zeitpunkt wiederherstellen, der zum Zeitpunkt des Löschens innerhalb der Beibehaltungsdauer liegt.
- Ein beibehaltener, automatisierter Backup kann nicht bearbeitet werden, da er aus System-Backups, Transaktionsprotokollen und den DB-Cluster-Eigenschaften besteht, die zum Zeitpunkt des Löschens des Quell-Clusters vorhanden waren.
- Die regionsübergreifende automatisierte Backup-Replikation wird für Aurora-DB-Cluster nicht unterstützt. Aurora unterstützt nicht die automatische Replikation von Snapshots und Transaktionsprotokollen in eine andere AWS-Region. Für eine regionsübergreifende Notfallwiederherstellung müssen Sie Aurora-Snapshots manuell in die gewünschte Zielregion kopieren.

Anzeigen beibehaltener automatisierter Backups für Amazon Aurora

Um Ihre beibehaltenen, automatisierten Backups anzuzeigen, wählen Sie zuerst im Navigationsbereich Automatisierte Backups aus und dann Beibehalten aus. Um einzelne Snapshots anzuzeigen, die mit einem beibehaltenen automatisierten Backup verknüpft sind, wählen Sie im Navigationsbereich Snapshot aus. Alternativ können Sie einzelne Snapshots beschreiben, die einem aufbewahrten automatischen Backup zugeordnet sind. Von da können Sie eine DB-Instance direkt aus einem dieser Snapshots wiederherstellen.

Zum Beschreiben Ihre gespeicherten automatisierten Backups mit AWS CLI, verwenden Sie einen der folgenden Befehle:

```
aws rds describe-db-cluster-automated-backups --db-cluster-resource-
id DB_cluster_resource_ID
```

Zum Beschreiben Ihrer automatischen Backups mit der RDS-API rufen Sie die `DescribeDBClusterAutomatedBackups`-Aktion mit einem der folgenden Parameter `DbClusterResourceId` auf:

Löschen beibehaltener automatisierter Backups für Amazon Aurora

Sie können aufbewahrte automatisierte Backups löschen, wenn sie nicht mehr benötigt werden. Gehen Sie wie folgt vor, um ein beibehaltenes automatisiertes Backup mithilfe der AWS-Managementkonsole, AWS CLI oder Amazon-RDS-API zu löschen.

Konsole

So löschen Sie eine aufbewahrte automatisierte Backup:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Automated backups (Automatisierte Backups) aus.
3. Wählen Sie die Registerkarte Beibehalten aus.

4. Wählen Sie die aufbewahrte automatisierte Sicherung, die Sie löschen möchten.
5. Klicken Sie bei Actions auf Delete.
6. Geben Sie auf der Bestätigungsseite **`delete me`** und wählen Sie Löschen aus.

AWS CLI

Sie können ein beibehaltenes, automatisiertes Backup mit dem AWS CLI-Befehl delete-db-cluster-automated-backup mit der folgenden Option löschen:

- `--db-cluster-resource-id` – Die Ressourcenkennung für den Quell-DB-Cluster.

  Sie finden die Ressourcenkennung für den Quell-DB-Cluster eines beibehaltenen, automatisierten Backups mit dem AWS CLI-Befehl describe-db-cluster-automated-backups.

Example

In diesem Beispiel wird das beibehaltene, automatisierte Backup für den Quell-DB-Cluster gelöscht, das die Ressourcen-ID `cluster-123ABCEXAMPLE` hat.

Für Linux, macOS oder Unix:

```
aws rds delete-db-cluster-automated-backup \
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

Für Windows:

```
aws rds delete-db-cluster-automated-backup ^
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

RDS-API

Sie können ein beibehaltenes, automatisiertes Backup mit der Amazon-RDS-API-Operation DeleteDBClusterAutomatedBackup mit dem folgenden Parameter löschen:

- `DbClusterResourceId` – Die Ressourcenkennung für den Quell-DB-Cluster.

  Sie finden die Ressourcenkennung für die Quell-DB-Instance eines beibehaltenen, automatisierten Backups mit der Amazon-RDS-API-Operation DescribeDBClusterAutomatedBackups.

Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora

Amazon Aurora verwaltet zwei Arten von Backups: automatische (kontinuierliche) Backups und Snapshots.

Automatischer Backup-Speicher

Das automatische (kontinuierliche) Backup für einen Cluster speichert alle Datenbankänderungen innerhalb eines bestimmten Aufbewahrungszeitraums inkrementell, um sie auf jeden beliebigen Zeitpunkt innerhalb dieses Aufbewahrungszeitraums wiederherstellen zu können. Die Aufbewahrungszeiträume können zwischen 1 und 35 Tagen liegen. Automatische Backups erfolgen inkrementell und werden auf der Grundlage der Speichermenge berechnet, die für eine Wiederherstellung auf einen Zeitpunkt innerhalb des Aufbewahrungszeitraums erforderlich ist.

Aurora bietet außerdem ein kostenloses Kontingent für die Backup-Nutzung an. Dieses kostenlose Nutzungskontingent entspricht der aktuellen Cluster-Volume-Größe (wie durch die Amazon-CloudWatch-Metrik `VolumeBytesUsed` dargestellt). Dieses Kontingent wird von der berechneten Nutzung automatischer Backups abgezogen. Für automatische Backups mit einem Aufbewahrungszeitraum von nur 1 Tag fallen ebenfalls keine Gebühren an.

Beispiel: Ihr automatisches Backup hat einen Aufbewahrungszeitraum von 7 Tagen und Sie möchten Ihren Cluster auf den Status von vor vier Tagen zurücksetzen. Aurora verwendet die im automatischen Backup gespeicherten inkrementellen Daten, um den Status des Clusters zu genau diesem Zeitpunkt vor vier Tagen wiederherzustellen.

Das automatische Backup speichert alle erforderlichen Informationen, um den Cluster auf jeden Zeitpunkt innerhalb des Aufbewahrungszeitraums wiederherstellen zu können. Das bedeutet, dass alle Änderungen während des Aufbewahrungszeitraums gespeichert werden, einschließlich der Schreibvorgänge zu neuen Informationen oder des Löschens vorhandener Informationen. Bei Datenbanken, in denen viele Änderungen vorgenommen werden, nimmt die Größe des automatischen Backups mit der Zeit zu. Wenn an einer Datenbank keine Änderungen mehr vorgenommen werden, können Sie damit rechnen, dass sich die Größe des automatischen Backups verringert, da die zuvor gespeicherten Änderungen im Laufe der Zeit nicht mehr innerhalb des Aufbewahrungszeitraums liegen.

Die in Rechnung gestellte Gesamtnutzung für das automatische Backup übersteigt niemals die kumulative Cluster-Volume-Größe während des Aufbewahrungszeitraums. Wenn der Aufbewahrungszeitraum beispielsweise 7 Tage beträgt und das Cluster-Volumen 100 GB pro Tag

betrug, dann übersteigt die in Rechnung gestellte Nutzung automatischer Backups niemals 700 GB (100 GB * 7).

Snapshot-Speicher

DB-Cluster-Snapshots sind immer vollständige Backups, die die Größe des Cluster-Volumes bei der Erstellung erfassen. Unabhängig davon, ob Sie Snapshots manuell oder über einen AWS Backup-Plan erstellen, behandelt Aurora sie als manuelle Snapshots. Aurora bietet unbegrenzten kostenlosen Speicherplatz für Snapshots innerhalb des Aufbewahrungszeitraums für automatische Backups. Sobald ein manueller Snapshot außerhalb dieses Zeitraums liegt, fallen Gebühren in GB/Monat an. Automatisierte System-Snapshots bleiben kostenlos, sofern Sie sie nicht kopieren. Weil automatische Backups keine Snapshot-Kopien abdecken, werden sie von AWS immer in Rechnung gestellt.

Allgemeine Informationen zu Aurora-Sicherungen finden Sie unter Backups. Weitere Informationen zu den Preisen von Backup-Speicher von Aurora finden Sie auf der Webseite Amazon Aurora – Preise.

Amazon-CloudWatch-Metriken für Aurora-Backup-Speicher

Sie können Ihre Aurora-Cluster überwachen und mit Amazon-CloudWatch-Metriken über die CloudWatch Konsole Berichte erstellen. Mit den folgenden CloudWatch-Metriken können Sie die Speicherkapazität Ihrer Aurora-Backups überprüfen und überwachen. Diese Metriken werden für jeden Aurora-DB-Cluster unabhängig voneinander berechnet.

- `BackupRetentionPeriodStorageUsed` – Stellt die Menge des Backup-Speichers in Bytes dar, die zurzeit zum Speichern automatischer Backups verwendet wird.
  - Der Wert hängt von der Größe des Cluster-Volumes und der Anzahl der Änderungen (Schreibvorgänge und Aktualisierungen) ab, die während des Aufbewahrungszeitraums am DB-Cluster vorgenommen werden. Dies liegt daran, dass das automatische Backup alle inkrementellen Änderungen am Cluster speichern muss, um eine Wiederherstellung auf einen beliebigen Zeitpunkt vornehmen zu können.
  - Diese Metrik zieht nicht das kostenlose Kontingent an Backup-Nutzung ab, das Aurora bereitstellt.
  - Diese Metrik gibt einen einzigen täglichen Datenpunkt für die an diesem Tag aufgezeichnete Nutzung automatischer Backups aus.
- `SnapshotStorageUsed` – Stellt die Menge des Backup-Speichers in Bytes dar, die zum Speichern manueller Snapshots über den Aufbewahrungszeitraum des automatischen Backups hinaus verwendet wird.

- Der Wert hängt von der Anzahl der Snapshots, die Sie über den Aufbewahrungszeitraum des automatischen Backups hinaus aufbewahren, und von der Größe der einzelnen Snapshots ab.
- Die Größe einzelner Snapshots entspricht der Größe des Cluster-Volumes zum Zeitpunkt der Snapshot-Erstellung.
- Snapshots sind vollständige Backups, keine inkrementellen.
- Diese Metrik gibt für jeden Snapshot, der in Rechnung gestellt wird, einen täglichen Datenpunkt aus. Verwenden Sie zum Abrufen Ihrer täglichen Gesamtnutzung von Snapshots die Summe dieser Metrik über einen Zeitraum von 1 Tag.

- `TotalBackupStorageBilled` – Stellt die Metriken für die gesamte in Rechnung gestellte Backup-Nutzung in Byte für den angegebenen Cluster dar:

  ```
  BackupRetentionPeriodStorageUsed + SnapshotStorageUsed - free tier
  ```

  - Diese Metrik gibt einen täglichen Datenpunkt für den Wert `BackupRetentionPeriodStorageUsed` abzüglich der von Aurora bereitgestellten kostenlosen Backup-Nutzungskontingente aus. Dieses kostenlose Kontingent entspricht der zuletzt aufgezeichneten Größe des DB-Cluster-Volumes. Dieser Datenpunkt stellt die tatsächlich in Rechnung gestellte Nutzung für das automatische Backup dar.
  - Diese Metrik gibt einzelne tägliche Datenpunkte für alle `SnapshotStorageUsed`-Werte aus.
  - Verwenden Sie zum Abrufen Ihrer täglich in Rechnung gestellten Backup-Nutzung die Summe dieser Metrik über einen Zeitraum von 1 Tag. Dadurch wird die vollständige in Rechnung gestellte Snapshot-Nutzung mit der in Rechnung gestellten automatischen Backup-Nutzung addiert, sodass sich Ihre gesamte in Rechnung gestellte Backup-Nutzung ergibt.

Weitere Informationen zur Verwendung der CloudWatch-Metriken finden Sie unter Verfügbarkeit von Aurora Metriken in der Amazon RDS Konsole.

Berechnung der Nutzung des Backup-Speichers

Die Nutzung für ein automatisches Backup wird anhand aller inkrementellen Datensätze berechnet, die gespeichert werden müssen, um eine Wiederherstellung auf einen beliebigen Zeitpunkt innerhalb des Aufbewahrungszeitraums des Backups vornehmen zu können. Zu diesen Änderungen gehören nicht nur die Anzahl der Schreibvorgänge, sondern auch die Größe und der Umfang der Datenänderungen. Bei jeder Art von Operation (INSERT, UPDATE, DELETE) werden Änderungsdatensätze erstellt, die für die zeitpunktbezogene Wiederherstellung beibehalten werden müssen. Daher können zwei Datenbanken zwar dieselbe Anzahl von Schreibvorgängen (IOPS)

aufweisen, ihre Speicheranforderungen für Backups können sich jedoch je nach der Datenmenge, die bei jeder Transaktion geändert wird, erheblich unterscheiden.

Beispiel: Sie haben ein automatisches Backup mit einem Aufbewahrungszeitraum von 7 Tagen. Die Größe Ihres Cluster-Volumes unmittelbar vor dem Aufbewahrungszeitraum betrug 100 GB. Das ist also die geringste Menge, die Aurora speichern muss. Dann haben Sie die folgende Aktivität für die nächsten 7 Tage. Dabei ist die inkrementelle Datensatzgröße die Menge an Speicherplatz, die zum Speichern der Änderungsdatensätze benötigt wird, die sich aus den Schreibvorgängen und Aktualisierungen Ihrer Datenbank ergeben.

Tag	Inkrementelle Datensatzgröße (GB)
1	10
2	15
3	25
4	20
5	10
6	25
7	30
Gesamt	135

Diese Daten bedeuten, dass die berechnete automatische Backup-Nutzung für Ihr Backup wie folgt aussieht:

```
100 GB (volume size before retention period) + 135 GB (size of incremental records) =
 235 GB total backup usage
```

Von der in Rechnung gestellten Nutzung wird dann das kostenlose Nutzungskontingent abgezogen. Angenommen, die aktuelle Größe Ihres Volumes beträgt 200 GB:

```
235 GB total backup usage - 200 GB (latest volume size) = 35 GB billed backup usage
```

Häufig gestellte Fragen

Wann werden mir Snapshots in Rechnung gestellt?

Manuelle Snapshots, die außerhalb des Aufbewahrungszeitraums des automatisierten Backups liegen (d. h. älter sind), werden Ihnen in Rechnung gestellt.

Was ist ein manueller Snapshot?

Ein manueller Snapshot ist ein Snapshot, der eine der folgenden Bedingungen erfüllt:

- Manuelle Anforderung durch Sie
- Erstellung durch einen automatischen Backup-Service wie beispielsweise AWS Backup
- Kopiert aus einem automatischen System-Snapshot, um ihn über den Aufbewahrungszeitraum hinaus aufzubewahren

Was geschieht mit meinen manuellen Snapshots, wenn ich meinen DB-Cluster lösche?

Manuelle Snapshots laufen erst ab, wenn Sie sie löschen.

Wenn Sie Ihren DB-Cluster löschen, sind die zuvor erstellten manuellen Snapshots weiterhin vorhanden. Wenn diese Snapshots zuvor nicht in Rechnung gestellt wurden, da sie sich innerhalb des Aufbewahrungszeitraums für automatische Backups befanden, sind sie jetzt nicht mehr abgedeckt und die Nutzung wird voll in Rechnung gestellt.

Wie kann ich meine Backup-Speicherkosten senken?

Es gibt mehrere Möglichkeiten, die mit der Backup-Nutzung verbundenen Kosten zu senken:

- Löschen Sie manuelle Snapshots, die außerhalb des Aufbewahrungszeitraums Ihres automatischen Backups liegen. Dies umfasst die von Ihnen erstellten Snapshots sowie die Snapshots, die Ihr AWS-Backup-Plan möglicherweise erstellt hat. Überprüfen Sie unbedingt Ihren AWS-Backup-Plan, um sicherzustellen, dass nicht versehentlich Snapshots außerhalb des Aufbewahrungszeitraums aufbewahrt werden.
- Prüfen Sie Ihre Schreibvorgänge und Aktualisierungen an Ihrer Datenbank, um zu ermitteln, ob Sie die Anzahl der von Ihnen vorgenommenen Änderungen reduzieren können. Da das automatische Backup alle inkrementellen Änderungen innerhalb des Aufbewahrungszeitraums speichert, hat die Reduzierung der Anzahl der von Ihnen vorgenommenen Aktualisierungen auch geringere Kosten für automatische Backups zur Folge.
- Prüfen Sie, ob es sinnvoll wäre, die Aufbewahrungsdauer des automatischen Backups zu reduzieren. Ein kürzerer Aufbewahrungszeitraum bedeutet, dass das Backup weniger Tage an

inkrementellen Daten speichert. Dies könnte geringere Gesamtkosten für das Backup zur Folge haben. Wenn Sie diesen Aufbewahrungszeitraum verkürzen, könnte dies jedoch auch dazu führen, dass einige Snapshots plötzlich in Rechnung gestellt werden, da sie sich jetzt außerhalb des Aufbewahrungszeitraums befinden. Überprüfen Sie unbedingt alle zusätzlichen Snapshot-Kosten, die möglicherweise anfallen, bevor Sie entscheiden, ob dies die richtige Option für Sie ist.

Wie wird Backup-Speicher abgerechnet?

Die Fakturierung für Backup-Speicher erfolgt nach GB/Monat.

Dies bedeutet, dass die Nutzung des Backup-Speichers als gewichteter Durchschnitt der Nutzung im angegebenen Monat berechnet wird. Im Folgenden finden Sie ein paar Beispiele für einen Monat mit 30 Tagen:

- Die in Rechnung gestellte Backup-Nutzung beträgt 100 GB für alle 30 Tage des Monats. Folgendes wird Ihnen in Rechnung gestellt:

```
(100 GB * 30) / 30 = 100 GB-month
```

- Die in Rechnung gestellte Backup-Nutzung beträgt 100 GB für die ersten 15 Tage des Monats, danach 0 GB für die restlichen 15 Tage. Folgendes wird Ihnen in Rechnung gestellt:

```
(100 GB * 15 + 0 GB * 15) / 30 = 50 GB-month
```

- Die in Rechnung gestellte Backup-Nutzung beträgt 50 GB für die ersten 10 Tage des Monats, 100 GB für die nächsten 10 Tage und 150 GB für die letzten 10 Tage. Folgendes wird Ihnen in Rechnung gestellt:

```
(50 GB * 10 + 100 GB * 10 + 150 GB * 10) / 30 = 100 GB-month
```

Wie wirkt sich die Rückverfolgungseinstellung für meinen DB-Cluster auf die Nutzung des Backup-Speichers aus?

Die Rückverfolgungseinstellung für einen Aurora-DB-Cluster wirkt sich nicht auf das Volumen der Sicherungsdaten für den entsprechenden Cluster aus. Der Speicher für Rückverfolgungsdaten wird von Amazon separat in Rechnung gestellt. Informationen zur Preisgestaltung von Aurora-Rückverfolgung finden Sie auf der Seite Amazon Aurora-Preisliste.

Inwiefern gelten die Speicherkosten für freigegebene Snapshots?

Wenn Sie einen Snapshot für einen anderen Benutzer freigeben, bleiben Sie der Besitzer dieses Snapshots. Die Speicherkosten gelten für den Snapshot-Besitzer. Wenn Sie einen freigegebenen Snapshot in Ihrem Besitz löschen, besteht kein Zugriff darauf mehr.

Um Zugriff auf einen freigegebenen Snapshot zu behalten, der im Besitz eines anderen Benutzers ist, können Sie den entsprechenden Snapshot kopieren. Dadurch werden Sie zum Besitzer des neuen Snapshots. Etwaige Speicherkosten für den kopierten Snapshot werden Ihrem Konto belastet.

Weitere Informationen zum Freigeben von Snapshots finden Sie unter Freigeben eines DB-Cluster-Snapshots. Weitere Informationen zum Kopieren von Snapshots finden Sie unter Kopieren eines DB-Cluster-Snapshots.

Erstellen eines DB-Cluster-Snapshots

Amazon RDS erstellt einen Snapshot für das Volume mit Ihrem DB-Cluster, damit der gesamte DB-Cluster gesichert wird und nicht nur einzelne Datenbanken. Beim Erstellen eines DB-Cluster-Snapshots wählen Sie den DB-Cluster aus, der gesichert werden soll, und benennen den DB-Cluster-Snapshot, damit Sie später mit diesem eine Wiederherstellung ausführen können. Die Zeit, die für die Erstellung eines DB-Cluster-Snapshots benötigt wird, hängt von der Größe Ihrer Datenbanken ab. Da der Snapshot das gesamte Speichervolume umfasst, wirkt sich die Größe von Dateien, wie z. B. temporäre Dateien, auch auf die Zeit aus, die zum Erstellen des Snapshots benötigt wird.

> **Note**
>
> Ihr DB-Cluster muss den Status `available` aufweisen, um einen DB-Cluster-Snapshot zu erstellen.

Im Gegensatz zu automatisierten Backups unterliegen manuelle Snapshots nicht dem Aufbewahrungszeitraum für Backups. Snapshots laufen nicht ab.

Für sehr langfristige Sicherungen empfehlen wir den Export von Snapshot-Daten in Amazon S3. Wenn die Hauptversion der DB-Engine nicht mehr unterstützt wird, können Sie diese Version nicht über einen Snapshot wiederherstellen. Weitere Informationen finden Sie unter Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3.

Sie können einen DB-Cluster-Snapshot mit der AWS-Managementkonsole, der AWS CLI oder der RDS-API erstellen.

Konsole

So erstellen Sie einen DB-Cluster-Snapshot

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.

   Die Liste manuellen Snapshots wird angezeigt.
3. Wählen Sie Take Snapshot (Snapshot erstellen) aus.

   Das Fenster Take DB Snapshot (DB-Snapshot erstellen) wird angezeigt.

4. Wählen Sie als Snapshot-Typ die Option DB-Cluster aus.
   Nehmen Sie einen Snapshot auf.
5. Wählen Sie den DB-Cluster aus, für den Sie einen Snapshot erstellen möchten.
6. Geben Sie den Snapshot-Namen ein.
7. Wählen Sie Snapshot erstellen aus.

   Die Liste Manuelle Snapshots wird angezeigt, wobei der Status des neuen DB-Cluster-Snapshots als `Creating` angezeigt wird. Nachdem sein Status`Available`, können Sie die Erstellungszeit sehen.

AWS CLI

Beim Erstellen eines DB-Cluster-Snapshots mithilfe der AWS CLI wählen Sie den DB-Cluster aus, der gesichert werden soll, und benennen den DB-Cluster-Snapshot, damit Sie später mit diesem eine Wiederherstellung ausführen können. Dies können Sie über dieAWS CLI `create-db-cluster-snapshot`-Befehl mit den folgenden Parametern:

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

In diesem Beispiel erstellen Sie einen DB-Cluster-Snapshot mit dem Namen *`mydbclustersnapshot`* für einen DB-Cluster namens *`mydbcluster`*.

Example

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mydbcluster \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Für Windows:

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS-API

Beim Erstellen eines DB-Cluster-Snapshots mithilfe der Amazon-RDS-API wählen Sie den DB-Cluster aus, der gesichert werden soll, und benennen den DB-Cluster-Snapshot, damit Sie später mit diesem eine Wiederherstellung ausführen können. Sie können dies tun, indem Sie den Amazon RDS API `CreateDBClusterSnapshot`-Befehl mit den folgenden Parametern verwenden:

- DBClusterIdentifier
- DBClusterSnapshotIdentifier

feststellen, ob der DB-Cluster-Snapshot verfügbar ist

Sie können überprüfen, ob der DB-Cluster-Snapshot verfügbar ist, indem Sie auf der Detailseite für den Cluster in der AWS-Managementkonsole unter Snapshots auf der Registerkarte Maintenance & backups (Wartung und Sicherungen), mithilfe des CLI-Befehls `describe-db-cluster-snapshots`oder mithilfe der API-Aktion `DescribeDBClusterSnapshots` nachsehen.

Sie können den CLI-Befehl `wait db-cluster-snapshot-available` auch verwenden, um die API alle 30 Sekunden abzufragen, bis der Snapshot verfügbar ist.

Wiederherstellen aus einem DB-Cluster-Snapshot

Amazon RDS erstellt einen Snapshot für das Volume mit Ihrem DB-Cluster, damit der gesamte DB-Cluster gesichert wird und nicht nur einzelne Datenbanken. Sie können einen neuen DB-Cluster erstellen, indem Sie einen DB-Snapshot wiederherstellen. Sie geben den Namen des DB-Cluster-Snapshot an, aus dem die Wiederherstellung gestartet werden soll, und anschließend einen Namen für den neuen DB-Cluster, der bei dieser Wiederherstellung erstellt wird. Sie können keine Wiederherstellung aus einem DB-Cluster-Snapshot auf einem bestehenden DB-Cluster vornehmen. Bei der Wiederherstellung wird ein neuer DB-Cluster erstellt.

Important

Sie können einen Snapshot nicht in einer DB-Engine-Version wiederherstellen, deren Aurora-Standard-Supportdatum überschritten wurde. Sie können erst auf eine Datenbank zugreifen, nachdem sie erfolgreich auf eine unterstützte Version aktualisiert wurde. Weitere Informationen zu unterstützten Versionen der Aurora-DB-Engine finden Sie unterAmazon RDS Extended Support mit Amazon Aurora.
Wenn ein Upgrade auf eine unterstützte Version für Ihren Cluster fehlschlägt, ändert sich der Cluster-Status in `upgrade_failed` und Aurora erstellt einen endgültigen Snapshot mit dem Präfix`rds-final`. Wenn Sie nach einem Upgrade-Fehler auf Ihre wiederhergestellte Datenbank in der veralteten Version zugreifen möchten, wenden Sie sich AWS an den Support.

Sie können den wiederhergestellten DB-Cluster nutzen, sobald sein Status `available` lautet.

Sie können es verwenden CloudFormation , um einen DB-Cluster aus einem DB-Cluster-Snapshot wiederherzustellen. Weitere Informationen finden Sie unter AWS: :RDS:: DBCluster im AWS CloudFormation Benutzerhandbuch.

Note

Durch die gemeinsame Nutzung eines manuellen DB-Cluster-Snapshots, ob verschlüsselt oder unverschlüsselt, können autorisierte AWS Konten einen DB-Cluster direkt aus dem Snapshot wiederherstellen, anstatt eine Kopie davon zu erstellen und von dort aus wiederherzustellen. Weitere Informationen finden Sie unter Freigeben eines DB-Cluster-Snapshots.

Informationen zur Wiederherstellung eines Aurora-DB-Clusters oder eines globalen Clusters mit einer RDS-Extended-Support-Version finden Sie unter Wiederherstellen eines Aurora-DB-Clusters oder eines globalen Clusters mit Amazon RDS Extended Support.

Überlegungen zu Parametergruppen

Wir empfehlen Ihnen, die DB-Parametergruppe und DB-Cluster-Parametergruppe für alle DB-Cluster-Snapshots aufzubewahren, die Sie erstellen, damit Sie Ihrem wiederhergestellten DB-Cluster die korrekte Parametergruppen zuordnen können.

Die Standard-DB-Parametergruppe und DB-Cluster-Parametergruppe sind dem wiederhergestellten Cluster zugeordnet, es sei denn, Sie wählen eine andere Instance aus. In den Standard-DB-Parametergruppen sind keine benutzerdefinierten Parametereinstellungen verfügbar.

Sie können die Parametergruppen angeben, wenn Sie den DB-Cluster wiederherstellen.

Weitere Informationen über DB-Parametergruppen und DB-Cluster-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Überlegungen zu Sicherheitsgruppen

Wenn Sie einen DB-Cluster wiederherstellen, werden die Standard-VPC (Virtual Private Cloud), DB-Subnetzgruppe und VPC-Sicherheitsgruppe mit der wiederhergestellten Instance verknüpft, wenn Sie keine anderen angeben.

- Wenn Sie die Amazon-RDS-Konsole verwenden, können Sie eine benutzerdefinierte VPC-Sicherheitsgruppe angeben, die dem Cluster zugeordnet werden soll, oder eine neue VPC-Sicherheitsgruppe erstellen.
- Wenn Sie die verwenden AWS CLI, können Sie eine benutzerdefinierte VPC-Sicherheitsgruppe angeben, die dem Cluster zugeordnet werden soll, indem Sie die `--vpc-security-group-ids` Option in den `restore-db-cluster-from-snapshot` Befehl aufnehmen.
- Wenn Sie die Amazon-RDS-API verwenden, können sie den `VpcSecurityGroupIds.VpcSecurityGroupId.N`-Parameter in der Aktion `RestoreDBClusterFromSnapshot` einschließen.

Sobald die Wiederherstellung abgeschlossen ist und Ihr neuer DB-Cluster verfügbar ist, können Sie auch die VPC-Einstellungen ändern, indem Sie den DB-Cluster bearbeiten. Weitere Informationen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Überlegungen zu Amazon Aurora

Mit Aurora stellen Sie einen DB-Cluster-Snapshot in einem DB-Cluster wieder her.

Sie können sowohl mit Aurora MySQL als auch mit Aurora PostrgreSQL einen DB-Cluster-Snapshot in einem Aurora Serverless-DB-Cluster wiederherstellen. Weitere Informationen finden Sie unter Wiederherstellen eines Aurora Serverless v1-DB-Clusters.

Mit Aurora MySQL stellen Sie einen DB-Cluster-Snapshot von einem Cluster ohne parallele Abfrage auf einen Cluster mit paralleler Abfrage wieder her. Da die parallele Abfrage normalerweise bei sehr großen Tabellen verwendet wird, ist die Snapshot-Methode die schnellste Art, um große Datenmengen in einen parallelen und abfragefähigen Aurora MySQL-Cluster aufzunehmen. Weitere Informationen finden Sie unter Parallel Query für Amazon Aurora MySQL.

Wiederherstellung aus einem Snapshot

Sie können einen DB-Cluster aus einem DB-Cluster Snapshot wiederherstellen, indem Sie die AWS-Managementkonsole, die AWS CLI oder die RDS API verwenden.

Konsole

So stellen Sie einen DB-Cluster aus einem DB-Cluster-Snapshot wieder her

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.
3. Wählen Sie den DB-Cluster-Snapshot für die Wiederherstellung aus.
4. Wählen Sie in Actions (Aktionen) die Option Restore Snapshot (Snapshot wiederherstellen) aus.

   Die Seite „Snapshot wiederherstellen“ wird angezeigt.
5. Wählen Sie die DB-Engine-Version, in der Sie den DB-Cluster wiederherstellen möchten.

   Standardmäßig wird der Snapshot in derselben DB-Engine-Version wie der Quell-DB-Cluster wiederhergestellt, sofern sie verfügbar ist.
6. Geben Sie für DB instance identifier (DB-Instance-Kennung) den Namen Ihrer wiederhergestellten DB-Instance ein. Beachten Sie, dass Amazon RDS die DB-Cluster-ID von der von Ihnen angegebenen DB-Instance-ID ableitet.
7. Geben Sie andere Einstellungen an, z. B. die Speicherkonfiguration des DB-Clusters.

Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.

8. Wählen Sie Restore DB Cluster (DB-Cluster wiederherstellen) aus.

AWS CLI

Um einen DB-Cluster aus einem DB-Cluster-Snapshot wiederherzustellen, verwenden Sie den AWS CLI Befehl restore-db-cluster-from-snapshot.

In diesem Beispiel führen Sie eine Wiederherstellung aus einem vorher erstellten DB-Cluster-Snapshot mit dem Namen durc `mydbclustersnapshot`. Sie stellen auf einen neuen DB-Cluster namens wieder he `mynewdbcluster`.

Sie können andere Einstellungen angeben, z. B. die DB-Engine-Version. Wenn Sie keine Engine-Version angeben, wird der DB-Cluster in der Standard-Engine-Version wiederhergestellt.

Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.

Example

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine aurora-mysql|aurora-postgresql
```

Für Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewdbcluster ^
    --snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql|aurora-postgresql
```

Nachdem der DB-Cluster wiederhergestellt wurde, müssen Sie den DB-Cluster in der Sicherheitsgruppe hinzufügen, die von dem DB-Cluster verwendet wurde, um den DB-Snapshot zu erstellen, wenn Sie dieselbe Funktionalität wie in dem vorherigen DB-Cluster erhalten möchten.

> **Important**
>
> Wenn Sie die Konsole für das Wiederherstellen eines DB-Clusters verwenden, erstellt Amazon RDS automatisch die primäre DB-Instance (Writer) für Ihren DB-Cluster. Wenn Sie AWS CLI für das Wiederherstellen eines DB-Clusters verwenden, müssen Sie die primäre Instance für Ihren DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance. Wenn Sie die primäre DB-Instance nicht erstellen, bleiben die DB-Cluster-Endpunkte im Status `creating`.
> Rufen Sie den create-db-instance AWS CLI Befehl auf, um die primäre Instance für Ihren DB-Cluster zu erstellen. Fügen Sie den Namen des DB-Clusters als `--db-cluster-identifier`-Optionswert ein.

RDS-API

Um einen DB-Cluster aus einem DB-Cluster-Snapshot wiederherzustellen, rufen Sie den RDS-API-Vorgang Restore DBCluster FromSnapshot mit den folgenden Parametern auf:

- `DBClusterIdentifier`
- `SnapshotIdentifier`

> **Important**
>
> Wenn Sie die Konsole für das Wiederherstellen eines DB-Clusters verwenden, erstellt Amazon RDS automatisch die primäre DB-Instance (Writer) für Ihren DB-Cluster. Wenn Sie die RDS-API für das Wiederherstellen eines DB-Clusters verwenden, müssen Sie die primäre Instance für Ihren DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance. Wenn Sie die primäre DB-Instance nicht erstellen, bleiben die DB-Cluster-Endpunkte im Status `creating`.
> Rufen Sie den RDS-API-Vorgang Create DBInstance auf, um die primäre Instance für Ihren DB-Cluster zu erstellen. Beziehen Sie den Namen des DB-Clusters als `DBClusterIdentifier`-Parameterwert mit ein.

Kopieren eines DB-Cluster-Snapshots

Mit Amazon Aurora können Sie automatisierte Backups oder manuelle DB-Cluster-Snapshots kopieren. Nach dem Kopieren eines Snapshots ist die Kopie ein manueller Snapshot. Sie können mehrere Kopien eines automatisierten Backups oder eines manuellen Snapshots erstellen, aber jede Kopie muss über eine eindeutige Kennung verfügen.

Sie können einen Snapshot innerhalb desselben kopieren AWS-Region, Sie können einen Snapshot zwischen AWS-Regionen den anderen kopieren und Sie können gemeinsam genutzte Snapshots kopieren. Sie können Snapshots in einem einzigen Schritt auf ein anderes AWS-Region OD-Konto kopieren.

> **Note**
>
> Amazon berechnet Ihnen die Kosten basierend auf dem Aufbewahrungszeitraum und der Menge der Amazon Aurora-Sicherungs- und -Snapshot-Daten. Weitere Informationen über den Speicher, der Aurora-Backups und -Snapshots zugeordnet ist, finden Sie unter Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora. Weitere Informationen zu den Preisen für Aurora-Speicher finden Sie unter Amazon RDS für Aurora: Preise.

Sehen Sie sich die Einschränkungen und Überlegungen zum Kopieren von DB-Cluster-Snapshots genauer an. Informationen zum Kopieren von DB-Cluster-Snapshots finden Sie in einem der folgenden Themen.

- Kopieren eines DB-Cluster-Snapshots mithilfe der AWS-Managementkonsole
- Kopieren eines unverschlüsselten DB-Cluster-Snapshots mit der AWS CLI- oder Amazon-RDS-API
- Kopieren eines verschlüsselten DB-Cluster-Snapshots mit der AWS CLI- oder Amazon-RDS-API
- Kopieren eines DB-Cluster-Snapshots in ein anderes Konto

Kopieren eines DB-Cluster-Snapshots mithilfe der AWS-Managementkonsole

Verwenden Sie zum Kopieren eines DB-Cluster-Snapshots die in diesem Thema beschriebenen Verfahren. Wenn es sich bei der Quell-Datenbank-Engine um Aurora handelt, ist der resultierende Snapshot ein DB-Cluster-Snapshot.

Für jedes AWS-Konto können Sie gleichzeitig bis zu fünf DB-Cluster-Snapshots aus einer AWS-Region in eine andere kopieren. Sie können sowohl verschlüsselte als auch unverschlüsselte DB-Cluster-Snapshots kopieren. Wenn Sie einen DB-Cluster-Snapshot in eine andere AWS-Region kopieren, erstellen Sie einen manuellen DB-Cluster-Snapshot, der in der betreffenden AWS-Region beibehalten wird. Beim Kopieren von DB-Cluster-Snapshots aus der Quell-AWS-Region fallen Amazon-RDS-Datenübertragungsgebühren an.

Weitere Informationen zu den Kosten von Datenübertragungen finden Sie unter Amazon RDS – Preise.

Nachdem die Kopie des DB-Cluster-Snapshots in der neuen AWS-Region erstellt wurde, verhält sich diese DB-Cluster-Snapshot-Kopie wie alle anderen DB-Cluster-Snapshots in der AWS-Region.

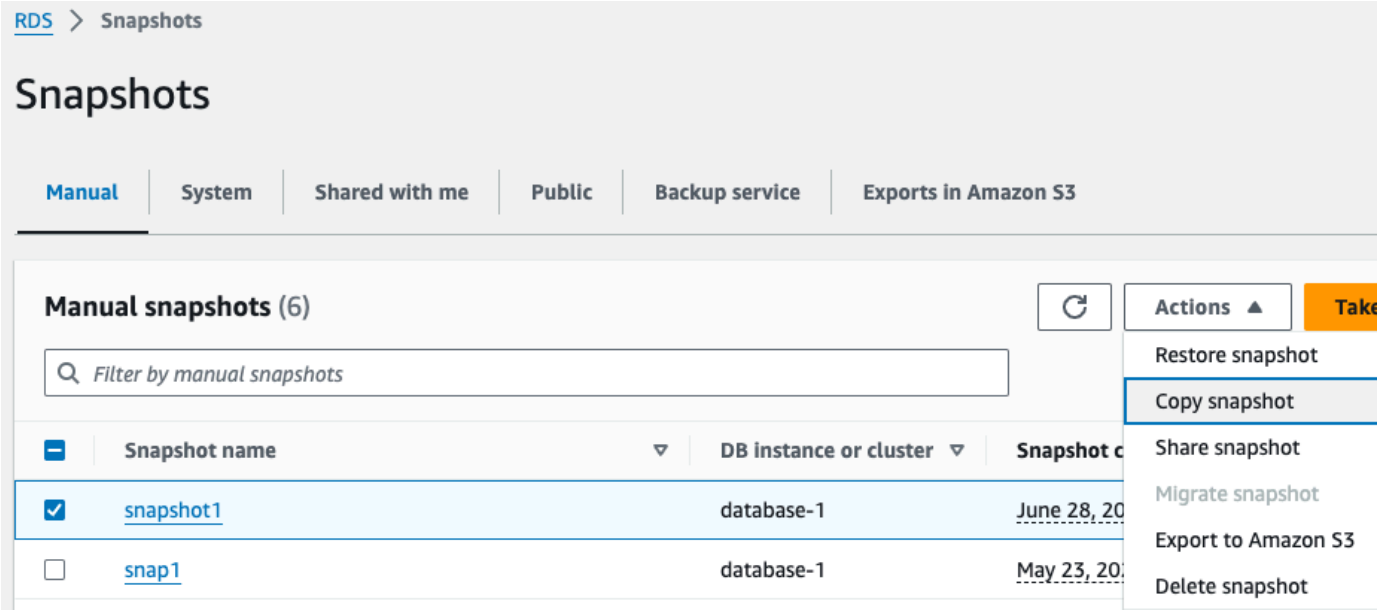
Mit diesem Verfahren kann ein verschlüsselter oder unverschlüsselter DB-Cluster-Snapshot in einer AWS-Region oder regionsübergreifend kopiert werden.

Um einen bereits laufenden Kopiervorgang abzubrechen, können Sie den DB-Cluster-Ziel-Snapshot löschen, während der DB-Cluster-Snapshot noch den Status copying hat.

Bevor Sie einen DB-Cluster-Snapshot kopieren, überprüfen Sie die Einschränkungen und Überlegungen zum Kopieren von Snapshots.

So kopieren Sie einen DB-Cluster-Snapshot

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.
3. Wählen Sie den zu kopierenden DB-Cluster-Snapshot.
4. Wählen Sie die Option Aktionen und anschließend Snapshot kopieren aus.



5. (Optional) Um den DB-Cluster-Snapshot in eine andere AWS-Region zu kopieren, wählen Sie die betreffende AWS-Region für Destination Region (Zielregion) aus.
6. Geben Sie den Namen für die Kopie des DB-Cluster-Snapshots in das Feld New DB Snapshot Identifier (Neue DB-Snapshot-Kennung) ein.
7. Wählen Sie zum Kopieren von Tags und Werten aus dem Snapshot in die Kopie des Snapshots die Option Copy Tags (Tags kopieren).
8. Wählen Sie Copy Snapshot (Snapshot kopieren) aus.

Einschränkungen

Im folgenden werden einige Einschränkungen beim Kopieren von Snapshots aufgeführt:

- Sie können einen Snapshot nicht in oder aus einem der folgenden AWS-Regionen Objekte kopieren:
  - China (Peking)
  - China (Ningxia)
- Sie können einen Snapshot zwischen AWS GovCloud (US-Ost) und AWS GovCloud (US-West) kopieren. Sie können jedoch keinen Snapshot zwischen diesen AWS GovCloud (US) Regionen und kommerziellen Regionen kopieren. AWS-Regionen
- Wenn Sie einen Quell-Snapshot löschen, bevor der Ziel-Snapshot verfügbar ist, kann das Kopieren des Snapshots fehlschlagen. Verifizieren Sie, dass der Ziel-Snapshot den Status `AVAILABLE` hat, bevor Sie einen Quell-Snapshot löschen.

- Pro Konto können bis zu fünf Snapshot-Kopieranforderungen an eine einzelne Zielregion aktiv sein.
- Wenn Sie mehrere Snapshot-Kopien für dieselbe Quell-DB-Instance anfordern, werden diese intern in die Warteschlange gestellt. Mit den später angeforderten Kopien wird erst begonnen, wenn die vorherigen Snapshot-Kopien fertiggestellt sind. Weitere Informationen finden Sie unter Warum ist die Erstellung meines EC2 AMI- oder EBS-Snapshots langsam? im AWS Knowledge Center.
- Je nach Umfang AWS-Regionen und Menge der zu kopierenden Daten kann eine regionsübergreifende Snapshot-Kopie Stunden in Anspruch nehmen. In einigen Fällen kann es eine große Anzahl von regionsübergreifenden Snapshot-Kopieranforderungen aus einer bestimmten -Region geben. In solchen Fällen stellt Amazon RDS neue regionsübergreifende Kopieranforderungen dieser -Quellregion gegebenenfalls in eine Warteschlange, bis aktive Kopiervorgänge abgeschlossen wurden. Zu Kopieranforderungen, die sich in der Warteschlange befinden, werden keine Fortschrittsinformationen angezeigt. Fortschrittsinformationen werden angezeigt, sobald der Kopiervorgang gestartet wird.
- Aurora unterstützt keine inkrementellen Snapshots. Snapshot-Kopien von Aurora-DB-Clustern sind immer vollständige Kopien. Eine vollständige Snapshot-Kopie enthält alle Daten und Metadaten, die zur Wiederherstellung der DB-Cluster erforderlich sind.

Überlegungen zum Kopieren von Snapshots

Nachfolgend sind die Überlegungen aufgeführt, die beim Kopieren von Snapshots angestellt werden sollten.

Themen

- Überlegungen zum Kopieren freigegebener Snapshots
- Überlegungen zum Kopieren verschlüsselter DB-Cluster-Snapshots
- Überlegungen zum regionsübergreifenden Kopieren von Snapshots
- Überlegungen zu Parametergruppen

Überlegungen zum Kopieren freigegebener Snapshots

Sie können Snapshots kopieren, die Ihnen von anderen AWS Konten zur Verfügung gestellt wurden. In einigen Fällen können Sie einen verschlüsselten Snapshot kopieren, der von einem anderen AWS Konto aus geteilt wurde. In diesen Fällen müssen Sie Zugriff auf die Datei haben AWS KMS key , mit der der Snapshot verschlüsselt wurde. Weitere Informationen finden Sie unter Freigeben verschlüsselter Snapshots.

Regions- und kontenübergreifendes Kopieren in einem einzigen Schritt

Um einen Snapshot regionsübergreifend und kontoübergreifend in einer einzigen Aktion zu kopieren, müssen Sie den Snapshot zunächst für das Zielkonto freigeben. AWS Wenn der Snapshot verschlüsselt ist, müssen Sie den AWS KMS Schlüssel auch mit dem AWS Zielkonto teilen. Wenn der Snapshot mit dem AWS KMS Standardschlüssel verschlüsselt ist, müssen Sie den Snapshot zuerst kopieren, um ihn erneut mit einem vom Kunden verwalteten Schlüssel zu verschlüsseln, bevor Sie ihn mit dem Zielkonto teilen. Nach der Freigabe können Sie vom Zielkonto aus eine Kopie auf dieses Konto (regionsübergreifend oder regionsübergreifend) initiieren.

Überlegungen zum Kopieren verschlüsselter DB-Cluster-Snapshots

Sie können einen Snapshot kopieren, der mit einem KMS-Schlüssel verschlüsselt wurde. Wenn Sie einen verschlüsselten Snapshot kopieren, muss auch die Kopie des Snapshots verschlüsselt werden. Wenn Sie einen verschlüsselten Snapshot innerhalb desselben kopieren AWS-Region, können Sie die Kopie mit demselben KMS-Schlüssel wie den ursprünglichen Snapshot verschlüsseln. Oder Sie können einen anderen KMS-Schlüssel angeben.

Wenn Sie einen verschlüsselten Snapshot in andere Regionen kopieren, müssen Sie einen gültigen KMS-Schlüssel für die Ziel- AWS-Region angeben. Es kann ein regionsspezifischer KMS-Schlüssel oder ein multiregionaler Schlüssel sein. Weitere Informationen über multiregionale Schlüssel finden Sie unter Verwenden von multiregionalen Schlüsseln in AWS KMS.

Weitere Informationen zur AWS KMS Schlüsselverwaltung für Amazon RDS finden Sie unterAWS KMS key-Verwaltung.

Der Quell-Snapshot bleibt den gesamten Kopiervorgang über verschlüsselt. Weitere Informationen finden Sie unter Einschränkungen von Amazon Aurora-verschlüsselten DB-Clustern.

> **Note**
>
> Bei Snapshots von Amazon Aurora-DB-Clustern können Sie einen unverschlüsselten DB-Cluster-Snapshot nicht verschlüsseln, wenn Sie den Snapshot kopieren.

Informationen zum Kopieren verschlüsselter DB-Cluster-Snapshots finden Sie in den folgenden Themen.

- Kopieren eines verschlüsselten DB-Cluster-Snapshots mit der AWS CLI- oder Amazon-RDS-API

- Kopieren eines DB-Cluster-Snapshots in ein anderes Konto

Überlegungen zum regionsübergreifenden Kopieren von Snapshots

Sie können DB-Cluster-Snapshots über AWS-Regionen hinweg kopieren. Es gibt jedoch bestimmte Einschränkungen und Überlegungen für das Kopieren von regionsübergreifenden Snapshots.

Je nach Umfang und AWS-Regionen Menge der zu kopierenden Daten kann es Stunden dauern, bis eine regionsübergreifende Snapshot-Kopie abgeschlossen ist.

In einigen Fällen kann es eine große Anzahl von regionsübergreifenden Snapshot-Kopieranforderungen aus einer bestimmten AWS-Region geben. In solchen Fällen kann Amazon RDS neue regionsübergreifende Kopieranfragen von dieser Quelle AWS-Region in eine Warteschlange stellen, bis einige in Bearbeitung befindliche Kopien abgeschlossen sind. Zu Kopieranforderungen, die sich in der Warteschlange befinden, werden keine Fortschrittsinformationen angezeigt. Fortschrittsinformationen werden angezeigt, sobald das Kopieren beginnt.

Für regionsübergreifende Snapshot-Kopien fallen Gebühren für die Datenübertragung an. Beim regionsübergreifenden Kopieren von Snapshots werden vollständige Kopien in den Zieldaten erstellt, die Datenübertragungsgebühren werden jedoch inkrementell berechnet. Inkrementelle Daten umfassen sowohl die neuen Daten, die der Datenbank eines Kunden seit der letzten Kopie hinzugefügt wurden, als auch alle Änderungen, die an vorhandenen Daten vorgenommen wurden. Weitere Informationen finden Sie unter Erstellen von Backup-Kopien über mehrere AWS-Regionen hinweg im AWS Backup Entwicklerhandbuch.

> **Note**
>
> Aurora kopiert die Mindestmenge an Daten, die erforderlich ist, um eine vollständige Kopie eines Snapshots in der Zielregion zu erstellen. Beim regionsübergreifenden Kopieren von Snapshots fallen Gebühren für die Datenübertragung an.

Überlegungen zu Parametergruppen

Wenn Sie einen Snapshot regionsübergreifend kopieren, enthält die Kopie nicht die vom ursprünglichen DB-Cluster verwendete Parametergruppe. Wenn Sie einen Snapshot wiederherstellen, um einen neuen DB-Cluster zu erstellen, erhält dieser DB-Cluster die Standardparametergruppe für den, in dem AWS-Region er erstellt wurde. Soll der neue DB-Cluster die gleichen Parameter wie das Original erhalten, gehen Sie folgendermaßen vor:

1. Erstellen Sie im Ziel AWS-Region eine DB-Cluster-Parametergruppe mit denselben Einstellungen wie der ursprüngliche DB-Cluster. Wenn im neuen System bereits eine vorhanden ist AWS-Region, können Sie diese verwenden.
2. Nachdem Sie den Snapshot im Ziel wiederhergestellt haben AWS-Region, ändern Sie den neuen DB-Cluster und fügen Sie die neue oder vorhandene Parametergruppe aus dem vorherigen Schritt hinzu.

Kopieren eines unverschlüsselten DB-Cluster-Snapshots mit der AWS CLI- oder Amazon-RDS-API

Verwenden Sie die in den folgenden Abschnitten beschriebenen Verfahren, um einen unverschlüsselten DB-Cluster-Snapshot mit der AWS-Managementkonsole-, AWS CLI- oder der Amazon-RDS-API zu kopieren.

Um einen bereits laufenden Kopiervorgang abzubrechen, können Sie den DB-Cluster-Ziel-Snapshot (`--target-db-cluster-snapshot-identifier` oder `TargetDBClusterSnapshotIdentifier`) löschen, während der DB-Cluster-Snapshot noch den Status copying hat.

Konsole

Weitere Informationen zum Kopieren eines DB-Cluster-Snapshots mithilfe der AWS-Managementkonsole finden Sie unter Kopieren eines DB-Cluster-Snapshots mithilfe der AWS-Managementkonsole.

AWS CLI

Verwenden Sie zum Kopieren eines DB-Cluster-Snapshots den AWS CLI-Befehl copy-db-cluster-snapshot. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, führen Sie den Befehl in der AWS-Region aus, in die der Snapshot kopiert werden soll.

Die folgenden Optionen werden zum Kopieren eines unverschlüsselten DB-Cluster-Snapshots verwendet:

- `--source-db-cluster-snapshot-identifier` – Der Bezeichner des zu kopierenden DB-Cluster-Snapshots. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, muss dieser Bezeichner im ARN-Format der Quell-AWS-Region angegeben werden.
- `--target-db-cluster-snapshot-identifier` – Der Bezeichner für die neue Kopie des DB-Cluster-Snapshots.

Der folgende Code erstellt eine Kopie des DB-Cluster-Snapshots `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` mit dem Namen `myclustersnapshotcopy` in der AWS-Region, in der der Befehl ausgeführt wird. Beim Erstellen der Kopie werden alle Tags des ursprünglichen Snapshots in die Snapshot-Kopie übernommen.

Example

Für Linux, macOS oder Unix:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --copy-tags
```

Für Windows:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --copy-tags
```

RDS-API

Zum Kopieren eines DB-Cluster-Snapshots verwenden Sie die Amazon-RDS-API-Operation CopyDBClusterSnapshot. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, führen Sie die Aktion in der AWS-Region aus, in die der Snapshot kopiert wird.

Die folgenden Parameter werden zum Kopieren eines unverschlüsselten DB-Cluster-Snapshots verwendet:

- `SourceDBClusterSnapshotIdentifier` – Der Bezeichner des zu kopierenden DB-Cluster-Snapshots. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, muss dieser Bezeichner im ARN-Format der Quell-AWS-Region angegeben werden.
- `TargetDBClusterSnapshotIdentifier` – Der Bezeichner für die neue Kopie des DB-Cluster-Snapshots.

Der folgende Code erstellt eine Kopie eines Snapshots `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` namens `myclustersnapshotcopy` in der USA West (Nordkalifornien)-Region . Beim Erstellen der Kopie werden alle Tags des ursprünglichen Snapshots in die Snapshot-Kopie übernommen.

Example

```
https://rds.us-west-1.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20130805
   &TargetDBSnapshotIdentifier=myclustersnapshotcopy
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
   &X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Kopieren eines verschlüsselten DB-Cluster-Snapshots mit der AWS CLI- oder Amazon-RDS-API

Verwenden Sie die in den folgenden Abschnitten beschriebenen Verfahren, um einen verschlüsselten DB-Cluster-Snapshot mit der AWS-Managementkonsole-, AWS CLI- oder der Amazon-RDS-API zu kopieren.

Um einen bereits laufenden Kopiervorgang abzubrechen, können Sie den DB-Cluster-Ziel-Snapshot (`--target-db-cluster-snapshot-identifier` oder `TargetDBClusterSnapshotIdentifier`) löschen, während der DB-Cluster-Snapshot noch den Status copying hat.

Konsole

Weitere Informationen zum Kopieren eines DB-Cluster-Snapshots mithilfe der AWS-Managementkonsole finden Sie unter Kopieren eines DB-Cluster-Snapshots mithilfe der AWS-Managementkonsole.

AWS CLI

Verwenden Sie zum Kopieren eines DB-Cluster-Snapshots den AWS CLI-Befehl copy-db-cluster-snapshot. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, führen Sie den Befehl in der AWS-Region aus, in die der Snapshot kopiert werden soll.

Die folgenden Optionen werden zum Kopieren eines verschlüsselten DB-Cluster-Snapshots verwendet:

- `--source-db-cluster-snapshot-identifier` – Der Bezeichner des zu kopierenden verschlüsselten DB-Cluster-Snapshots. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, muss dieser Bezeichner im ARN-Format der Quell-AWS-Region angegeben werden.
- `--target-db-cluster-snapshot-identifier` – Der Bezeichner für die neue Kopie des verschlüsselten DB-Cluster-Snapshots.
- `--kms-key-id` - Die KMS-Schlüsselkennung für den Schlüssel, der für die Verschlüsselung der Kopie des DB-Cluster-Snapshots verwendet werden soll.

  Sie können diese Option optional verwenden, wenn der DB-Cluster-Snapshot verschlüsselt ist, Sie den Snapshot in dieselbe AWS-Region kopieren und Sie einen neuen KMS-Schlüssel zur Verschlüsselung der Kopie angeben möchten. Andernfalls wird die Kopie des DB-Cluster-Snapshots mit demselben KMS-Schlüssel wie der ursprüngliche DB-Cluster-Snapshot verschlüsselt.

  Sie müssen diese Option angeben, wenn der DB-Cluster-Snapshot verschlüsselt ist und Sie den Snapshot in eine andere AWS-Region kopieren. In diesem Fall müssen Sie einen KMS-Schlüssel für die Ziel-AWS-Region angeben.

Das folgende Codebeispiel kopiert den verschlüsselten DB-Cluster-Snapshot aus der Region USA West (Oregon) in die Region US East (N. Virginia). Der Befehl wird in der US East (N. Virginia)-Region aufgerufen.

Example

Für Linux, macOS oder Unix:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
```

```
  --kms-key-id my-us-east-1-key
```

Für Windows:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --kms-key-id my-us-east-1-key
```

Der Parameter `--source-region` ist erforderlich, wenn Sie einen verschlüsselten DB-Cluster-Snapshot zwischen den Regionen AWS GovCloud (USA-Ost) und AWS GovCloud (USA-West) kopieren. Geben Sie für `--source-region` die AWS-Region der Quell-DB-Instance an. Die in `source-db-cluster-snapshot-identifier` angegebene AWS-Region muss mit der AWS-Region übereinstimmen, die in `--source-region` festgelegt ist.

Wenn `--source-region` nicht festgelegt ist, geben Sie einen `--pre-signed-url`-Wert an. Eine vorsignierte URL ist eine URL, die eine mit der Signaturversion 4 signierte Anforderung für den Befehl `copy-db-cluster-snapshot` enthält, die in der Quell-AWS-Region aufgerufen wird. Weitere Informationen über die Option `pre-signed-url` finden Sie unter copy-db-cluster-snapshot in der AWS CLI-Befehlsreferenz.

RDS-API

Zum Kopieren eines DB-Cluster-Snapshots verwenden Sie die Amazon-RDS-API-Operation CopyDBClusterSnapshot. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, führen Sie die Aktion in der AWS-Region aus, in die der Snapshot kopiert wird.

Die folgenden Parameter werden zum Kopieren eines verschlüsselten DB-Cluster-Snapshots verwendet:

- `SourceDBClusterSnapshotIdentifier` – Der Bezeichner des zu kopierenden verschlüsselten DB-Cluster-Snapshots. Wenn Sie den Snapshot in eine andere AWS-Region kopieren, muss dieser Bezeichner im ARN-Format der Quell-AWS-Region angegeben werden.
- `TargetDBClusterSnapshotIdentifier` – Der Bezeichner für die neue Kopie des verschlüsselten DB-Cluster-Snapshots.
- `KmsKeyId` - Die KMS-Schlüsselkennung für den Schlüssel, der für die Verschlüsselung der Kopie des DB-Cluster-Snapshots verwendet werden soll.

Sie können diesen Parameter optional verwenden, wenn der DB-Cluster-Snapshot verschlüsselt ist, Sie den Snapshot in dieselbe AWS-Region kopieren und Sie einen neuen KMS-Schlüssel angeben, der zur Verschlüsselung der Kopie verwendet wird. Andernfalls wird die Kopie des DB-Cluster-Snapshots mit demselben KMS-Schlüssel wie der ursprüngliche DB-Cluster-Snapshot verschlüsselt.

Sie müssen diesen Parameter angeben, wenn der DB-Cluster-Snapshot verschlüsselt ist und Sie den Snapshot in eine andere AWS-Region kopieren. In diesem Fall müssen Sie einen KMS-Schlüssel für die Ziel-AWS-Region angeben.

- `PreSignedUrl` – Wenn Sie den Snapshot in eine andere AWS-Region kopieren, müssen Sie den Parameter `PreSignedUrl` angeben. Der Wert `PreSignedUrl` muss eine URL sein, die eine mit Signature Version 4 signierte Anforderung für die Aktion `CopyDBClusterSnapshot` enthält, die in der Quell-AWS-Region aufgerufen werden muss, aus der der DB-Cluster-Snapshot kopiert wird. Weitere Informationen zur Verwendung einer vorsignierten URL erhalten Sie unter CopyDBClusterSnapshot.

Das folgende Codebeispiel kopiert den verschlüsselten DB-Cluster-Snapshot aus der Region USA West (Oregon) in die Region US East (N. Virginia). Die Aktion wird in der US East (N. Virginia)-Region aufgerufen.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBClusterSnapshot
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBClusterSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards
%25253Aus-west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-
snapshot-20161115
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
```

```
        %2526X-Amz-Expires%253D3600
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
    &TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

Der Parameter `PreSignedUrl` ist erforderlich, wenn Sie einen verschlüsselten DB-Cluster-Snapshot zwischen den Regionen AWS GovCloud (USA-Ost) und AWS GovCloud (USA-West) kopieren. Der Wert `PreSignedUrl` muss eine URL sein, die eine mit Signature Version 4 signierte Anforderung für die Operation `CopyDBClusterSnapshot` enthält, die in der Quell-AWS-Region aufgerufen werden muss, aus der der DB-Cluster-Snapshot kopiert wird. Weitere Informationen zur Verwendung einer vorsignierten URL erhalten Sie unter CopyDBClusterSnapshot in der Amazon-RDS-API-Referenz.

Verwenden Sie stattdessen den AWS CLI-Befehl copy-db-cluster-snapshot mit der Option `--source-region`, um eine vorsignierte URL automatisch (und nicht manuell) zu generieren.

Kopieren eines DB-Cluster-Snapshots in ein anderes Konto

Sie können es anderen AWS-Konten ermöglichen, DB-Cluster-Snapshots zu kopieren, die Sie mit den Amazon-RDS-API-Aktionen `ModifyDBClusterSnapshotAttribute` und `CopyDBClusterSnapshot` angeben. DB-Cluster-Snapshots können nur in der ursprünglichen AWS-Region in ein anderes Konto kopiert werden. Für das kontoübergreifende Kopieren macht Konto A den Snapshot zum Kopieren verfügbar und Konto B kopiert den Snapshot.

1. Rufen Sie unter Verwendung von Konto A `ModifyDBClusterSnapshotAttribute` auf und geben Sie **`restore`** für den Parameter `AttributeName` und die ID für Konto B für den Parameter `ValuesToAdd` an.

2. (Wenn der Snapshot verschlüsselt ist) Aktualisieren Sie unter Verwendung von Konto A die Schlüsselrichtlinie für den KMS-Schlüssel, fügen Sie den ARN von Konto B zuerst als `Principal` hinzu und erlauben Sie dann die Aktion `kms:CreateGrant`
3. (Wenn der Snapshot verschlüsselt ist) Verwenden Sie Konto B und wählen oder erstellen Sie einen Benutzer. Hängen Sie eine IAM-Richtlinie an den Benutzer an, die ihm das Kopieren eines verschlüsselten DB-Cluster-Snapshots unter Verwendung Ihres KMS-Schlüssels erlaubt.
4. Rufen Sie unter Verwendung von Konto B `CopyDBClusterSnapshot` auf und verwenden Sie den Parameter `SourceDBClusterSnapshotIdentifier`, um den ARN des zu kopierenden DB-Cluster-Snapshots anzugeben, der die ID für Konto A enthalten muss.

Sie können alle AWS-Konten auflisten, die einen DB-Cluster-Snapshot wiederherstellen dürfen, indem Sie die API-Operationen DescribeDBSnapshotAttributes oder DescribeDBClusterSnapshotAttributes verwenden.

Um die Freigabeberechtigung eines AWS-Kontos zu widerrufen, verwenden Sie die Aktion `ModifyDBSnapshotAttribute` oder `ModifyDBClusterSnapshotAttribute`, während `AttributeName` auf `restore` gesetzt ist und der Parameter `ValuesToRemove` die ID des betreffenden Kontos enthält.

Kopieren eines unverschlüsselten DB-Cluster-Snapshots in ein anderes Konto

Verwenden Sie das folgende Verfahren, um einen unverschlüsselten DB-Cluster-Snapshot in ein anderes Konto in derselben AWS-Region zu kopieren.

1. Rufen Sie im Quellkonto für den DB-Cluster-Snapshot `ModifyDBClusterSnapshotAttribute` auf und geben Sie **`restore`** für den Parameter `AttributeName` und die ID für das Zielkonto für den Parameter `ValuesToAdd` an.

   Das Ausführen des folgenden Beispiels mit dem Konto `987654321` gewährt den beiden mit den Bezeichnern AWS und `123451234512` angegebenen `123456789012`-Konten das Recht zum Wiederherstellen des DB-Cluster-Snapshots mit dem Namen `manual-snapshot1`.

   ```
   https://rds.us-west-2.amazonaws.com/
    ?Action=ModifyDBClusterSnapshotAttribute
    &AttributeName=restore
    &DBClusterSnapshotIdentifier=manual-snapshot1
    &SignatureMethod=HmacSHA256&SignatureVersion=4
    &ValuesToAdd.member.1=123451234512
    &ValuesToAdd.member.2=123456789012
   ```

```
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. Rufen Sie im Zielkonto `CopyDBClusterSnapshot` auf und verwenden Sie den Parameter `SourceDBClusterSnapshotIdentifier`, um den ARN des zu kopierenden DB-Cluster-Snapshots anzugeben, der die ID für das Quellkonto enthalten muss.

   Das Ausführen des folgenden Beispiels mit dem Konto `123451234512` kopiert den DB-Cluster-Snapshot `aurora-cluster1-snapshot-20130805` aus dem Konto `987654321` und erstellt einen DB-Cluster-Snapshot namens `dbclustersnapshot1`.

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Kopieren eines verschlüsselten DB-Cluster-Snapshots in ein anderes Konto

Verwenden Sie das folgende Verfahren, um einen verschlüsselten DB-Cluster-Snapshot in ein anderes Konto in derselben AWS-Region zu kopieren.

1. Rufen Sie im Quellkonto für den DB-Cluster-Snapshot `ModifyDBClusterSnapshotAttribute` auf und geben Sie **`restore`** für den Parameter `AttributeName` und die ID für das Zielkonto für den Parameter `ValuesToAdd` an.

Das Ausführen des folgenden Beispiels mit dem Konto `987654321` gewährt den beiden mit den Bezeichnern AWS und `123451234512` angegebenen `123456789012`-Konten das Recht zum Wiederherstellen des DB-Cluster-Snapshots mit dem Namen `manual-snapshot1`.

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. Erstellen Sie im Quellkonto für den DB-Cluster-Snapshot einen benutzerdefinierten KMS-Schlüssel in derselben AWS-Region wie im verschlüsselten DB-Cluster-Snapshot. Bei der Erstellung des vom Kunden verwalteten Schlüssels gewähren Sie für das Ziel-AWS-Konto Zugriff darauf. Weitere Informationen finden Sie unter Erstellen Sie einen vom Kunden verwalteten Schlüssel und gewähren Sie Zugriff darauf..
3. Kopieren Sie den Snapshot und geben Sie ihn für das Ziel AWS-Konto frei. Weitere Informationen finden Sie unter Kopieren und Freigeben des Snapshots über das Quellkonto.
4. Rufen Sie im Zielkonto `CopyDBClusterSnapshot` auf und verwenden Sie den Parameter `SourceDBClusterSnapshotIdentifier`, um den ARN des zu kopierenden DB-Cluster-Snapshots anzugeben, der die ID für das Quellkonto enthalten muss.

   Das Ausführen des folgenden Beispiels mit dem Konto `123451234512` kopiert den DB-Cluster-Snapshot `aurora-cluster1-snapshot-20130805` aus dem Konto `987654321` und erstellt einen DB-Cluster-Snapshot namens `dbclustersnapshot1`.

   ```
   https://rds.us-west-2.amazonaws.com/
      ?Action=CopyDBClusterSnapshot
      &CopyTags=true
      &SignatureMethod=HmacSHA256
      &SignatureVersion=4
   ```

```
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Freigeben eines DB-Cluster-Snapshots

Mithilfe von Amazon RDS können Sie einen manuellen DB-Cluster-Snapshot wie folgt freigeben:

- Die Freigabe eines manuellen DB-Cluster-Snapshots, ob verschlüsselt oder nicht verschlüsselt, ermöglicht autorisierten AWS-Konten, den Snapshot zu kopieren.
- Die Freigabe eines manuellen DB-Cluster-Snapshots, ob verschlüsselt oder nicht verschlüsselt, ermöglicht autorisierten AWS-Konten die direkte Wiederherstellung eines DB-Clusters aus dem Snapshot, statt eine Kopie des DB-Clusters zu erstellen und es aus dieser Kopie wiederherzustellen.

> **Note**
>
> Um einen automatisierten DB-Cluster-Snapshot freizugeben, erstellen Sie einen manuellen DB-Cluster-Snapshot, indem Sie den automatisierten Snapshot kopieren und diese Kopie dann freigeben. Dieser Vorgang gilt auch für Ressourcen, die von AWS Backup generiert wurden.

Weitere Informationen zum Kopieren von Snapshots finden Sie unter Kopieren eines DB-Cluster-Snapshots. Weitere Informationen zum Wiederherstellen einer DB-Instance aus einem DB-Cluster-Snapshot finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.

Weitere Informationen zum Wiederherstellen eines DB-Clusters aus einem DB-Cluster-Snapshot finden Sie unter Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters.

Sie können einen manuellen Snapshot für bis zu 20 weitere AWS-Konten freigeben.

Bei der Freigabe manueller Snapshots für andere AWS-Konten gilt die folgende Einschränkung:

- Wenn Sie einen DB-Cluster über die AWS Command Line Interface (AWS CLI) oder Amazon-RDS-API aus einem freigegebenen Snapshot wiederherstellen, müssen Sie den Amazon-Ressourcennamen (ARN) des freigegebenen Snapshots als Snapshot-Bezeichner angeben.

In den folgenden Abschnitten erfahren Sie, wie Sie Snapshots, öffentliche Snapshots und verschlüsselte Snapshots freigeben. Außerdem lernen Sie, wie Sie das Freigeben von Snapshots beenden.

Themen

- Freigeben eines Snapshots
- Freigeben öffentlicher Snapshots
- Freigeben verschlüsselter Snapshots
- Beenden der Snapshot-Freigabe

Freigeben eines Snapshots

Mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API geben Sie einen manuellen DB-Cluster-Snapshot frei.

Konsole

Mithilfe der Amazon-RDS-Konsole geben Sie einen manuellen DB-Cluster-Snapshot für bis zu 20 AWS-Konten frei. Sie können die Konsole auch verwenden, um die Freigabe eines manuellen Snapshots für ein oder mehrere Konten zu beenden.

Freigeben eines manuellen DB-Cluster-Snapshots über die Amazon-RDS-Konsole

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.
3. Wählen Sie den manuellen Snapshot, den Sie freigeben möchten.
4. Wählen Sie unter Actions (Aktionen) die Option Share Snapshots (Snapshot freigeben).
5. Wählen Sie für DB Snapshot Visibility (Sichtbarkeit des DB-Snapshots) eine der folgenden Optionen.

    - Wenn die Quelle unverschlüsselt ist, wählen Sie Öffentlich aus, um allen AWS-Konten zu erlauben, einen DB-Cluster aus Ihrem manuellen DB-Cluster-Snapshot wiederherzustellen. Alternativ können Sie Privat auswählen, um nur AWS-Konten, die Sie angeben, zu erlauben, einen DB-Cluster aus Ihrem manuellen DB-Cluster-Snapshot wiederherzustellen.

      > ⚠ Warning
      >
      > Wenn Sie die Sichtbarkeit des DB-Snapshots auf Öffentlich festlegen, können alle AWS-Konten einen DB-Cluster aus Ihrem manuellen DB-Cluster-Snapshot

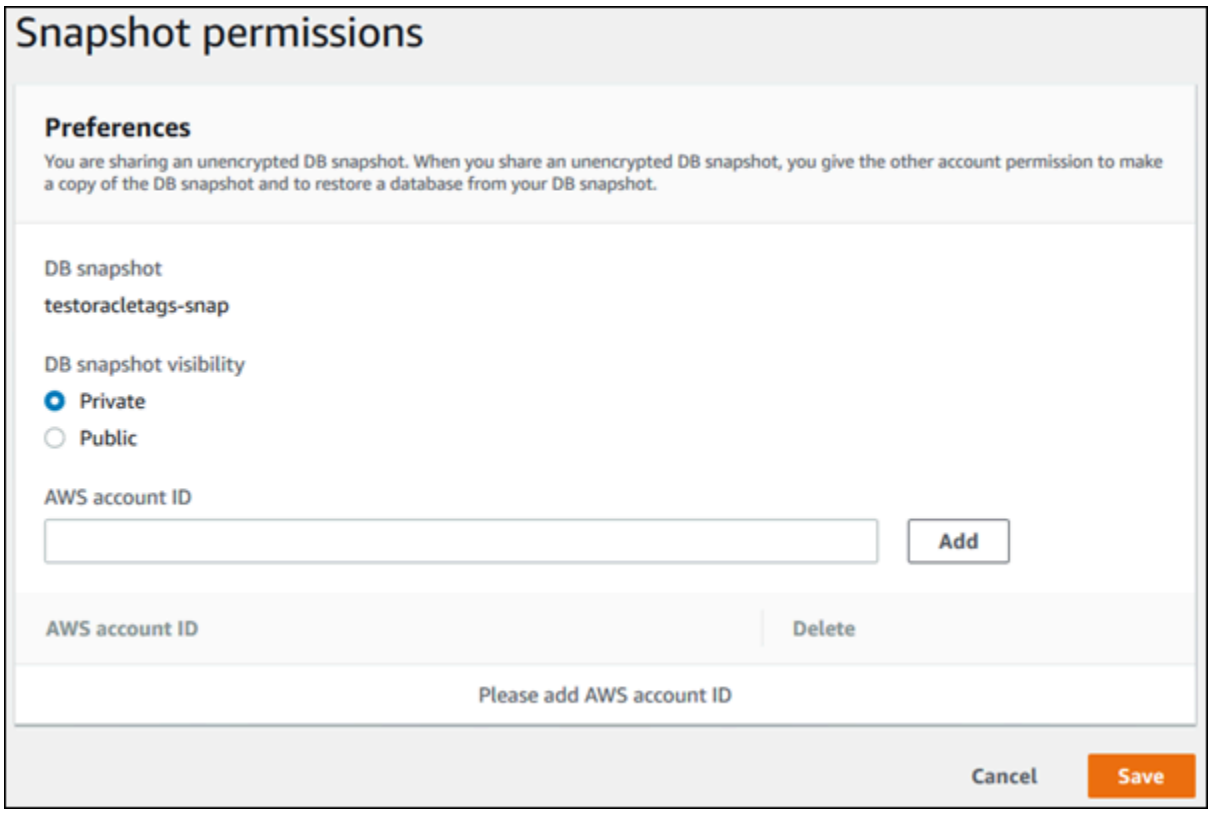
wiederherstellen und haben Zugriff auf Ihre Daten. Geben Sie keine manuellen DB-Cluster-Snapshots als Public (Öffentlich) frei, die private Informationen enthalten. Weitere Informationen finden Sie unter Freigeben öffentlicher Snapshots.

- Ist die Quelle verschlüsselt, ist DB Snapshot Visibility (Sichtbarkeit des DB-Snapshots) auf Private (Privat) festgelegt, da verschlüsselte Snapshots nicht als öffentlich freigegeben werden können.

> **Note**
>
> Mit dem standardmäßigen AWS KMS key verschlüsselte Snapshots können nicht freigegeben werden. Informationen zur Umgehung dieses Problems finden Sie unter Freigeben verschlüsselter Snapshots.

6. Geben Sie in AWS-Konto-ID den AWS-Konto-Bezeichner für ein Konto ein, für das Sie die Wiederherstellung eines DB-Clusters aus Ihrem manuellen Snapshot zulassen möchten, und wählen Sie anschließend Hinzufügen aus. Wiederholen Sie diesen Vorgang, um zusätzliche AWS-Konto-Bezeichner einzuschließen (bis zu 20 AWS-Konten).

   Wenn Sie beim Hinzufügen eines AWS-Konto-Bezeichners zur Liste der zulässigen Konten einen Fehler machen, können Sie ihn aus der Liste löschen, indem Sie rechts neben dem falschen AWS-Konto-Bezeichner Löschen wählen.



7. Nachdem Sie für alle AWS-Konten, für die Sie die Wiederherstellung aus dem manuellen Snapshot zulassen möchten, Bezeichner hinzugefügt haben, wählen Sie Speichern, um Ihre Änderungen zu speichern.

AWS CLI

Verwenden Sie den Befehl `aws rds modify-db-cluster-snapshot-attribute`, um einen DB-Cluster-Snapshot freizugeben. Nutzen Sie den Parameter `--values-to-add`, um eine Liste der IDs für die AWS-Konten hinzuzufügen, die zur Wiederherstellung des manuellen Snapshots autorisiert sind.

Example einen Snapshot mit einem einzigen Konto teilen

Im folgenden Beispiel wird für den AWS-Konto-Bezeichner `123456789012` erlaubt, den DB-Cluster-Snapshot namens `cluster-3-snapshot` wiederherzustellen.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
```

```
--values-to-add 123456789012
```

Für Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example einen Snapshot mit mehreren Konten teilen

Im folgenden Beispiel wird für zwei AWS-Konto-Bezeichner, `111122223333` und `444455556666`, erlaubt, den DB-Cluster-Snapshot namens `manual-cluster-snapshot1` wiederherzustellen.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

Für Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> Bei Verwendung der Windows-Befehlszeile müssen doppelte Anführungszeichen (") im JSON-Code mit einem umgekehrten Schrägstrich (\) als Escape-Zeichen versehen werden.

Um die AWS-Konten aufzulisten, für die das Wiederherstellen eines Snapshots erlaubt wurde, verwenden Sie den AWS CLI-Befehl `describe-db-cluster-snapshot-attributes`.

RDS-API

Sie können einen manuellen DB-Cluster-Snapshot auch über die Amazon-RDS-API für andere AWS-Konten freigeben. Rufen Sie dazu die Operation `ModifyDBClusterSnapshotAttribute` auf.

Geben Sie `restore` für `AttributeName` an und verwenden Sie den Parameter `ValuesToAdd`, um eine Liste der IDs für die AWS-Konten hinzuzufügen, die zur Wiederherstellung des manuellen Snapshots autorisiert sind.

Um einen manuellen Snapshot öffentlich zu machen, sodass alle AWS-Konten ihn wiederherstellen können, verwenden Sie den Wert `all`. Achten Sie jedoch darauf, den Wert `all` keinen manuellen Snapshots hinzuzufügen, die private Informationen enthalten, die nicht für alle AWS-Konten verfügbar sein sollen. Darüber hinaus sollten Sie `all` nicht für verschlüsselte Snapshots angeben, da die öffentliche Bereitstellung dieser Snapshots nicht unterstützt wird.

Um alle AWS-Konten aufzulisten, für die die Wiederherstellung eines Snapshots erlaubt ist, verwenden Sie die API-Operation `DescribeDBClusterSnapshotAttributes`.

Freigeben öffentlicher Snapshots

Sie können einen nicht verschlüsselten manuellen Snapshot als öffentlich freigeben. Damit ist der Snapshot für alle AWS-Konten verfügbar. Achten Sie bei der Freigabe eines Snapshots als öffentlich darauf, dass in Ihren öffentlichen Snapshots keine privaten Informationen enthalten sind.

Wenn ein Snapshot öffentlich freigegeben wird, erhalten alle AWS-Konten die Berechtigung sowohl zum Kopieren des Snapshots als auch zum Erstellen von DB-Clustern aus dem Snapshot.

Die Speicherung von Backups öffentlicher Snapshots anderer Konten wird Ihnen nicht in Rechnung gestellt. Ihnen werden nur Ihre eigenen Snapshots berechnet.

Wenn Sie einen öffentlichen Snapshot kopieren, sind Sie der Eigentümer der Kopie. Ihnen wird die Speicherung von Backups Ihrer Snapshot-Kopie in Rechnung gestellt. Wenn Sie einen DB-Cluster aus einem öffentlichen Snapshot erstellen, wird Ihnen dieser DB-Cluster in Rechnung gestellt. Informationen zur Preisgestaltung von Amazon Aurora finden Sie in der Aurora-Preisliste.

Sie können nur Ihre eigenen öffentlichen Snapshots löschen. Um einen freigegebenen oder öffentlichen Snapshot zu löschen, müssen Sie sich bei dem AWS-Konto anmelden, in dessen Besitz sich der Snapshot befindet.

Aufrufen öffentlicher Snapshots anderer AWS-Konten

Sie können öffentliche Snapshots aufrufen, die anderen Konten in einer bestimmten AWS-Region gehören. Rufen Sie dazu die Registerkarte Öffentlich der Seite Snapshots in der Amazon-RDS-Konsole auf. Ihre Snapshots (die Ihrem Konto gehören) werden auf dieser Registerkarte nicht angezeigt.

So rufen Sie öffentliche Snapshots auf

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.
3. Wählen Sie die Registerkarte Public (Öffentlich).

   Die öffentlichen Snapshots werden angezeigt. Sie können in der Spalte Owner (Eigentümer) sehen, welches Konto einen öffentlichen Snapshot besitzt.

   > **Note**
   >
   > Möglicherweise müssen Sie die Seiteneinstellungen ändern, indem Sie das Zahnradsymbol oben rechts im Bereich Public Snapshots (Öffentliche Snapshots aufrufen, damit diese Spalte eingeblendet wird.

Aufrufen Ihrer eigenen öffentlichen Snapshots

Sie können den folgenden AWS CLI-Befehl (nur Unix) verwenden, um die öffentlichen Snapshots im Besitz von Ihrem AWS-Konto in einer bestimmten AWS-Region aufzurufen.

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |
 grep account_number
```

Die zurückgegebene Ausgabe ähnelt bei öffentlichen Snapshots dem folgenden Beispiel.

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot1",
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot2",
```

Freigeben von öffentlichen Snapshots aus veralteten DB-Engine-Versionen

Das Wiederherstellen oder Kopieren von öffentlichen Snapshots aus veralteten DB-Engine-Versionen wird nicht unterstützt. Gehen Sie wie folgt vor, um Ihren vorhandenen öffentlichen Snapshot, der nicht unterstützt wird, zum Wiederherstellen oder Kopieren verfügbar zu machen:

1. Markieren Sie den Snapshot als privat.
2. Stellen Sie den Snapshot wieder her.

3. Führen Sie ein Upgrade des wiederhergestellten DB-Clusters auf eine unterstützte Engine-Version durch.
4. Erstellen Sie einen neuen Snapshot.
5. Geben Sie den Snapshot erneut öffentlich frei.

Freigeben verschlüsselter Snapshots

Sie können DB-Cluster-Snapshots freigeben, die während der Speicherung mittels des AES-256-Verschlüsselungsalgorithmus verschlüsselt wurden, wie in Verschlüsseln von Amazon Aurora-Ressourcen beschrieben.

Die folgenden Einschränkungen gelten für die Freigabe verschlüsselter Snapshots:

- Sie können verschlüsselte Snapshots nicht als öffentlich freigeben.
- Sie können keine Snapshots freigeben, die mit dem Standard-KMS-Schlüssel des AWS-Konto-Kontos verschlüsselt wurden, das den Snapshot freigegeben hat.

  Weitere Informationen zur AWS KMS-Schlüsselverwaltung für Amazon RDS finden Sie unter AWS KMS key-Verwaltung.

Gehen Sie wie folgt vor, um das Problem mit dem Standard-KMS-Schlüssel zu umgehen:

1. Erstellen Sie einen vom Kunden verwalteten Schlüssel und gewähren Sie Zugriff darauf..
2. Kopieren und Freigeben des Snapshots über das Quellkonto.
3. Kopieren des freigegebenen Snapshots in das Zielkonto.

Erstellen Sie einen vom Kunden verwalteten Schlüssel und gewähren Sie Zugriff darauf.

Zunächst erstellen Sie einen benutzerdefinierten KMS-Schlüssel in derselben AWS-Region wie der verschlüsselte DB-Cluster-Snapshot. Bei der Erstellung des vom Kunden verwalteten Schlüssels gewähren Sie für ein anderes AWS-Konto Zugriff darauf.

So erstellen Sie einen vom Kunden verwalteten Schlüssel und gewähren Zugriff darauf

1. Melden Sie sich bei der AWS-Managementkonsole über das Quell-AWS-Konto an.
2. Öffnen Sie die AWS KMS-Konsole unter https://console.aws.amazon.com/kms.

3. Um die AWS-Region zu ändern, verwenden Sie die Regionenauswahl in der oberen rechten Ecke der Seite.
4. Klicken Sie im Navigationsbereich auf Kundenverwaltete Schlüssel.
5. Klicken Sie auf Create key.
6. Gehen Sie auf der Seite Schlüssel konfigurieren wie folgt vor:
   a. Wählen Sie als Schlüsseltyp die Option Symmetrisch aus.
   b. Wählen Sie für Schlüsselnutzung die Option Verschlüsseln und entschlüsseln aus.
   c. Erweitern Sie Advanced options (Erweiterte Optionen).
   d. Wählen Sie für Schlüsselmaterialursprung KMS aus.
   e. Wählen Sie für Regionalität die Option Einzelregionaler Schlüssel aus.
   f. Wählen Sie Weiter.
7. Gehen Sie auf der Seite Etiketten hinzufügen wie folgt vor:
   a. Geben Sie für Alias einen Anzeigenamen für Ihren KMS-Schlüssel ein, z. B. **`share-snapshot`**.
   b. (Optional) Geben Sie eine Beschreibung für Ihren KMS-Schlüssel ein.
   c. (Optional) Fügen Sie Ihrem KMS-Schlüssel Tags hinzu.
   d. Wählen Sie Weiter aus.
8. Klicken Sie auf der Seite Definieren wichtiger administrativer Berechtigungen auf Weiter.
9. Gehen Sie auf der Seite Schlüsselnutzungsberechtigungen definieren wie folgt vor:
   a. Wählen Sie für Andere AWS-Konten die Option Ein weiteres AWS-Konto-Konto hinzufügen aus.
   b. Geben Sie die ID für das AWS-Konto ein, auf das Sie Zugriff gewähren möchten.

      Sie können Zugriff auf mehrere AWS-Konten gewähren.
   c. Wählen Sie Weiter aus.
10. Überprüfen Sie Ihren KMS-Schlüssel und wählen Sie dann Beenden aus.

Kopieren und Freigeben des Snapshots über das Quellkonto

Als Nächstes kopieren Sie den Quell-DB-Cluster-Snapshot mit dem vom Kunden verwalteten Schlüssel in einen neuen Snapshot. Dann geben Sie ihn für das Ziel AWS-Konto frei.

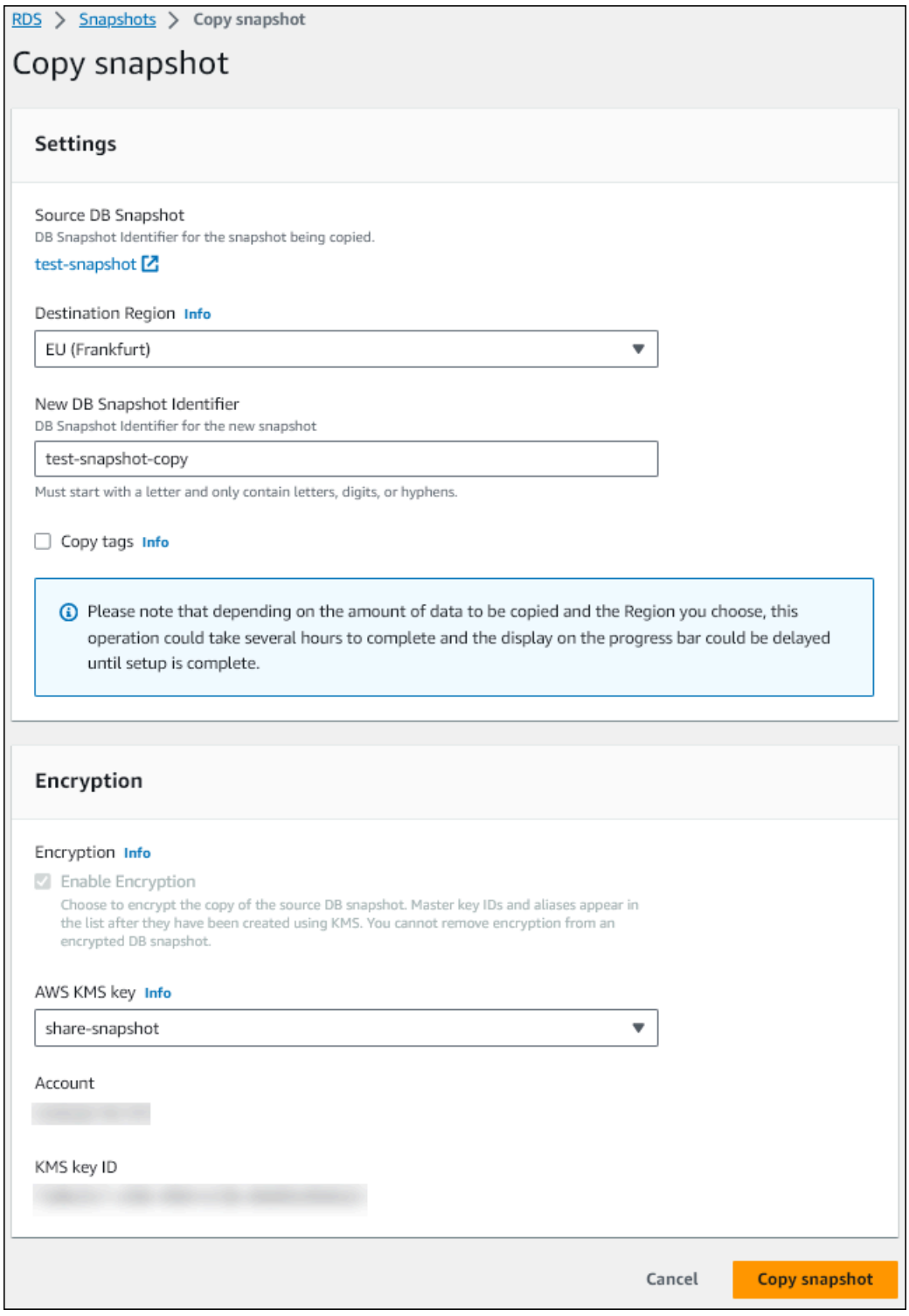
So kopieren Sie den Snapshot und geben Sie ihn frei

1. Melden Sie sich bei der AWS-Managementkonsole über das Quell-AWS-Konto an.
2. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
3. Wählen Sie im Navigationsbereich die Option Snapshots.
4. Wählen Sie den zu kopierenden DB-Cluster-Snapshot.
5. Wählen Sie für Actions (Aktionen) die Option Copy Snapshot (Snapshot kopieren).
6. Gehen Sie auf der Seite Snapshot kopieren wie folgt vor:

    a. Wählen Sie als Zielregion die AWS-Region aus, in der Sie den vom Kunden verwalteten Schlüssel im vorherigen Verfahren erstellt haben.

    b. Geben Sie den Namen für die Kopie des DB-Cluster-Snapshots in das Feld New DB Snapshot Identifier (Neue DB-Snapshot-Kennung) ein.

    c. Wählen Sie für AWS KMS key den vom Kunden verwalteten Schlüssel aus, den Sie erstellt haben.



d. Wählen Sie Snapshot kopieren aus.

7. Wenn die Snapshot-Kopie aus, wenn sie verfügbar ist.
8. Wählen Sie unter Aktionen die Option Snapshots freigeben aus.
9. Gehen Sie auf der Seite Snapshot-Berechtigungen wie folgt vor:

a. Geben Sie die AWS-Konto-ID ein, für die Sie die Snapshot-Kopie freigeben, und wählen Sie dann Hinzufügen aus.

b. Wählen Sie Speichern.

Der Snapshot wird freigegeben.

Kopieren des freigegebenen Snapshots in das Zielkonto

Jetzt können Sie den freigegebenen Snapshot in das Ziel AWS-Konto kopieren.

So kopieren Sie den freigegebenen Snapshot

1. Melden Sie sich über das Ziel-AWS-Konto bei der AWS-Managementkonsole an.
2. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
3. Wählen Sie im Navigationsbereich die Option Snapshots.
4. Wählen Sie die Registerkarte Für mich freigegeben aus.
5. Wählen Sie den freigegebenen Snapshot aus.
6. Wählen Sie für Aktionen die Option Snapshot kopieren aus.
7. Wählen Sie Ihre Einstellungen für das Kopieren des Snapshots wie im vorherigen Verfahren, verwenden Sie jedoch einen AWS KMS key, der zum Zielkonto gehört.

   Wählen Sie Snapshot kopieren aus.

Beenden der Snapshot-Freigabe

Um die Freigabe eines DB-Cluster-Snapshots zu beenden, entziehen Sie dem Ziel AWS-Konto die Berechtigung.

Konsole

So beenden Sie die Freigabe eines manuellen DB-Cluster-Snapshots für ein AWS-Konto

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.

3. Wählen Sie den manuellen Snapshot, für den Sie die Freigabe beenden möchten.
4. Wählen Sie die Option Actions (Aktionen) und anschließend Share Snapshot (Snapshot teilen) aus.
5. Um die Berechtigung für ein AWS-Konto zu entfernen, wählen Sie für den AWS-Kontobezeichner des betreffenden Kontos die Option Löschen aus der Liste der autorisierten Konten aus.
6. Wählen Sie Speichern, um Ihre Änderungen zu speichern.

CLI

Verwenden Sie den Parameter `--values-to-remove`, um einen AWS-Konto-Bezeichner aus der Liste zu entfernen.

Example das Stoppen der Snapshot-Freigabe

Im folgenden Beispiel wird verhindert, dass die AWS-Konto-ID 444455556666 den Snapshot wiederherstellt.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

Für Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

RDS-API

Um die Freigabeberechtigung für ein AWS-Konto zu entfernen, verwenden Sie den Vorgang `ModifyDBClusterSnapshotAttribute`, wobei `AttributeName` auf `restore` festgelegt und der Parameter `ValuesToRemove` angegeben ist. Um einen manuellen Snapshot als privat zu markieren, entfernen Sie den Wert `all` aus der Liste der Werte für das Attribut `restore`.

Exportieren von DB-Cluster-Daten nach Amazon S3

Sie können Daten aus einem Live-DB-Cluster von Amazon Aurora in einen Amazon-S3-Bucket exportieren. Der Exportprozess läuft im Hintergrund und beeinträchtigt nicht die Leistung Ihres aktiven DB-Clusters.

Standardmäßig werden alle Daten im DB-Cluster exportiert. Sie können sich jedoch dafür entscheiden, bestimmte Sätze von Datenbanken, Schemata oder Tabellen zu exportieren.

Amazon Aurora klont den DB-Cluster, extrahiert Daten aus dem Klon und speichert die Daten in einem Amazon-S3-Bucket. Die Daten werden in einem Apache Parquet-Format gespeichert, das komprimiert und konsistent ist. Einzelne Parquet-Dateien haben in der Regel eine Größe zwischen 1 und 10 MB.

Die schnellere Leistung, die Sie mit dem Export von Snapshot-Daten für Aurora MySQL Version 2 und Version 3 erzielen können, gilt nicht für den Export von DB-Cluster-Daten. Weitere Informationen finden Sie unter Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3.

Der Export des gesamten DB-Clusters wird Ihnen in Rechnung gestellt, unabhängig davon, ob Sie alle oder nur einen Teil der Daten exportieren. Weitere Informationen finden Sie auf der Seite Amazon-Aurora-Preise.

Nachdem die Daten exportiert wurden, können Sie die exportierten Daten direkt mit Tools wie Amazon Athena oder Amazon Redshift Spectrum analysieren. Weitere Informationen zur Verwendung Athena zum Lesen von Parquet-Daten finden Sie unter Parquet SerDe im Amazon Athena-Benutzerhandbuch. Weitere Informationen zur Verwendung von Redshift Spectrum zum Lesen von Parquet-Daten finden Sie unter COPY from columnar data formats im Amazon Redshift Database Developer Guide.

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur Versions- und Regionsverfügbarkeit beim Exportieren von DB-Cluster-Daten nach S3 finden Sie unter Unterstützte Regionen und Aurora-DB-Engines zum Exportieren von Cluster-Daten nach Amazon S3.

Sie verwenden den folgenden Prozess, um DB-Cluster-Daten in einen Amazon-S3-Bucket zu exportieren. Weitere Informationen finden Sie in den folgenden Abschnitten.

Übersicht über das Exportieren von DB-Cluster-Daten

1. Identifizieren Sie den DB-Cluster, dessen Daten Sie exportieren möchten.

2. Richten Sie den Zugriff auf den Amazon S3-Bucket ein.

   Ein Bucket ist ein Container für Amazon S3-Objekte oder -Dateien. Um die Informationen für den Zugriff auf einen Bucket bereitzustellen, führen Sie die folgenden Schritte aus:

   a. Identifizieren Sie den S3-Bucket, in den die DB-Cluster-Daten exportiert werden soll. Der S3-Bucket muss sich in derselben AWS-Region wie der Cluster befinden. Weitere Informationen finden Sie unter Identifizieren des Amazon-S3-Buckets, in den exportiert werden soll.

   b. Erstellen Sie eine AWS Identity and Access Management (IAM)-Rolle, die der DB-Cluster-Exportaufgabe Zugriff auf den S3-Bucket gewährt. Weitere Informationen finden Sie unter Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle.

3. Erstellen Sie einen symmetrischen Verschlüsselungs-AWS KMS key für die serverseitige Verschlüsselung. Der KMS-Schlüssel wird von der Cluster-Exportaufgabe verwendet, um serverseitige AWS KMS-Verschlüsselung beim Schreiben der Exportdaten in S3 einzurichten.

   Die KMS-Schlüsselrichtlinie muss die beiden Berechtigungen `kms:CreateGrant` und `kms:DescribeKey` enthalten. Weitere Informationen zur Verwendung von KMS-Schlüsseln in Amazon Aurora finden Sie unter AWS KMS key-Verwaltung.

   Wenn Ihre KMS-Schlüsselrichtlinie eine Anweisung zur Zugriffsverweigerung enthält, müssen Sie den AWS-Service-Prinzipal `export.rds.amazonaws.com` explizit ausschließen.

   Sie können einen KMS-Schlüssel in Ihrem AWS-Konto oder Sie können einen kontoübergreifenden KMS-Schlüssel verwenden. Weitere Informationen finden Sie unter Ein kontoübergreifendes Konto verwenden AWS KMS key.

4. Exportieren Sie den DB-Cluster mit der Konsole oder dem CLI-Befehl `start-export-task` nach Amazon S3. Weitere Informationen finden Sie unter Erstellen von DB-Cluster-Exportaufgaben.

5. Um auf Ihre exportierten Daten im Amazon S3-Bucket zuzugreifen, siehe Hochladen, Herunterladen und Verwalten von Objekten im Amazon Simple Storage Service User Guide.

In den folgenden Abschnitten erfahren Sie, wie Sie Exportaufgaben von DB-Clustern einrichten, exportieren und überwachen sowie Fehler bei diesen beheben.

Themen

- Überlegungen zu Exporten von DB-Clustern

- Einrichten des Zugriffs auf einen Amazon-S3-Bucket
- Erstellen von DB-Cluster-Exportaufgaben
- Überwachen von DB-Cluster-Exportaufgaben
- Abbrechen einer DB-Cluster-Exportaufgabe
- Fehlerbehebung bei Problemen mit DB-Cluster-Exporten

Überlegungen zu Exporten von DB-Clustern

In den folgenden Abschnitten erfahren Sie mehr über Einschränkungen, Dateibenennungskonventionen sowie Datenkonvertierung und Speicherung beim Exportieren von DB-Cluster-Daten nach Amazon S3.

Themen

- Einschränkungen
- Benennungskonvention für Dateien
- Datenkonvertierung und Speicherformat

Einschränkungen

Das Exportieren von DB-Cluster-Daten nach Amazon S3 unterliegt folgenden Einschränkungen:

- Sie können nicht mehrere Exportaufgaben für denselben DB-Cluster gleichzeitig ausführen. Dies gilt sowohl für vollständige als auch Teilexporte.
- Sie können bis zu fünf gleichzeitige DB-Snapshot-Exportaufgaben pro AWS-Konto durchführen.
- DB-Cluster von Aurora Serverless v1 unterstützen keine Exporte nach S3.
- Aurora MySQL und Aurora PostgreSQL unterstützen Exporte nach S3 nur für den bereitgestellten Engine-Modus.
- Exporte nach S3 unterstützen keine S3-Präfixe, die einen Doppelpunkt (:) enthalten.
- Die folgenden Zeichen im S3-Dateipfad werden während des Exports in Unterstriche (_) konvertiert:

```
\ ` " (space)
```

- Wenn eine Datenbank, ein Schema oder eine Tabelle andere Zeichen als den folgenden enthält, wird ein teilweiser Export nicht unterstützt. Sie können jedoch den gesamten DB-Cluster exportieren.
  - Lateinische Buchstaben (A–Z)
  - Ziffern (0–9)
  - Dollar-Symbol ($)
  - Unterstrich (_)
- Leerzeichen ( ) und bestimmte andere Zeichen werden in den Spaltennamen von Datenbanktabellen nicht unterstützt. Tabellen mit den folgenden Zeichen in Spaltennamen werden beim Export übersprungen:

  ```
  , ; { } ( ) \n \t = (space)
  ```

- Tabellen mit Schrägstrichen (/) im Namen werden beim Export übersprungen.
- Temporäre und nicht protokollierte Tabellen von Aurora PostgreSQL werden beim Export übersprungen.
- Wenn die Daten ein großes Objekt wie BLOB oder CLOB mit einer Größe von 500 MB oder mehr enthalten, schlägt der Export fehl.
- Wenn eine Zeile in einer Tabelle ca. 2 GB groß oder noch größer ist, wird die Tabelle beim Export übersprungen.
- Bei Teilexporten beträgt die maximale Größe der `ExportOnly`-Liste 200 KB.
- Es wird dringend empfohlen, für jede Exportaufgabe einen eindeutigen Namen zu verwenden. Wenn Sie keinen eindeutigen Aufgabennamen verwenden, erhalten Sie möglicherweise die folgende Fehlermeldung:

  ExportTaskAlreadyExistsFault: Beim Aufrufen des StartExportTask-Vorgangs ist ein Fehler aufgetreten (ExportTaskAlreadyExists): Die Exportaufgabe mit der ID *xxxxx* ist bereits vorhanden.
- Da einige Tabellen möglicherweise übersprungen werden, empfehlen wir, nach dem Export die Anzahl der Zeilen und Tabellen in den Daten zu überprüfen.

Benennungskonvention für Dateien

Exportierte Daten für bestimmte Tabellen werden im Format *base_prefix*/*files* gespeichert, wobei das Basispräfix folgendes ist:

```
export_identifier/database_name/schema_name.table_name/
```

Zum Beispiel:

```
export-1234567890123-459/rdststcluster/mycluster.DataInsert_7ADB5D19965123A2/
```

Ausgabedateien verwenden die folgende Benennungskonvention, wobei *partition_index* alphanumerisch ist:

```
partition_index/part-00000-random_uuid.format-based_extension
```

Zum Beispiel:

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
    a/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
```

Die Namenskonvention für Dateien kann geändert werden. Daher empfehlen wir beim Lesen von Zieltabellen, dass Sie alles innerhalb des Basispräfixes für die Tabelle lesen.

Datenkonvertierung und Speicherformat

Wenn Sie einen DB-Cluster in einen Amazon-S3-Bucket exportieren, konvertiert Amazon Aurora Daten in das Parquet-Format, exportiert Daten darin und speichert Daten in diesem Format. Weitere Informationen finden Sie unter Datenkonvertierung beim Exportieren in einen Amazon S3-Bucket.

Einrichten des Zugriffs auf einen Amazon-S3-Bucket

Sie identifizieren den Amazon-S3-Bucket und gewähren der DB-Cluster-Exportaufgabe die Berechtigung, darauf zuzugreifen.

Themen

- Identifizieren des Amazon-S3-Buckets, in den exportiert werden soll
- Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle
- Einen kontoübergreifenden Amazon-S3-Bucket verwenden

Identifizieren des Amazon-S3-Buckets, in den exportiert werden soll

Identifizieren Sie den Amazon-S3-Bucket, in den die DB-Cluster-Daten exportiert werden sollen. Verwenden Sie einen vorhandenen S3-Bucket oder erstellen Sie einen neuen S3-Bucket.

> **Note**
>
> Der S3-Bucket muss sich in derselben AWS Region wie der DB-Cluster befinden.

Weitere Informationen zur Arbeit mit Amazon S3-Buckets finden Sie im Amazon Simple Storage Service User Guide:

- Wie zeige ich die Eigenschaften für einen S3-Bucket an?
- Wie aktiviere ich die Standardverschlüsselung für einen Amazon S3-Bucket?
- Wie erstelle ich einen S3-Bucket?

Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle

Bevor Sie DB-Cluster-Daten nach Amazon S3 exportieren, gewähren Sie den Exportaufgaben Schreibzugriff auf den Amazon-S3-Bucket.

Zum Gewähren dieser Berechtigung erstellen Sie eine IAM-Richtlinie, die Zugriff auf den Bucket ermöglicht. Erstellen Sie dann eine IAM-Rolle und fügen Sie der Rolle die Richtlinie an. Die IAM-Rolle können Sie später Ihrer DB-Cluster-Exportaufgabe zuweisen.

> **Important**
>
> Wenn Sie planen, den für den Export Ihres DB-Clusters AWS-Managementkonsole zu verwenden, können Sie festlegen, dass die IAM-Richtlinie und die Rolle automatisch erstellt werden, wenn Sie den DB-Cluster exportieren. Detaillierte Anweisungen finden Sie unter Erstellen von DB-Cluster-Exportaufgaben.

So erteilen Sie Aufgaben Zugriff auf Amazon S3

1. Erstellen Sie eine IAM-Richtlinie. Diese Richtlinie bietet dem Bucket und den Objekten Berechtigungen, die Ihrer DB-Cluster-Exportaufgabe Zugriff auf Amazon S3 erlauben.

Die Richtlinie muss die folgenden erforderlichen Aktionen enthalten, um die Übertragung von Dateien aus Amazon Aurora in einen S3-Bucket zu erlauben:

- `s3:PutObject*`
- `s3:GetObject*`
- `s3:ListBucket`
- `s3:DeleteObject*`
- `s3:GetBucketLocation`

Fügen Sie in die Richtlinie die folgenden Ressourcen zur Identifizierung des S3-Buckets und der Objekte im Bucket ein. Die folgende Liste von Ressourcen zeigt das ARN-Format (Amazon-Ressourcenname) für den Amazon S3-Zugriff.

- `arn:aws:s3:::amzn-s3-demo-bucket`
- `arn:aws:s3:::amzn-s3-demo-bucket/*`

Weitere Informationen zur Erstellung einer IAM-Richtlinie für Amazon Aurora finden Sie unter Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff. Siehe auch Tutorial: Erstellen und Anfügen Ihrer ersten vom Kunden verwalteten Richtlinie im IAM-Benutzerhandbuch.

Mit dem folgenden AWS CLI Befehl wird eine IAM-Richtlinie `ExportPolicy` mit diesen Optionen erstellt. Sie gewährt Zugriff auf einen Bucket namens *amzn-s3-demo-bucket*.

> **Note**
>
> Nachdem Sie die Richtlinie erstellt haben, notieren Sie den ARN der Richtlinie. Sie benötigen den ARN für einen nachfolgenden Schritt, in dem Sie die Richtlinie an eine IAM-Rolle anhängen.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
```

```
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. Erstellen Sie eine IAM-Rolle, damit Aurora diese IAM-Rolle in Ihrem Namen übernehmen kann, um auf Ihre Amazon-S3-Buckets zuzugreifen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen IAM-Benutzer im IAM-Benutzerhandbuch.

   Das folgende Beispiel zeigt, wie der AWS CLI Befehl verwendet wird, um eine Rolle mit dem Namen `rds-s3-export-role` zu erstellen.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. Fügen Sie die erstellte IAM-Richtlinie der IAM-Rolle an, die Sie erstellt haben.

Mit dem folgenden AWS CLI Befehl wird die zuvor erstellte Richtlinie an die angegebene `rds-s3-export-role` Rolle angehängt. Ersetzen Sie *`your-policy-arn`* durch den Richtlinien-ARN, den Sie in einem früheren Schritt notiert haben.

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-export-role
```

Einen kontoübergreifenden Amazon-S3-Bucket verwenden

Sie können S3-Buckets kontenübergreifend AWS verwenden. Weitere Informationen finden Sie unter Einen kontoübergreifenden Amazon-S3-Bucket verwenden.

Erstellen von DB-Cluster-Exportaufgaben

Erstellen Sie Exportaufgaben zum Exportieren von Daten aus Ihrem Aurora-DB-Cluster in einen Amazon-S3-Bucket. Sie können bis zu fünf gleichzeitige DB-Cluster-Exportaufgaben pro AWS-Konto durchführen.

> **Note**
>
> Das Exportieren von DB-Cluster-Daten kann je nach Datenbanktyp und -größe eine Weile dauern. Die Exportaufgabe klont zuerst die gesamte Datenbank und skaliert sie, bevor die Daten in Amazon S3 extrahiert werden. Der Fortschritt des Vorgangs während dieser Phase wird als Starting (Startet) angezeigt. Wenn die Aufgabe zum Exportieren von Daten zu S3 wechselt, wird der Fortschritt als In progress (In Bearbeitung) angezeigt.
> Die Zeit, die für den Export benötigt wird, hängt von den in der Datenbank gespeicherten Daten ab. Beispielsweise exportieren Tabellen mit gut verteilten numerischen Primärschlüssel- oder Indexspalten am schnellsten. Bei Tabellen, die keine Spalte enthalten, die für die Partitionierung geeignet ist, und bei Tabellen mit nur einem Index für eine auf Zeichenfolgen basierende Spalte, dauert dies länger, da der Export einen langsameren Single-Thread-Prozess verwendet.

Sie können DB-Cluster-Daten mit Hilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API nach Amazon S3 exportieren.

Wenn Sie eine Lambda-Funktion zum Exportieren der DB-Cluster-Daten verwenden, fügen Sie die Aktion `kms:DescribeKey` der Lambda-Funktionsrichtlinie hinzu. Weitere Informationen finden Sie unter AWS Lambda Berechtigungen.

Konsole

Die Konsolenoption Export to Amazon S3 (Nach Amazon S3 exportieren) wird nur für DB-Cluster angezeigt, die nach Amazon S3 exportiert werden können. Ein DB-Cluster ist aus folgenden Gründen möglicherweise nicht für den Export verfügbar:

- Die DB-Engine wird für den S3-Export nicht unterstützt.
- Die DB-Cluster-Version wird für den S3-Export nicht unterstützt.
- Der S3-Export wird in der AWS-Region, in der der DB-Cluster erstellt wurde, nicht unterstützt.

So exportieren Sie DB-Cluster-Daten

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den DB-Cluster aus, dessen Daten Sie exportieren möchten.
4. Wählen Sie für Actions (Aktionen) die Option Export to Amazon S3 (Nach Amazon S3 exportieren) aus.

   Das Fenster Export to Amazon S3 (Nach Amazon S3 exportieren) erscheint.
5. Geben Sie für Export identifier (Export-ID) einen Namen ein, um die Exportaufgabe zu identifizieren. Dieser Wert wird auch für den Namen der im S3-Bucket erstellten Datei verwendet.
6. Wählen Sie die zu exportierenden Daten aus:

   - Wählen Sie All (Alle) aus, um alle Daten im DB-Cluster zu exportieren.
   - Wählen Sie Partial (Teilweise) aus, um bestimmte Teile des DB-Clusters zu exportieren. Um zu ermitteln, welche Teile des Clusters exportiert werden sollen, geben Sie eine oder mehrere Datenbanken, Schemas oder Tabellen für Bezeichner ein, getrennt durch Leerzeichen.

     Verwenden Sie das folgende Format:

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
     [.tablen]
     ```

Zum Beispiel:

```
mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
 mydatabase2.myschema2.mytable2
```

7. Wählen Sie für S3 bucket (S3-Bucket) den Bucket aus, in den exportiert werden soll.

   Um die exportierten Daten einem Ordnerpfad im S3-Bucket zuzuordnen, geben Sie den optionalen Pfad für S3 prefix (S3-Präfix) ein.

8. Wählen Sie für die IAM-Rolle entweder eine Rolle, die Ihnen Schreibzugriff auf den gewählten S3-Bucket gewährt, oder erstellen Sie eine neue Rolle.

   - Wenn Sie eine Rolle durch Befolgen der Schritte in Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle erstellt haben, wählen Sie diese Rolle.
   - Wenn Sie keine Rolle erstellt haben, die Ihnen Schreibzugriff auf den von Ihnen gewählten S3-Bucket gewährt, wählen Sie Create a new role (Neue Rolle erstellen) aus, um die Rolle automatisch zu erstellen. Geben Sie dann unter IAM role name (IAM-Rollenname) einen Namen für die Rolle ein.

9. Geben Sie für KMS key (KMS-Schlüssel) den ARN für den Schlüssel ein, der für die Verschlüsselung der exportierten Daten verwendet werden soll.
10. Wählen Sie Export to Amazon S3 (Nach Amazon S3 exportieren) aus.

AWS CLI

Um DB-Cluster-Daten mit der AWS CLI nach Amazon S3 zu exportieren, verwenden Sie den Befehl start-export-task mit den folgenden erforderlichen Optionen:

- `--export-task-identifier`
- `--source-arn` – Der Amazon-Ressourcenname (ARN) des DB-Clusters.
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

In den folgenden Beispielen heißt die Exportaufgabe *`my-cluster-export`*, die die Daten in einen S3-Bucket namens *`amzn-s3-demo-destination-bucket`* exportiert.

Example

Für Linux, macOS oder Unix:

```
aws rds start-export-task \
    --export-task-identifier my-cluster-export \
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Für Windows:

```
aws rds start-export-task ^
    --export-task-identifier my-DB-cluster-export ^
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

Beispiel für eine Ausgabe folgt.

```
{
    "ExportTaskIdentifier": "my-cluster-export",
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "IamRoleArn": "arn:aws:iam:123456789012:role/ExportTest",
    "KmsKeyId": "my-key",
    "Status": "STARTING",
    "PercentProgress": 0,
    "TotalExtractedDataInGB": 0,
}
```

Wenn Sie einen Ordnerpfad im S3-Bucket für den DB-Cluster-Export bereitstellen möchten, nehmen Sie die Option `--s3-prefix` in den Befehl start-export-task auf.

RDS-API

Wenn Sie DB-Cluster-Daten Snapshot mit der Amazon-RDS-API nach Amazon S3 exportieren möchten, verwenden Sie die Operation StartExportTask mit den folgenden erforderlichen Parametern:

- `ExportTaskIdentifier`
- `SourceArn` – der ARN des DB-Clusters
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

Überwachen von DB-Cluster-Exportaufgaben

Sie können DB-Cluster-Exporte mit der AWS-Managementkonsole, der AWS CLI oder der RDS-API überwachen.

Konsole

So überwachen Sie DB-Cluster-Exporte

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Exports in Amazon S3 (Exporte in Amazon S3) aus.

   DB-Cluster-Exporte werden in der Spalte Source type (Quelltyp) angegeben. Der Exportstatus wird in der Spalte Status angezeigt.
3. Um ausführliche Informationen über einen bestimmten DB-Cluster-Export anzuzeigen, wählen Sie die Exportaufgabe aus.

AWS CLI

Um DB-Cluster-Exporte mit der AWS CLI zu überwachen, verwenden Sie den Befehl describe-export-tasks.

Das folgende Beispiel zeigt, wie Sie aktuelle Informationen über alle Ihre DB-Cluster-Exporte anzeigen können.

Example

```
aws rds describe-export-tasks

{
```

```
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2022-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:parameter-groups-
test"
        },
{
            "Status": "COMPLETE",
            "TaskStartTime": "2022-10-31T20:58:06.998Z",
            "TaskEndTime": "2022-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2022-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket amzn-s3-demo-bucket2 isn't located in the
 current AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
```

```
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-30-06-45"
        }
    ]
}
```

Wenn Sie Informationen über eine bestimmte Exportaufgabe anzeigen möchten, nehmen Sie die Option `--export-task-identifier` in den Befehl `describe-export-tasks` auf. Um die Ausgabe zu filtern, fügen Sie die Option `--Filters` ein. Weitere Optionen finden Sie beim describe-export-tasks-Befehl.

RDS-API

Um Informationen über DB-Cluster-Exporte mit der Amazon-RDS-API anzuzeigen, verwenden Sie die DescribeExportTasks-Operation.

Um den Abschluss des Export-Workflows zu verfolgen oder einen anderen Workflow zu initiieren, können Sie Themen von Amazon Simple Notification Service abonnieren. Weitere Informationen zu Amazon SNS finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

Abbrechen einer DB-Cluster-Exportaufgabe

Sie können eine DB-Cluster-Exportaufgabe mit Hilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API abbrechen.

> **Note**
>
> Durch das Abbrechen einer Exportaufgabe werden keine Daten entfernt, die nach Amazon S3 exportiert wurden. Informationen zum Löschen der Daten über die Konsole finden Sie unter Wie lösche ich Objekte aus einem S3-Bucket?. Um die Daten mit der CLI zu löschen, verwenden Sie den Befehl delete-object.

Konsole

So brechen Sie eine DB-Cluster-Exportaufgabe ab

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich die Option Exports in Amazon S3 (Exporte in Amazon S3) aus.

   DB-Cluster-Exporte werden in der Spalte Source type (Quelltyp) angegeben. Der Exportstatus wird in der Spalte Status angezeigt.
3. Wählen Sie die Exportaufgabe aus, die Sie abbrechen möchten.
4. Klicken Sie auf Abbrechen.
5. Wählen Sie die Bestätigungsseite Cancel export task (Exportaufgabe abbrechen) aus.

AWS CLI

Um eine Exportaufgabe mit der AWS CLI abzubrechen, verwenden Sie den Befehl cancel-export-task aus. Der Befehl erfordert die Option `--export-task-identifier`.

Example

```
aws rds cancel-export-task --export-task-identifier my-export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my-export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:export-example-1"
}
```

RDS-API

Wenn Sie eine Exportaufgabe mit der Amazon-RDS-API abbrechen möchten, verwenden Sie den Befehl CancelExportTask mit dem Parameter `ExportTaskIdentifier`.

Fehlerbehebung bei Problemen mit DB-Cluster-Exporten

Nutzen Sie die folgenden Abschnitte, um Fehlermeldungen und PostgreSQL-Berechtigungsfehler bei DB-Cluster-Exporten nach Amazon S3 zu beheben.

Fehlermeldungen für Amazon-S3-Exportaufgaben

In der folgenden Tabelle werden die Nachrichten beschrieben, die zurückgegeben werden, wenn Amazon-S3-Exportaufgaben fehlschlagen.

Fehlernachricht	Beschreibung
Der Quell-DB-Cluster konnte nicht gefunden werden oder es konnte nicht darauf zugegriffen werden: [Clustername]	Der Quell-DB-Cluster kann nicht geklont werden.
Ein unbekannter interner Fehler ist aufgetreten.	Die Aufgabe ist fehlgeschlagen, da eine unbekannte Störung, eine Ausnahme oder ein Fehler aufgetreten ist.
Beim Schreiben der Metadaten der Exportaufgabe in den S3-Bucket [Bucket-Name] ist ein unbekannter interner Fehler aufgetreten.	Die Aufgabe ist fehlgeschlagen, da eine unbekannte Störung, eine Ausnahme oder ein Fehler aufgetreten ist.
Der RDS-Export konnte die Metadaten der Exportaufgabe nicht schreiben, da er die IAM-Rolle [Rolle ARN] nicht übernehmen kann.	Die Exportaufgabe geht davon aus, dass Ihre IAM-Rolle überprüft, ob es erlaubt ist Metadaten in Ihren S3-Bucket zu schreiben Wenn die Aufgabe Ihre IAM-Rolle nicht übernehmen kann, schlägt sie fehl.
Der RDS-Export konnte die Metadaten der Exportaufgabe nicht in den S3-Bucket [Bucket-Name] mit der IAM-Rolle [Rolle ARN] mit dem KMS-Schlüssel [Schlüssel-ID] schreiben. Fehlercode: [Fehlercode]	Eine oder mehrere Berechtigungen fehlen, sodass die Exportaufgabe nicht auf den S3-Bucket zugreifen kann. Diese Fehlermeldung wird ausgelöst, wenn einer der folgenden Fehlercodes empfangen wird: • `AWSSecurityTokenServiceException` mit dem Fehlercode `AccessDenied` • `AmazonS3Exception` mit dem Fehlercode `NoSuchBucket` , `AccessDenied` , `KMS.KMSInvalidStateException` , `403 Forbidden` , oder `KMS.DisabledException`

Fehlernachricht	Beschreibung
	Diese Fehlercodes weisen darauf hin, dass für die IAM-Rolle, den S3-Bucket oder den KMS-Schlüssel Einstellungen falsch konfiguriert sind.
Die IAM-Rolle [Rolle ARN] ist nicht berechtigt, [S3-Aktion] im S3-Bucket [Bucket-Name] aufzurufen. Überprüfen Sie Ihre Berechtigungen und versuchen Sie den Export erneut.	Die IAM-Richtlinie ist falsch konfiguriert. Die Berechtigung für die spezifische S3-Aktion für den S3-Bucket fehlt, was zu einem Fehler der Exportaufgabe führt.
Fehler bei der Überprüfung des KMS-Schlüssels. Überprüfen Sie die Anmeldeinformation Ihres KMS-Schlüssels und versuchen Sie es erneut.	Die Überprüfung der KMS-Schlüsselanmeldeinformationen ist fehlgeschlagen.
Die Überprüfung der S3-Anmeldeinformation ist fehlgeschlagen. Überprüfen Sie die Berechtigungen für Ihren S3-Bucket und Ihre IAM-Richtlinie.	Die Überprüfung der S3-Anmeldeinformation ist fehlgeschlagen.
Der S3-Bucket [Bucket-Name] ist ungültig. Entweder befindet sie sich nicht in der aktuellen AWS-Region oder es gibt sie nicht. Überprüfen Sie Ihren S3-Bucket-Namen und versuchen Sie den Export erneut.	Der S3-Bucket ist ungültig.
Der S3-Bucket [Bucket-Name] befindet sich nicht in der aktuellen AWS-Region. Überprüfen Sie Ihren S3-Bucket-Namen und versuchen Sie den Export erneut.	Der S3-Bucket liegt in der falschen AWS-Region.

Fehlerbehebung bei PostgreSQL-Berechtigungsfehlern

Beim Exportieren von PostgreSQL-Datenbanken in Amazon S3 wird möglicherweise ein `PERMISSIONS_DO_NOT_EXIST`-Fehler angezeigt, der besagt, dass bestimmte Tabellen

übersprungen wurden. Dieser Fehler tritt normalerweise auf, wenn der Superuser, den Sie beim Erstellen des DB-Clusters angegeben haben, keine Berechtigungen für den Zugriff auf diese Tabellen besitzt.

Führen Sie den folgenden Befehl aus, um diesen Fehler zu beheben:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

Weitere Informationen zu Superuser-Berechtigungen finden Sie unter Berechtigungen von Hauptbenutzerkonten.

Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3

Sie können DB-Cluster-Snapshot-Daten in einen Amazon-S3-Bucket exportieren. Der Exportprozess läuft im Hintergrund und beeinträchtigt nicht die Leistung Ihres aktiven DB-Clusters.

Wenn Sie einen DB-Cluster-Snapshot exportieren, extrahiert Amazon Aurora Daten aus dem Snapshot und speichert sie in einem Amazon-S3-Bucket. Sie können manuelle Snapshots und automatisierte System-Snapshots exportieren. Standardmäßig werden alle Daten im Snapshot exportiert. Sie können sich jedoch dafür entscheiden, bestimmte Sätze von Datenbanken, Schemata oder Tabellen zu exportieren.

> **Note**
>
> Um Daten eines DB-Cluster-Snapshots zu exportieren, muss der Snapshot wiederhergestellt werden. Die Wiederherstellungszeiten werden von verschiedenen Faktoren beeinflusst, z. B. vom Ausmaß des Netzwerkverkehrs, den ein AWS-Region-Benutzer in Verhältnis zu dessen Bandbreite empfängt. Bei einem plötzlichen Anstieg des Datenverkehrs kann es zu längeren Bearbeitungszeiten als erwartet kommen.
> Eine Alternative zur Verkürzung der S3-Exportzeiten für Aurora-Datenbanken ist ein Live-Export von DB-Clustern nach S3. Der DB-Cluster-Export hat kürzere Startzeiten als der DB-Snapshot-Export, da kein Snapshot wiederhergestellt werden muss. Weitere Informationen finden Sie unter Exportieren von DB-Cluster-Daten nach Amazon S3.

Die Daten werden in einem Apache Parquet-Format gespeichert, das komprimiert und konsistent ist. Einzelne Parquet-Dateien haben in der Regel eine Größe zwischen 1 und 10 MB.

Nachdem die Daten exportiert wurden, können Sie die exportierten Daten direkt mit Tools wie Amazon Athena oder Amazon Redshift Spectrum analysieren. Weitere Informationen zur Verwendung Athena zum Lesen von Parquet-Daten finden Sie unter Parquet SerDe im Amazon Athena-Benutzerhandbuch. Weitere Informationen zur Verwendung von Redshift Spectrum zum Lesen von Parquet-Daten finden Sie unter COPY from columnar data formats im Amazon Redshift Database Developer Guide.

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur Versions- und Regionsverfügbarkeit beim Exportieren von Snapshot-Daten von DB-Cluster nach S3 finden Sie unter

Unterstützte Regionen und Aurora-DB-Engines zum Exportieren von Snapshot-Daten nach Amazon S3.

Sie verwenden den folgenden Prozess, um DB-Snapshot-Daten in eine Amazon S3 einen Bucket zu exportieren. Weitere Informationen finden Sie in den folgenden Abschnitten.

Übersicht über das Exportieren von Snapshot-Daten

1. Identifizieren Sie den zu exportierenden Snapshot.

   Verwenden Sie einen vorhandenen automatischen oder manuellen Snapshot oder erstellen Sie einen manuellen Snapshot einer DB-Instance.
2. Richten Sie den Zugriff auf den Amazon S3-Bucket ein.

   Ein Bucket ist ein Container für Amazon S3-Objekte oder -Dateien. Um die Informationen für den Zugriff auf einen Bucket bereitzustellen, führen Sie die folgenden Schritte aus:

   a. Identifizieren Sie den S3-Bucket, in den der Snapshot exportiert werden soll. Der S3-Bucket muss sich in derselben AWS-Region wie der Snapshot befinden. Weitere Informationen finden Sie unter Identifizieren des Amazon-S3-Buckets, in den exportiert werden soll.

   b. Erstellen Sie eine AWS Identity and Access Management (IAM)-Rolle, die der Snapshot-Exportaufgabe Zugriff auf den S3 Bucket gewährt. Weitere Informationen finden Sie unter Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle.
3. Erstellen Sie einen symmetrischen Verschlüsselungs-AWS KMS key für die serverseitige Verschlüsselung. Der KMS-Schlüssel wird von der Snapshot-Exportaufgabe verwendet, um AWS KMS serverseitige Verschlüsselung beim Schreiben der Exportdaten in S3 einzurichten.

   Die KMS-Schlüsselrichtlinie muss die beiden Berechtigungen `kms:CreateGrant` und `kms:DescribeKey` enthalten. Weitere Informationen zur Verwendung von KMS-Schlüsseln in Amazon Aurora finden Sie unter AWS KMS key-Verwaltung.

   Wenn Ihre KMS-Schlüsselrichtlinie eine Anweisung zur Zugriffsverweigerung enthält, müssen Sie den AWS-Service-Prinzipal `export.rds.amazonaws.com` explizit ausschließen.

   Sie können einen KMS-Schlüssel in Ihrem AWS-Konto oder Sie können einen kontoübergreifenden KMS-Schlüssel verwenden. Weitere Informationen finden Sie unter Ein kontoübergreifendes Konto verwenden AWS KMS key.
4. Exportieren Sie den Snapshot mit der Konsole oder dem CLI-Befehl `start-export-task` nach Amazon S3. Weitere Informationen finden Sie unter Erstellen von Snapshot-Exportaufgaben.

5. Um auf Ihre exportierten Daten im Amazon S3-Bucket zuzugreifen, siehe Hochladen, Herunterladen und Verwalten von Objekten im Amazon Simple Storage Service User Guide.

In den folgenden Abschnitten erfahren Sie, wie Sie Snapshot-Exportaufgaben von DB-Clustern einrichten, exportieren und überwachen sowie Fehler bei diesen beheben.

Themen

- Überlegungen zu Exporten von DB-Cluster-Snapshots
- Einrichten des Zugriffs auf einen Amazon-S3-Bucket
- Erstellen von Snapshot-Exportaufgaben
- Überwachung von Snapshot-Exporten
- Abbrechen einer Snapshot-Exportaufgabe
- Exportleistung in Aurora MySQL
- Fehlerbehebung bei Snapshot-Exporten

Überlegungen zu Exporten von DB-Cluster-Snapshots

Einschränkungen

Das Exportieren von DB-Snapshot-Daten nach Amazon S3 hat die folgenden Einschränkungen:

- Sie können nicht mehrere Exportaufgaben für denselben DB-Cluster-Snapshot gleichzeitig ausführen. Dies gilt sowohl für vollständige als auch Teilexporte.
- Sie können bis zu fünf gleichzeitige DB-Snapshot-Exportaufgaben pro AWS-Konto durchführen.
- Sie können Snapshot-Daten nicht von Aurora Serverless v1-DB-Clustern nach S3 exportieren.
- Exporte nach S3 unterstützen keine S3-Präfixe, die einen Doppelpunkt (:) enthalten.
- Die folgenden Zeichen im S3-Dateipfad werden während des Exports in Unterstriche (_) konvertiert:

```
\ ` " (space)
```

- Wenn eine Datenbank, ein Schema oder eine Tabelle andere Zeichen als den folgenden enthält, wird ein teilweiser Export nicht unterstützt. Sie können jedoch den gesamten DB-Snapshot exportieren.
  - Lateinische Buchstaben (A–Z)

  - Ziffern (0–9)
  - Dollar-Symbol ($)
  - Unterstrich (_)
- Leerzeichen ( ) und bestimmte andere Zeichen werden in den Spaltennamen von Datenbanktabellen nicht unterstützt. Tabellen mit den folgenden Zeichen in Spaltennamen werden beim Export übersprungen:

```
, ; { } ( ) \n \t = (space)
```

- Tabellen mit Schrägstrichen (/) im Namen werden beim Export übersprungen.
- Temporäre und nicht protokollierte Tabellen von Aurora PostgreSQL werden beim Export übersprungen.
- Wenn die Daten ein großes Objekt wie BLOB oder CLOB mit einer Größe von 500 MB oder mehr enthalten, schlägt der Export fehl.
- Wenn eine Zeile in einer Tabelle ca. 2 GB groß oder noch größer ist, wird die Tabelle beim Export übersprungen.
- Bei Teilexporten beträgt die maximale Größe der `ExportOnly`-Liste 200 KB.
- Es wird dringend empfohlen, für jede Exportaufgabe einen eindeutigen Namen zu verwenden. Wenn Sie keinen eindeutigen Aufgabennamen verwenden, erhalten Sie möglicherweise die folgende Fehlermeldung:

  ExportTaskAlreadyExistsFault: Beim Aufrufen des StartExportTask-Vorgangs ist ein Fehler aufgetreten (ExportTaskAlreadyExists): Die Exportaufgabe mit der ID *xxxxx* ist bereits vorhanden.
- Sie können einen Snapshot löschen, während Sie seine Daten nach S3 exportieren, aber die Speicherkosten für diesen Snapshot werden Ihnen so lange in Rechnung gestellt, bis die Exportaufgabe abgeschlossen ist.
- Sie können exportierte Snapshot-Daten aus S3 nicht in einem neuen DB-Cluster wiederherstellen.

Benennungskonvention für Dateien

Exportierte Daten für bestimmte Tabellen werden im Format *base_prefix*/*files* gespeichert, wobei das Basispräfix folgendes ist:

```
export_identifier/database_name/schema_name.table_name/
```

Zum Beispiel:

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

Es gibt zwei Konventionen für die Benennung von Dateien.

- Aktuelle Konvention:

  ```
  batch_index/part-partition_index-random_uuid.format-based_extension
  ```

  Der Batch-Index ist eine Sequenznummer, die einen aus der Tabelle gelesenen Datenstapel darstellt. Wenn wir Ihre Tabelle nicht in kleine Datenblöcke aufteilen können, die parallel exportiert werden können, gibt es mehrere Batch-Indizes. Das Gleiche passiert bei einer Partitionierung Ihrer Tabelle in mehrere Tabellen. Es wird mehrere Batch-Indizes geben, einen für jede Tabellenpartition Ihrer Haupttabelle.

  Wenn wir Ihre Tabelle in kleine Datenblöcke aufteilen können, die parallel gelesen werden, gibt es nur den Batch-Index-Ordner 1.

  Im Batch-Index-Ordner befinden sich eine oder mehrere Parquet-Dateien, die die Daten Ihrer Tabelle enthalten. Das Präfix des Parquet-Dateinamens ist `part-`*`partition_index`*. Wenn Ihre Tabelle partitioniert ist, gibt es mehrere Dateien, die mit dem Partitionsindex `00000` beginnen.

  Es kann Lücken in der Reihenfolge des Partitionsindex geben. Dies liegt daran, dass jede Partition mit einer Bereichsabfrage aus Ihrer Tabelle abgerufen wird. Wenn sich im Bereich dieser Partition keine Daten befinden, wird diese Sequenznummer übersprungen.

  Angenommen, die `id`-Spalte ist der Primärschlüssel der Tabelle und deren Minimal- und Maximalwerte sind `100` und `1000`. Wenn wir versuchen, diese Tabelle mit 9 Partitionen zu exportieren, lesen wir sie mit parallelen Abfragen wie den folgenden:

  ```
  SELECT * FROM table WHERE id <= 100 AND id < 200
   SELECT * FROM table WHERE id <= 200 AND id < 300
  ```

  Dies sollte 9 Dateien von `part-00000-`*`random_uuid`*`.gz.parquet` bis `part-00008-`*`random_uuid`*`.gz.parquet` generieren. Wenn es jedoch keine Zeilen mit IDs zwischen `200` und `350` gibt, ist eine der abgeschlossenen Partitionen leer und es wird keine Datei für diese erstellt. Im vorherigen Beispiel wurde `part-00001-`*`random_uuid`*`.gz.parquet` nicht erstellt.

- Ältere Konvention:

```
part-partition_index-random_uuid.format-based_extension
```

Diese entspricht der aktuellen Konvention, allerdings ohne das *batch_index*-Präfix, zum Beispiel:

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
 part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
 part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

Die Namenskonvention für Dateien kann geändert werden. Daher empfehlen wir beim Lesen von Zieltabellen, dass Sie alles innerhalb des Basispräfixes für die Tabelle lesen.

Datenkonvertierung beim Exportieren in einen Amazon S3-Bucket

Wenn Sie einen DB-Snapshot in einen Amazon-S3-Bucket exportieren, konvertiert Amazon Aurora Daten in das Parquet-Format, exportiert Daten darin und speichert Daten im Parquet-Format. Weitere Informationen über Parquet finden Sie auf der Website Apache Parquet.

Parquet speichert alle Daten als einen der folgenden primitiven Typen:

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – Ein Byte-Array mit variabler Länge, auch bekannt als Binary
- FIXED_LEN_BYTE_ARRAY – Ein Byte-Array fester Länge, das verwendet wird, wenn die Werte eine konstante Größe haben

Es gibt nur wenige Parquet-Datentypen, um die Komplexität beim Lesen und Schreiben des Formats zu reduzieren. Parquet bietet logische Typen zur Erweiterung primitiver Typen. Ein logischer Typ ist als Annotation, bei der Daten in einem `LogicalType`-Metadatenfeld implementiert sind. Die logische Typannotation beschreibt, wie der primitive Typ zu interpretieren ist.

Wenn der logische Typ `STRING` einen `BYTE_ARRAY`-Typ annotiert, gibt er an, dass das Byte-Array als UTF-8-kodierte Zeichenfolge interpretiert werden soll. Nach Abschluss einer Exportaufgabe informiert Sie Amazon Aurora, wenn eine Zeichenfolgenkonvertierung stattgefunden hat. Die zugrunde liegenden exportierten Daten entsprechen immer den Daten aus der Quelle. Aufgrund des Kodierungsunterschieds in UTF-8 können jedoch einige Zeichen beim Einlesen von Tools wie Athena anders als in der Quelle erscheinen.

Weitere Informationen finden Sie unter Logische Typdefinitionen für Parquet in der Parquet-Dokumentation.

Themen

- MySQL-Datentypzuordnung zu Parquet
- PostgreSQL-Datentyp-Mapping zu Parquet

MySQL-Datentypzuordnung zu Parquet

Die folgende Tabelle zeigt die Zuordnung von MySQL-Datentypen zu Parquet-Datentypen, wenn die Daten konvertiert und nach Amazon S3 exportiert werden.

Quelldatentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zur Konvertierung
Numerische Datentypen			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN_BYTE_ARRAY(9)	DECIMAL(20,0)	Parquet unterstützt nur signierte Typen, daher erfordert das Mapping ein zusätzliches Byte (8 plus 1), um den Typ BIGINT_UNSIGNED zu speichern.
BIT	BYTE_ARRAY		

Quelldatentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zur Konvertierung
DECIMAL	INT32	DECIMAL (p,s)	Wenn der Quellwert kleiner als 2^{31} ist, wird er als INT32 gespeichert.
	INT64	DECIMAL (p,s)	Wenn der Quellwert 2^{31} oder größer ist, aber kleiner als 2^{63} ist, wird er als INT64 gespeichert.
	FIX_LEN_BYTE_ARRAY(N)	DECIMAL (p,s)	Wenn der Quellwert 2^{63} oder größer ist, wird er als FIXED_LEN_BYTE_ARRAY(N) gespeichert.
	BYTE_ARRAY	STRING	Parquet unterstützt maximal 38 Dezimalstellen. Der Dezimalwert wird in eine Zeichenfolge vom Typ BYTE_ARRAY konvertiert und als UTF8 kodiert.
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		
INT UNSIGNED	INT64		
MEDIUMINT	INT32		

Quelldatentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zur Konvertierung
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL (p,s)	Wenn der Quellwert kleiner als 2^{31} ist, wird er als INT32 gespeichert.
	INT64	DECIMAL (p,s)	Wenn der Quellwert 2^{31} oder größer ist, aber kleiner als 2^{63} ist, wird er als INT64 gespeichert.
	FIXED_LEN _ARRAY(N)	DECIMAL (p,s)	Wenn der Quellwert 2^{63} oder größer ist, wird er als FIXED_LEN _BYTE_ARRAY(N) gespeichert.
	BYTE_ARRAY	STRING	Parquet unterstützt keine numerische Genauigkeit größer als 38. Der Numeric-Wert wird in eine Zeichenfolge vom Typ BYTE_ARRAY konvertiert und als UTF8 kodiert.
SMALLINT	INT32		

Quelldatentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zur Konvertierung
SMALLINT UNSIGNED	INT32		
TINYINT	INT32		
TINYINT UNSIGNED	INT32	INT (16, true)	
Zeichenfolgen-Datentypen			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	
MEDIUMBLOB	BYTE_ARRAY		
MEDIUMTEXT	BYTE_ARRAY	STRING	
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		

Quelldatentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zur Konvertierung
VARCHAR	BYTE_ARRAY	STRING	
Datums- und Uhrzeit-Datentypen			
DATUM	BYTE_ARRAY	STRING	Ein Datum wird in eine Zeichenfolge vom Typ BYTE_ARRAY konvertiert und als UTF8 kodiert.
DATETIME	INT64	TIMESTAMP_MICROS	
TIME	BYTE_ARRAY	STRING	Ein TIME-Typ wird in einem BYTE_ARRAY in eine Zeichenfolge konvertiert und als UTF8 kodiert.
TIMESTAMP	INT64	TIMESTAMP_MICROS	
YEAR	INT32		
Geometrische Datentypen			
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		

Quelldatentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zur Konvertierung
POLYGON	BYTE_ARRAY		
JSON-Datentyp			
JSON	BYTE_ARRAY	STRING	

PostgreSQL-Datentyp-Mapping zu Parquet

Die folgende Tabelle zeigt das Mapping von PostgreSQL-Datentypen zu Parquet-Datentypen, wenn Daten konvertiert und nach Amazon S3 exportiert werden.

PostgreSQL-Datentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zum Mapping
Numerische Datentypen			
BIGINT	INT64		
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	Ein DECIMAL-Typ wird in eine Zeichenfolge in einem BYTE_ARRAY-Typ konvertiert und als UTF8 kodiert. Diese Konvertierung soll Komplikationen aufgrund von Datengenauigkeit und Datenwerten, die keine Zahlen sind (NaN), vermeiden.

PostgreSQL-Datentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zum Mapping
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT (16, true)	
SMALLSERIAL	INT32	INT (16, true)	
Zeichenfolgen- und verwandte Datentypen			
ARRAY	BYTE_ARRAY	STRING	Ein Array wird in eine Zeichenfolge konvertiert und als BINARY (UTF8) kodiert. Diese Konvertierung dient dazu, Komplikationen aufgrund von Datengenauigkeit, Datenwerten, die keine Zahl sind (NaN), und Zeitdatenwerten zu vermeiden.
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		

PostgreSQL-Datentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zum Mapping
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
Datums- und Uhrzeit-Datentypen			
DATUM	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	
TIME WITH TIME ZONE	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP WITH TIME ZONE	BYTE_ARRAY	STRING	
Geometrische Datentypen			
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	

PostgreSQL-Datentyp	Parquet-Primitiv-Typ	Logische Typannotation	Anmerkungen zum Mapping
LINESEGMENT	BYTE_ARRAY	STRING	
PATH	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	
POLYGON	BYTE_ARRAY	STRING	
JSON-Datentypen			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
Weitere Datentypen			
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	Network-Datentyp
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	Network-Datentyp
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	–		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	
UUID	BYTE_ARRAY	STRING	

Einrichten des Zugriffs auf einen Amazon-S3-Bucket

Sie identifizieren den Amazon-S3-Bucket und gewähren dem Snapshot die Berechtigung, auf den Bucket zuzugreifen.

Themen

- Identifizieren des Amazon-S3-Buckets, in den exportiert werden soll
- Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle
- Einen kontoübergreifenden Amazon-S3-Bucket verwenden
- Ein kontoübergreifendes Konto verwenden AWS KMS key

Identifizieren des Amazon-S3-Buckets, in den exportiert werden soll

Identifizieren Sie den Amazon-S3-Bucket, in den der DB-Snapshot exportiert werden soll. Verwenden Sie einen vorhandenen S3-Bucket oder erstellen Sie einen neuen S3-Bucket.

> **Note**
>
> Der S3-Bucket, in den exportiert werden soll, muss sich in derselben AWS Region wie der Snapshot befinden.

Weitere Informationen zur Arbeit mit Amazon S3-Buckets finden Sie im Amazon Simple Storage Service User Guide:

- Wie zeige ich die Eigenschaften für einen S3-Bucket an?
- Wie aktiviere ich die Standardverschlüsselung für einen Amazon S3-Bucket?
- Wie erstelle ich einen S3-Bucket?

Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle

Bevor Sie DB-Snapshot-Daten nach Amazon S3 exportieren, geben Sie den Snapshot-Exportaufgaben Schreibzugriffsrechte auf den Amazon S3-Bucket.

Zum Gewähren dieser Berechtigung erstellen Sie erstellen Sie eine IAM-Richtlinie, die Zugriff auf den Bucket ermöglicht. Erstellen Sie dann eine IAM-Rolle und fügen Sie der Rolle die Richtlinie an. Die IAM-Rolle können Sie später Ihrer Snapshot-Exportaufgabe zuweisen.

> **Important**
>
> Wenn Sie planen, den zum Exportieren Ihres Snapshots AWS-Managementkonsole zu verwenden, können Sie festlegen, dass die IAM-Richtlinie und die Rolle beim Exportieren des Snapshots automatisch erstellt werden. Detaillierte Anweisungen finden Sie unter Erstellen von Snapshot-Exportaufgaben.

So geben Sie DB-Snapshot-Aufgaben Zugriff auf Amazon S3

1. Erstellen Sie eine IAM-Richtlinie. Diese Richtlinie bietet die Berechtigungen für den Bucket und die Objekte, die den Zugriff auf Ihre Snapshot-Exportaufgabe ermöglichen Amazon S3.

   Die Richtlinie muss die folgenden erforderlichen Aktionen enthalten, um die Übertragung von Dateien aus Amazon Aurora in einen S3-Bucket zu ermöglichen:

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

   Fügen Sie in die Richtlinie die folgenden Ressourcen zur Identifizierung des S3-Buckets und der Objekte im Bucket ein. Die folgende Liste von Ressourcen zeigt das ARN-Format (Amazon-Ressourcenname) für den Amazon S3-Zugriff.

   - `arn:aws:s3:::`*`amzn-s3-demo-bucket`*
   - `arn:aws:s3:::`*`amzn-s3-demo-bucket`*`/*`

   Weitere Informationen zur Erstellung einer IAM-Richtlinie für Amazon Aurora finden Sie unter Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff. Siehe auch Tutorial: Erstellen und Anfügen Ihrer ersten vom Kunden verwalteten Richtlinie im IAM-Benutzerhandbuch.

   Mit dem folgenden AWS CLI Befehl wird eine IAM-Richtlinie `ExportPolicy` mit diesen Optionen erstellt. Sie gewährt Zugriff auf einen Bucket namens *`amzn-s3-demo-bucket`*.

> **Note**
>
> Nachdem Sie die Richtlinie erstellt haben, notieren Sie den ARN der Richtlinie. Sie benötigen den ARN für einen nachfolgenden Schritt, in dem Sie die Richtlinie an eine IAM-Rolle anhängen.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. Erstellen Sie eine IAM-Rolle, damit Aurora diese IAM-Rolle in Ihrem Namen übernehmen kann, um auf Ihre Amazon-S3-Buckets zuzugreifen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen IAM-Benutzer im IAM-Benutzerhandbuch.

   Das folgende Beispiel zeigt, wie der AWS CLI Befehl verwendet wird, um eine Rolle mit dem Namen `rds-s3-export-role` zu erstellen.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
     "Version": "2012-10-17",
     "Statement": [
```

```
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. Fügen Sie die erstellte IAM-Richtlinie der IAM-Rolle an, die Sie erstellt haben.

   Mit dem folgenden AWS CLI Befehl wird die zuvor erstellte Richtlinie an die angegebene `rds-s3-export-role` Rolle angehängt. Ersetzen Sie *`your-policy-arn`* durch den Richtlinien-ARN, den Sie in einem früheren Schritt notiert haben.

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-export-role
```

Einen kontoübergreifenden Amazon-S3-Bucket verwenden

Sie können Amazon S3 S3-Buckets AWS kontenübergreifend verwenden. Um einen kontoübergreifenden Bucket zu verwenden, fügen Sie eine Bucket-Richtlinie hinzu, um den Zugriff auf die IAM-Rolle zu erlauben, die Sie für die S3-Exporte verwenden. Weitere Informationen finden Sie unter Beispiel 2: Bucket-Eigentümer erteilt kontoübergreifende Bucket-Berechtigungen.

- Fügen Sie eine Bucket-Richtlinie an Ihren Bucket an, wie im folgenden Beispiel gezeigt.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Admin"
            },
            "Action": [
                "s3:PutObject*",
```

```
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket",
                "arn:aws:s3:::amzn-s3-demo-destination-bucket/*"
            ]
        }
    ]
}
```

Ein kontoübergreifendes Konto verwenden AWS KMS key

Sie können ein Cross-Konto verwenden AWS KMS key , um Amazon S3 S3-Exporte zu verschlüsseln. Zuerst fügen Sie dem lokalen Konto eine Schlüsselrichtlinie hinzu und fügen dann IAM-Richtlinien im externen Konto hinzu. Weitere Informationen finden Sie unter Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben.

So verwenden Sie einen kontoübergreifenden KMS-Schlüssel

1. Fügen Sie dem lokalen Konto eine Schlüsselrichtlinie hinzu.

   Das folgende Beispiel gibt `ExampleRole` und `ExampleUser` im externen Konto 444455556666 Berechtigungen im lokalen Konto 123456789012.

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
```

```
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "*"
}
```

2. Fügen Sie IAM-Richtlinien im externen Konto hinzu.

   Die folgende IAM-Richtlinie erlaubt es dem Prinzipal, den KMS-Schlüssel im Konto 123456789012 für kryptografische Operationen zu erlauben. Um diese Berechtigung an `ExampleRole` und `ExampleUser` zu erteilen, fügen Sie die Richtlinie zu ihnen im Konto 444455556666 an.

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

Erstellen von Snapshot-Exportaufgaben

Erstellen Sie Snapshot-Exportaufgaben, um Daten von Ihrem Snapshot in einen Amazon-S3-Bucket zu exportieren. Sie können bis zu fünf gleichzeitige DB-Snapshot-Exportaufgaben pro AWS-Konto durchführen.

> **Note**
>
> Das Exportieren von RDS-Snapshots kann je nach Datenbanktyp und -größe eine Weile dauern. Die Exportaufgabe stellt zuerst die gesamte Datenbank wieder her und skaliert sie,

bevor die Daten in Amazon S3 extrahiert werden. Der Fortschritt des Vorgangs während dieser Phase wird als Starting (Startet) angezeigt. Wenn die Aufgabe zum Exportieren von Daten zu S3 wechselt, wird der Fortschritt als In progress (In Bearbeitung) angezeigt.
Die Zeit, die für den Export benötigt wird, hängt von den in der Datenbank gespeicherten Daten ab. Beispielsweise exportieren Tabellen mit gut verteilten numerischen Primärschlüssel- oder Indexspalten am schnellsten. Bei Tabellen, die keine Spalte enthalten, die für die Partitionierung geeignet ist, und bei Tabellen mit nur einem Index für eine auf Zeichenfolgen basierende Spalte, dauert dies länger, da der Export einen langsameren Single-Thread-Prozess verwendet. Diese längere Exportzeit tritt auf, da der Export einen langsameren Single-Thread-Prozess verwendet.

Sie können einen DB-Snapshot mit Hilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API zu Amazon S3 exportieren.

Wenn Sie eine Lambda-Funktion zum Exportieren eines Snapshots verwenden, fügen Sie die Aktion `kms:DescribeKey` der Lambda-Funktionsrichtlinie hinzu. Weitere Informationen finden Sie unter AWS Lambda Berechtigungen.

Konsole

Die Konsolenoption Export to Amazon S3 (Zu Amazon S3-Konsole exportieren) wird nur für Snapshots angezeigt, die zu Amazon S3 exportiert werden können. Ein Snapshot ist aus folgenden Gründen möglicherweise nicht für den Export verfügbar:

- Die DB-Engine wird für den S3-Export nicht unterstützt.
- Die DB-Instance-Version wird für den S3-Export nicht unterstützt.
- Der S3-Export wird in der AWS-Region, in der der Snapshot erstellt wurde, nicht unterstützt.

So exportieren Sie einen DB-Snapshot:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.
3. Wählen Sie auf den Registerkarten die Art des Snapshots aus, den Sie exportieren möchten.
4. Wählen Sie in der Liste der Snapshots den Snapshot aus, den Sie exportieren möchten.

5. Wählen Sie für Actions (Aktionen) die Option Export to Amazon S3 (Nach Amazon S3 exportieren) aus.

   Das Fenster Export to Amazon S3 (Nach Amazon S3 exportieren) erscheint.
6. Geben Sie für Export identifier (Export-ID) einen Namen ein, um die Exportaufgabe zu identifizieren. Dieser Wert wird auch für den Namen der im S3-Bucket erstellten Datei verwendet.
7. Wählen Sie die zu exportierenden Daten aus:

   - Wählen Sie All (Alle), um alle Daten im Snapshot zu exportieren.
   - Wählen Sie Partial (Teilweise), um bestimmte Teile des Snapshots zu exportieren. Um zu ermitteln, welche Teile des Snapshots exportiert werden sollen, geben Sie eine oder mehrere Datenbanken, Schemas oder Tabellen für Bezeichner ein, getrennt durch Leerzeichen.

     Verwenden Sie das folgende Format:

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]
     [.tablen]
     ```

     Zum Beispiel:

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```
8. Wählen Sie für S3 bucket (S3-Bucket) den Bucket aus, in den exportiert werden soll.

   Um die exportierten Daten einem Ordnerpfad im S3-Bucket zuzuordnen, geben Sie den optionalen Pfad für S3 prefix (S3-Präfix) ein.
9. Wählen Sie für die IAM-Rolle entweder eine Rolle, die Ihnen Schreibzugriff auf den gewählten S3-Bucket gewährt, oder erstellen Sie eine neue Rolle.

   - Wenn Sie eine Rolle durch Befolgen der Schritte in Bereitstellen des Zugriffs auf einen Amazon S3-Bucket mit einer IAM-Rolle erstellt haben, wählen Sie diese Rolle.
   - Wenn Sie keine Rolle erstellt haben, die Ihnen Schreibzugriff auf den von Ihnen gewählten S3-Bucket gewährt, wählen Sie Create a new role (Neue Rolle erstellen) aus, um die Rolle automatisch zu erstellen. Geben Sie dann unter IAM role name (IAM-Rollenname) einen Namen für die Rolle ein.
10. Für AWS KMS key geben Sie den ARN für den Schlüssel ein, der für die Verschlüsselung der exportierten Daten verwendet werden soll.

11. Wählen Sie Export to Amazon S3 (Nach Amazon S3 exportieren) aus.

AWS CLI

Um einen DB–Snapshot mit AWS CLI nach Amazon S3 zu exportieren, verwenden Sie den Befehl start-export-task mit den folgenden erforderlichen Optionen:

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

In den folgenden Beispielen heißt die Snapshot-Exportaufgabe *my-snapshot-export*. Sie exportiert Snapshot in einen S3-Bucket namens *amzn-s3-demo-destination-bucket*.

Example

Für Linux, macOS oder Unix:

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
    --s3-bucket-name amzn-s3-demo-destination-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Für Windows:

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name amzn-s3-demo-destination-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

Beispiel für eine Ausgabe folgt.

```
{
```

```
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-destination-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

Um einen Ordnerpfad im S3-Bucket für den Snapshot-Export bereitzustellen, fügen Sie die Option `--s3-prefix` im Befehl start-export-task hinzu.

RDS-API

Um einen DB-Snapshot mit der Amazon-RDS-API nach Amazon S3 zu exportieren, verwenden Sie die Operation StartExportTask mit den folgenden erforderlichen Parametern:

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

Überwachung von Snapshot-Exporten

Sie können DB-Snapshot-Exporte mit der AWS-Managementkonsole, der AWS CLI oder der RDS-API überwachen.

Konsole

So überwachen Sie DB-Snapshot-Exporte:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Exports in Amazon S3 (Exporte in Amazon S3) aus.

DB-Snapshot-Exporte werden in der Spalte Source type (Quelltyp) angegeben. Der Exportstatus wird in der Spalte Status angezeigt.

3. Um ausführliche Informationen über einen bestimmten Snapshot-Export anzuzeigen, wählen Sie die Exportaufgabe aus.

AWS CLI

Um DB-Snapshot-Exporte mit der AWS CLI zu überwachen, verwenden Sie den Befehl describe-export-tasks.

Das folgende Beispiel zeigt, wie Sie aktuelle Informationen über alle Ihre Snapshot-Exporte anzeigen können.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
```

```
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket my-exports isn't located in the current AWS
 Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
}
```

Um Informationen über einen bestimmten Snapshot-Export anzuzeigen, fügen Sie die Option `--export-task-identifier` mit dem Befehl `describe-export-tasks` ein. Um die Ausgabe zu filtern, fügen Sie die Option `--Filters` ein. Weitere Optionen finden Sie beim describe-export-tasks-Befehl.

RDS-API

Um Informationen über DB-Snapshot-Exporte mit der Amazon-RDS-API anzuzeigen, verwenden Sie die DescribeExportTasks-Operation.

Um den Abschluss des Exportworkflows zu verfolgen oder einen anderen Workflow zu initiieren, können Sie Themen von Amazon Simple Notification Service abonnieren. Weitere Informationen zu Amazon SNS finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

Abbrechen einer Snapshot-Exportaufgabe

Sie können eine DB-Snapshot-Exportaufgabe mit Hilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API abbrechen.

> **Note**
>
> Das Abbrechen einer Snapshot-Exportaufgabe entfernt keine Daten, die nach Amazon S3 exportiert wurden. Informationen zum Löschen der Daten über die Konsole finden Sie unter Wie lösche ich Objekte aus einem S3-Bucket?. Um die Daten mit der CLI zu löschen, verwenden Sie den Befehl delete-object.

Konsole

So brechen Sie eine Aufgabe zum Export von Snapshots ab:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Exports in Amazon S3 (Exporte in Amazon S3) aus.

   DB-Snapshot-Exporte werden in der Spalte Source type (Quelltyp) angegeben. Der Exportstatus wird in der Spalte Status angezeigt.
3. Wählen Sie die Snapshot-Exportaufgabe aus, die Sie abbrechen möchten.
4. Klicken Sie auf Abbrechen.
5. Wählen Sie die Bestätigungsseite Cancel export task (Exportaufgabe abbrechen) aus.

AWS CLI

Um eine Snapshot-Exportaufgabe mit der AWS CLI abzubrechen, verwenden Sie den Befehl cancel-export-task aus. Der Befehl erfordert die Option `--export-task-identifier`.

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS-API

Um einen Snapshot-Exportauftrag mit der Amazon-RDS-API abzubrechen, verwenden Sie den Befehl CancelExportTask mit dem Parameter `ExportTaskIdentifier`.

Exportleistung in Aurora MySQL

Die DB-Cluster-Snapshots der Versionen 2 und 3 von Aurora MySQL verwenden einen fortschrittlichen Exportmechanismus, um die Leistung zu verbessern und die Exportzeit zu verkürzen. Der Mechanismus umfasst Optimierungen wie mehrere Export-Threads und Parallelabfragen von Aurora MySQL, um die Vorteile der gemeinsamen Speicherarchitektur von Aurora zu nutzen. Die Optimierungen werden adaptiv angewendet, abhängig von der Größe und Struktur des Datensatzes.

Sie müssen die Parallelabfragen nicht aktivieren, um den schnelleren Exportvorgang zu verwenden, aber der Prozess hat dieselben Einschränkungen wie die Parallelabfragen. Darüber hinaus werden einige Datenwerte nicht unterstützt, z. B. Daten, an denen der Tag des Monats `0` oder das Jahr `0000` lautet. Weitere Informationen finden Sie unter Parallel Query für Amazon Aurora MySQL.

Wenn Leistungsoptimierungen angewendet werden, sehen Sie möglicherweise auch viel größere (~200 GB) Parquet-Dateien für Exporte von Aurora MySQL Version 2 und 3.

Wenn der schnellere Exportvorgang nicht verwendet werden kann, beispielsweise aufgrund inkompatibler Datentypen oder Werte, wechselt Aurora automatisch in einen Single-Thread-

Exportmodus ohne Parallelabfrage. Je nachdem, welcher Prozess verwendet wird und wie viel Daten exportiert werden sollen, kann die Exportleistung variieren.

Fehlerbehebung bei Snapshot-Exporten

Nutzen Sie die folgenden Abschnitte, um Fehlermeldungen und PostgreSQL-Berechtigungsfehler bei DB-Cluster-Exporten nach Amazon S3 zu beheben.

Fehlermeldungen für Amazon-S3-Exportaufgaben

In der folgenden Tabelle werden die Nachrichten beschrieben, die zurückgegeben werden, wenn Amazon-S3-Exportaufgaben fehlschlagen.

Fehlernachricht	Beschreibung
Ein unbekannter interner Fehler ist aufgetreten.	Die Aufgabe ist fehlgeschlagen, da eine unbekannte Störung, eine Ausnahme oder ein Fehler aufgetreten ist.
Beim Schreiben der Metadaten der Exportaufgabe in den S3-Bucket [Bucket-Name] ist ein unbekannter interner Fehler aufgetreten.	Die Aufgabe ist fehlgeschlagen, da eine unbekannte Störung, eine Ausnahme oder ein Fehler aufgetreten ist.
Der RDS-Export konnte die Metadaten der Exportaufgabe nicht schreiben, da er die IAM-Rolle [Rolle ARN] nicht übernehmen kann.	Die Exportaufgabe geht davon aus, dass Ihre IAM-Rolle überprüft, ob es erlaubt ist Metadaten in Ihren S3-Bucket zu schreiben Wenn die Aufgabe Ihre IAM-Rolle nicht übernehmen kann, schlägt sie fehl.
Der RDS-Export konnte die Metadaten der Exportaufgabe nicht in den S3-Bucket [Bucket-Name] mit der IAM-Rolle [Rolle ARN] mit dem KMS-Schlüssel [Schlüssel-ID] schreiben. Fehlercode: [Fehlercode]	Eine oder mehrere Berechtigungen fehlen, sodass die Exportaufgabe nicht auf den S3-Bucket zugreifen kann. Diese Fehlermeldung wird ausgelöst, wenn einer der folgenden Fehlercodes empfangen wird: • `AWSSecurityTokenServiceException` mit dem Fehlercode `AccessDenied` • `AmazonS3Exception` mit dem Fehlercode `NoSuchBucket` , `AccessDenied` , `KMS.KMSIn`

Fehlernachricht	Beschreibung
	`validStateException` , `403 Forbidden` , oder `KMS.DisabledException` Diese Fehlercodes weisen darauf hin, dass für die IAM-Rolle, den S3-Bucket oder den KMS-Schlüssel Einstellungen falsch konfiguriert sind.
Die IAM-Rolle [Rolle ARN] ist nicht berechtigt, [S3-Aktion] im S3-Bucket [Bucket-Name] aufzurufen. Überprüfen Sie Ihre Berechtigungen und versuchen Sie den Export erneut.	Die IAM-Richtlinie ist falsch konfiguriert. Die Berechtigung für die spezifische S3-Aktion für den S3-Bucket fehlt, was zu einem Fehler der Exportaufgabe führt.
Fehler bei der Überprüfung des KMS-Schlüssels. Überprüfen Sie die Anmeldeinformation Ihres KMS-Schlüssels und versuchen Sie es erneut.	Die Überprüfung der KMS-Schlüsselanmeldeinformationen ist fehlgeschlagen.
Die Überprüfung der S3-Anmeldeinformation ist fehlgeschlagen. Überprüfen Sie die Berechtigungen für Ihren S3-Bucket und Ihre IAM-Richtlinie.	Die Überprüfung der S3-Anmeldeinformation ist fehlgeschlagen.
Der S3-Bucket [Bucket-Name] ist ungültig. Entweder befindet sie sich nicht in der aktuellen AWS-Region oder es gibt sie nicht. Überprüfen Sie Ihren S3-Bucket-Namen und versuchen Sie den Export erneut.	Der S3-Bucket ist ungültig.
Der S3-Bucket [Bucket-Name] befindet sich nicht in der aktuellen AWS-Region. Überprüfen Sie Ihren S3-Bucket-Namen und versuchen Sie den Export erneut.	Der S3-Bucket liegt in der falschen AWS-Region.

Fehlerbehebung bei PostgreSQL-Berechtigungsfehlern

Beim Exportieren von PostgreSQL-Datenbanken in Amazon S3 wird möglicherweise ein `PERMISSIONS_DO_NOT_EXIST`-Fehler angezeigt, der besagt, dass bestimmte Tabellen übersprungen wurden. Dieser Fehler tritt normalerweise auf, wenn der Superuser, den Sie beim Erstellen der DB-Instance angegeben haben, keine Berechtigungen für den Zugriff auf diese Tabellen besitzt.

Führen Sie den folgenden Befehl aus, um diesen Fehler zu beheben:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

Weitere Informationen zu Superuser-Berechtigungen finden Sie unter Berechtigungen von Hauptbenutzerkonten.

Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit

Sie können einen DB-Cluster auf einen bestimmten Zeitpunkt wiederherstellen, wodurch ein neuer DB-Cluster erstellt wird.

Wenn Sie einen DB-Cluster zu einem bestimmten Zeitpunkt wiederherstellen, können Sie die Standard-VPC-Sicherheitsgruppe (Virtual Private Cloud) auswählen. Oder Sie können eine benutzerdefinierte VPC-Sicherheitsgruppe auf Ihren DB-Cluster anwenden.

Wiederhergestellte DB-Cluster werden automatisch dem Standard-DB-Cluster und DB-Parametergruppen zugeordnet. Sie können jedoch eine benutzerdefinierte Parametergruppe anwenden, indem Sie diese während einer Wiederherstellung angeben.

Amazon Aurora lädt Protokolldatensätze für DB-Cluster laufend nach Amazon S3 hoch. Um den letzten wiederherstellbaren Zeitpunkt für einen DB-Cluster zu ermitteln, verwenden Sie den AWS CLI describe-db-clustersBefehl und sehen Sie sich den Wert an, der im `LatestRestorableTime` Feld für den DB-Cluster zurückgegeben wurde.

Sie können die Backup auf jeden beliebigen Zeitpunkt innerhalb des Aufbewahrungszeitraums für Backups vornehmen. Um den frühesten wiederherstellbaren Zeitpunkt für einen DB-Cluster zu ermitteln, verwenden Sie den AWS CLI describe-db-clustersBefehl und sehen Sie sich den Wert an, der im `EarliestRestorableTime` Feld für den DB-Cluster zurückgegeben wird.

Der Backup-Aufbewahrungszeitraum des wiederhergestellten DB-Clusters entspricht dem des Quell-DB-Clusters.

> **Note**
>
> Informationen in diesem Thema gelten für Amazon Aurora. Weitere Informationen zum Wiederherstellen einer Amazon-RDS-DB-Instance finden Sie unter Wiederherstellen einer DB-Instance zu einer bestimmten Zeit.
> Weitere Informationen zum Sichern und Wiederherstellen eines Aurora-DB-Clusters finden Sie unter Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters.
> Für Aurora MySQL können Sie einen bereitgestellten DB-Cluster als Aurora Serverless-DB-Cluster wiederherstellen. Weitere Informationen finden Sie unter Wiederherstellen eines Aurora Serverless v1-DB-Clusters.
> Sie können es auch verwenden AWS Backup , um Backups von Amazon Aurora Aurora-DB-Clustern zu verwalten. Wenn Ihr DB-Cluster mit einem Backup-Plan verknüpft ist AWS Backup, wird dieser Backup-Plan für die point-in-time Wiederherstellung verwendet. Weitere

Informationen finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit mit AWS Backup.

Informationen zur Wiederherstellung eines Aurora-DB-Clusters oder eines globalen Clusters mit einer RDS-Version mit erweitertem Support finden Sie unter Wiederherstellen eines Aurora-DB-Clusters oder eines globalen Clusters mit Amazon RDS Extended Support.

Stellen Sie einen DB-Cluster anhand eines automatisierten Backups, eines gespeicherten automatisierten Backups oder mithilfe von zu einem bestimmten Zeitpunkt wieder her AWS Backup.

Themen

- Wiederherstellen eines DB-Clusters zu einem bestimmten Zeitpunkt
- Wiederherstellen eines DB-Clusters zu einem bestimmten Zeitpunkt aus einem beibehaltenen, automatisierten Backup
- Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit mit AWS Backup

Wiederherstellen eines DB-Clusters zu einem bestimmten Zeitpunkt

Sie können einen DB-Cluster mithilfe der AWS-Managementkonsole, der oder der RDS-API auf einen bestimmten Zeitpunkt wiederherstellen. AWS CLI

Konsole

Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Automated backups (Automatisierte Backups) aus.

   Die automatisierten Backups werden auf der Registerkarte Current Region (Aktuelle Region) angezeigt.

3. Wählen Sie den DB-Cluster aus, den Sie wiederherstellen möchten.
4. Wählen Sie unter Aktionen die Option Wiederherstellen auf einen bestimmten Zeitpunkt aus.

   Anschließend wird das Fenster Restore to point in time (Zu einem bestimmten Zeitpunkt wiederherstellen) angezeigt.
5. Wählen Sie Späteste Wiederherstellungszeit, um auf den spätesten möglichen Zeitpunkt wiederherzustellen oder wählen Sie Benutzerdefiniert, um eine Zeit auszuwählen.

   Wenn Sie Benutzerdefiniert wählen, geben Sie das Datum und die Uhrzeit ein, auf die Sie den Cluster wiederherstellen möchten.

   > **Note**
   >
   > Zeiten werden in Ihrer lokalen Zeitzone angezeigt, die durch einen Offset von Coordinated Universal Time (UTC) angezeigt wird. UTC-5 ist beispielsweise Eastern Standard Time/Central Daylight Time.
6. Geben Sie für DB-Cluster-Kennung den Namen des wiederhergestellten DB-Ziel-Clusters ein. Der Name muss eindeutig sein.
7. Wählen Sie bei Bedarf andere Optionen aus, z. B. die DB-Instance-Klasse und die DB-Cluster-Speicherkonfiguration.

   Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.
8. Wählen Sie Wiederherstellen auf einen bestimmten Zeitpunkt aus.

AWS CLI

Um einen DB-Cluster zu einem bestimmten Zeitpunkt wiederherzustellen, verwenden Sie den AWS CLI Befehl restore-db-cluster-to-, point-in-time um einen neuen DB-Cluster zu erstellen.

Sie können andere Einstellungen festlegen. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.

Wenn in der Anfrage Tags angegeben werden, werden die bereitgestellten Tags auf den wiederhergestellten DB-Cluster angewendet. Wenn in der Anfrage keine Tags angegeben werden und der Quell-DB-Cluster in der Region aktiv ist und über Tags verfügt, fügt Aurora die neuesten Tags aus dem Quell-DB-Cluster zum wiederhergestellten DB-Cluster hinzu.

Example

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-to-point-in-time \
       --source-db-cluster-identifier mysourcedbcluster \
       --db-cluster-identifier mytargetdbcluster \
       --restore-to-time 2017-10-14T23:45:00.000Z
```

Für Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
       --source-db-cluster-identifier mysourcedbcluster ^
       --db-cluster-identifier mytargetdbcluster ^
       --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

Wenn Sie die Konsole für das Wiederherstellen eines DB-Clusters zu einer bestimmen Zeit verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihren DB-Cluster. Wenn Sie den verwenden AWS CLI , um einen DB-Cluster zu einem bestimmten Zeitpunkt wiederherzustellen, müssen Sie die primäre Instance für Ihren DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance.
Rufen Sie den create-db-instance AWS CLI Befehl auf, um die primäre Instance für Ihren DB-Cluster zu erstellen. Fügen Sie den Namen des DB-Clusters als `--db-cluster-identifier`-Optionswert ein.

RDS-API

Zum Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit rufen Sie den Amazon-RDS-API `RestoreDBClusterToPointInTime`-Betrieb mit den folgenden Parametern auf:

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

> **Important**
>
> Wenn Sie die Konsole für das Wiederherstellen eines DB-Clusters zu einer bestimmen Zeit verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihren DB-Cluster. Wenn Sie RDS API für das Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit verwenden, müssen Sie die primäre Instance für Ihren DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance.
> Um die primäre Instance für Ihren DB-Cluster zu erstellen, rufen Sie den RDS-API-Vorgang Create aufDBInstance. Beziehen Sie den Namen des DB-Clusters als `DBClusterIdentifier`-Parameterwert mit ein.

Wiederherstellen eines DB-Clusters zu einem bestimmten Zeitpunkt aus einem beibehaltenen, automatisierten Backup

Sie können einen DB-Cluster aus einem beibehaltenen, automatisierten Backup wiederherstellen, nachdem Sie den Quell-DB-Cluster gelöscht haben, wenn sich das Backup innerhalb des Aufbewahrungszeitraums des Quell-Clusters befindet. Der Vorgang ähnelt dem Wiederherstellen eines DB-Clusters aus einem automatisierten Backup.

> **Note**
>
> Mit diesem Verfahren können Sie einen Aurora Serverless v1-DB-Cluster nicht wiederherstellen, da automatische Backups für Aurora Serverless v1-Cluster nicht beibehalten werden.

Konsole

Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Automated backups (Automatisierte Backups) aus.
3. Wählen Sie die Registerkarte Beibehalten aus.

4. Wählen Sie den -DB-Cluster aus, den Sie wiederherstellen möchten.
5. Wählen Sie unter Aktionen die Option Restore to point in time (Zu einem bestimmten Zeitpunkt wiederherstellen) aus.

   Anschließend wird das Fenster Restore to point in time (Zu einem bestimmten Zeitpunkt wiederherstellen) angezeigt.
6. Wählen Sie Späteste Wiederherstellungszeit, um auf den spätesten möglichen Zeitpunkt wiederherzustellen oder wählen Sie Benutzerdefiniert, um eine Zeit auszuwählen.

   Wenn Sie Benutzerdefiniert wählen, geben Sie das Datum und die Uhrzeit ein, auf die Sie den Cluster wiederherstellen möchten.

   > **Note**
   >
   > Zeiten werden in Ihrer lokalen Zeitzone angezeigt, die durch einen Offset von Coordinated Universal Time (UTC) angezeigt wird. Beispiel: UTC-5 ist Ost Standardzeit/ Zentral Sommerzeit.
7. Geben Sie für DB-Cluster-Kennung den Namen des wiederhergestellten DB-Ziel-Clusters ein. Der Name muss eindeutig sein.
8. Wählen Sie bei Bedarf andere Optionen aus, z. B. DB-Instance-Class.

   Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.
9. Wählen Sie Restore to point in time (Zu einem bestimmten Zeitpunkt wiederherstellen) aus.

AWS CLI

Um einen DB-Cluster zu einem angegebenen Zeitpunkt wiederherzustellen, verwenden Sie den AWS CLI-Befehl restore-db-cluster-to-point-in-time, um einen neuen DB-Cluster zu erstellen.

Sie können andere Einstellungen festlegen. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.

Das Tagging von Ressourcen wird für diesen Vorgang unterstützt. Wenn Sie die Option `--tags` verwenden, werden die Tags des Quell-DB-Clusters ignoriert und die bereitgestellten verwendet. Andernfalls werden die neuesten Tags aus dem Quell-Cluster verwendet.

Example

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Für Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

Wenn Sie die Konsole für das Wiederherstellen eines DB-Clusters zu einer bestimmen Zeit verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihren DB-Cluster. Wenn Sie AWS CLI für das Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit verwenden, müssen Sie die primäre Instance für Ihren DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance.
Rufen Sie den AWS CLI-Befehl create-db-instance auf, um die primäre Instance für Ihr DB-Cluster zu erstellen. Fügen Sie den Namen des DB-Clusters als `--db-cluster-identifier`-Optionswert ein.

RDS-API

Zum Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit rufen Sie den Amazon-RDS-API `RestoreDBClusterToPointInTime`-Betrieb mit den folgenden Parametern auf:

- `SourceDbClusterResourceId`
- `DBClusterIdentifier`
- `RestoreToTime`

> **Important**
>
> Wenn Sie die Konsole für das Wiederherstellen eines DB-Clusters zu einer bestimmen Zeit verwenden, erstellt Amazon RDS automatisch die primäre Instance (Writer) für Ihren DB-Cluster. Wenn Sie RDS API für das Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit verwenden, müssen Sie die primäre Instance für Ihren DB-Cluster explizit erstellen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance.
> Rufen Sie die RDS-API-Operation CreateDBInstance auf, um die primäre Instance für Ihren DB-Cluster zu erstellen. Beziehen Sie den Namen des DB-Clusters als `DBClusterIdentifier`-Parameterwert mit ein.

Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit mit AWS Backup

Verwenden Sie AWS Backup, um Ihre automatisierten Backups zu verwalten und sie dann zu einem bestimmten Zeitpunkt wiederherzustellen. Dazu erstellen Sie einen Backup-Plan in AWS Backup und weisen Ihren DB-Cluster als Ressource zu. Anschließend aktivieren Sie in der Backup-Regel kontinuierliche Backups für PITR. Weitere Informationen zu Backup-Plänen und Backup-Regeln finden Sie im AWSBackup-Leitfaden für Entwickler.

Aktivieren kontinuierlicher Backups in AWS Backup

Sie aktivieren kontinuierliche Backups in den Backup-Regeln.

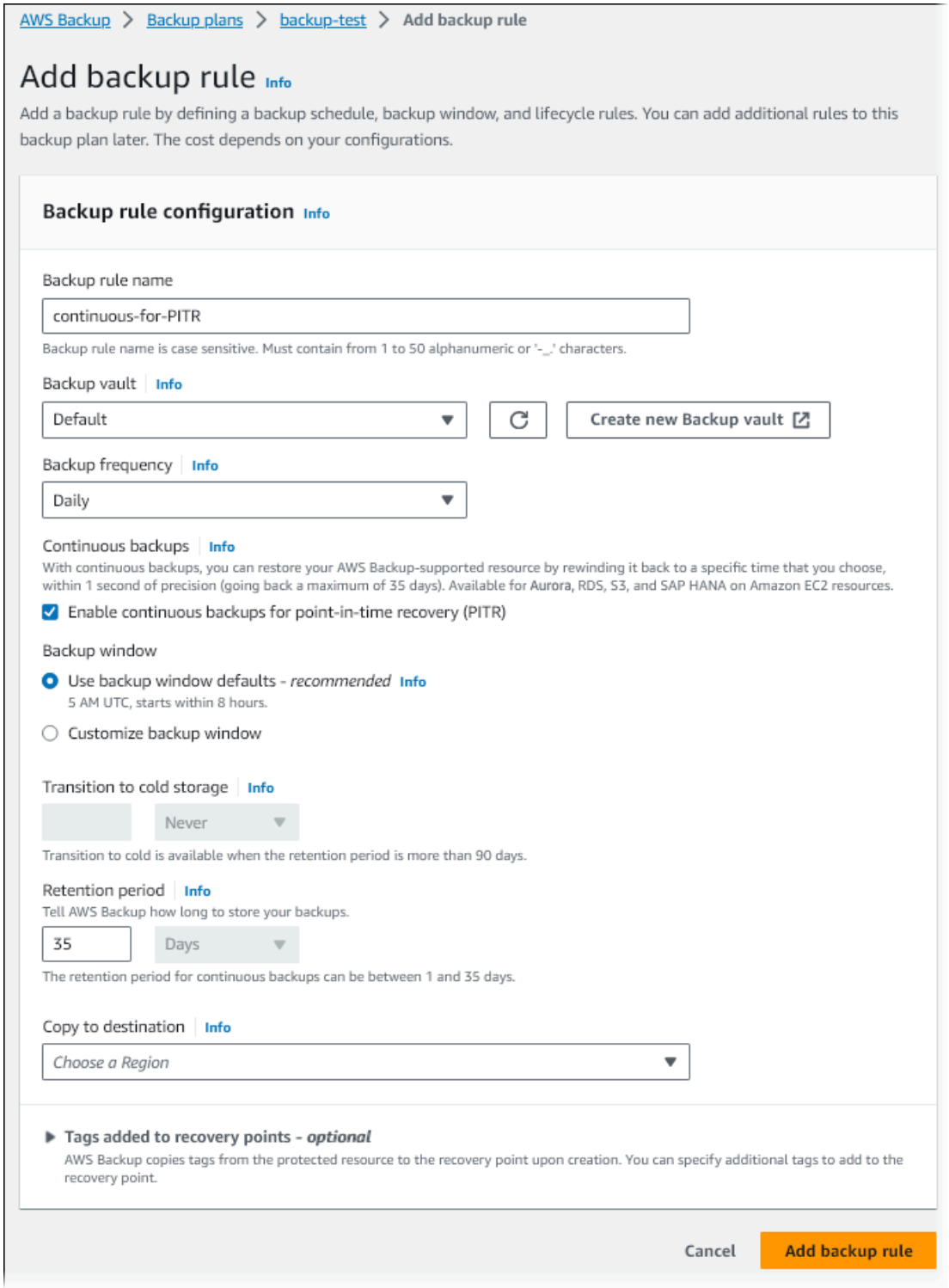
So aktivieren Sie kontinuierliche Backups für PITR

1. Melden Sie sich bei der AWS-Managementkonsole an, und öffnen Sie die AWS Backup-Konsole unter https://console.aws.amazon.com/backup.
2. Wählen Sie im Navigationsbereich Backup-Pläne aus.
3. Unter Name des Backup-Plans wählen Sie den Backup-Plan aus, den Sie für die Sicherung Ihres DB-Clusters verwenden.

4. Wählen Sie im Abschnitt Konfiguration der Sicherungsregel die Option Sicherungsregel hinzufügen aus.

   Die Seite Sicherungsregel hinzufügen wird angezeigt.

5. Markieren Sie das Kontrollkästchen Aktivieren kontinuierlicher Backups für die zeitpunktbezogene Wiederherstellung (PITR).



6. Wählen Sie nach Bedarf andere Einstellungen, und wählen Sie dann Sicherungsregel hinzufügen.

Wiederherstellung aus einem kontinuierlichen Backup in AWS Backup

Verwenden Sie einen Backup-Tresor, um die Wiederherstellung zu einem bestimmten Zeitpunkt zu erstellen.

Konsole

Sie können zum Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit die AWS-Managementkonsole verwenden.

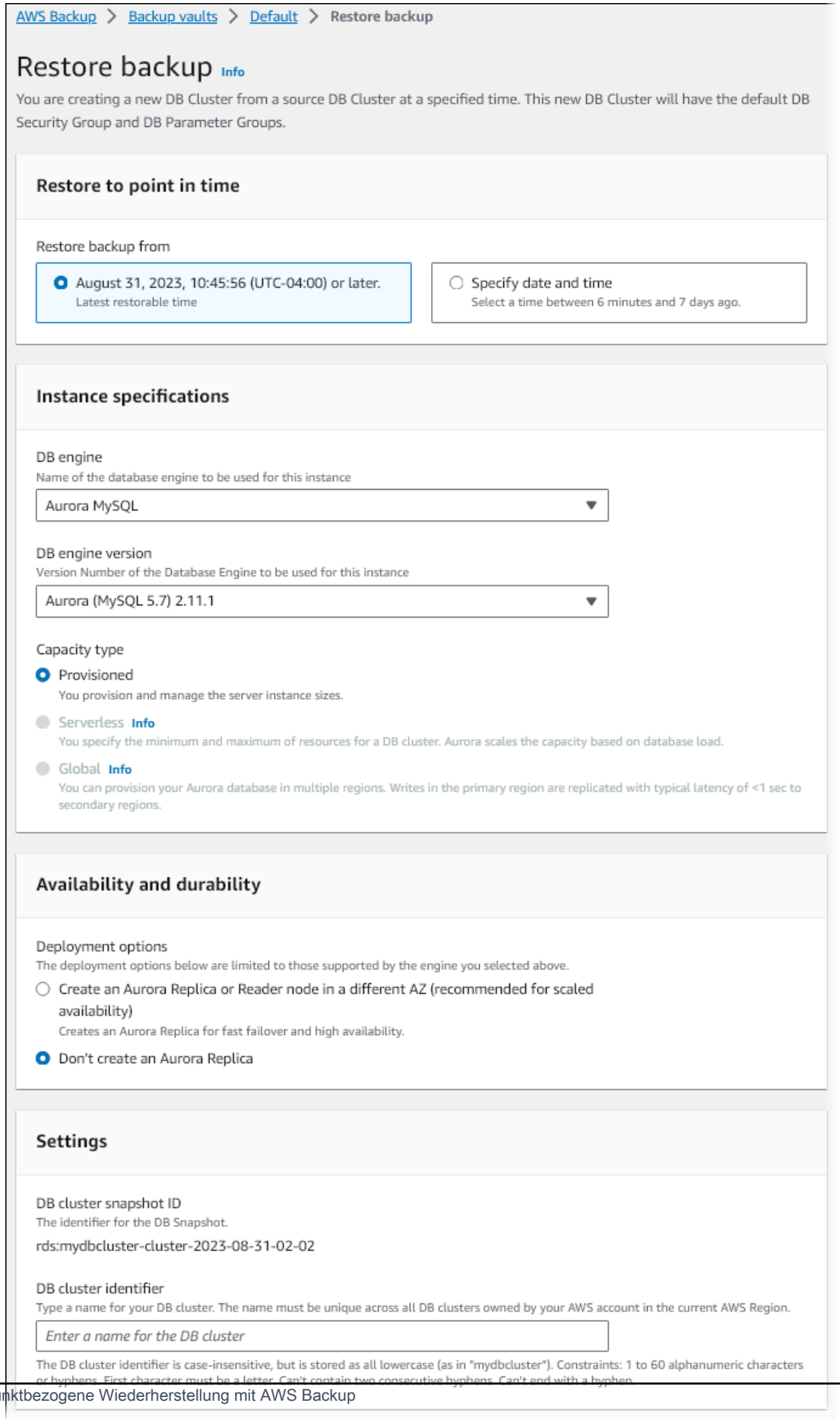
So führen Sie eine Wiederherstellung aus einer kontinuierlichen Sicherung in AWS Backup durch

1. Melden Sie sich bei der AWS-Managementkonsole an, und öffnen Sie die AWS Backup-Konsole unter https://console.aws.amazon.com/backup.
2. Wählen Sie im Navigationsbereich Backup vaults (Sicherungstresore) aus.
3. Wählen Sie den Backup-Tresor aus, der Ihr kontinuierliches Backup enthält, zum Beispiel Standard.

   Die Seite mit den Details zum Backup-Tresor wird angezeigt.
4. Unter Wiederherstellungspunkte wählen Sie den Wiederherstellungspunkt für die automatische Sicherung aus.

   Der Sicherungstyp ist Kontinuierlich und der Name ist `continuous:cluster-`*`AWS-Backup-job-number`*.
5. Wählen Sie unter Aktionen die Option Wiederherstellen aus.

   Die Seite Sicherung wiederherstellen wird angezeigt.


AWS Backup > Backup vaults > Default > Restore backup
Restore backup Info
You are creating a new DB Cluster from a source DB Cluster at a specified time. This new DB Cluster will have the default DB Security Group and DB Parameter Groups.
Restore to point in time
Restore backup from
August 31, 2023, 10:45:56 (UTC-04:00) or later.
Latest restorable time
Specify date and time
Select a time between 6 minutes and 7 days ago.
Instance specifications
DB engine
Name of the database engine to be used for this instance
Aurora MySQL
DB engine version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL 5.7) 2.11.1
Capacity type
Provisioned
You provision and manage the server instance sizes.
Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.
Global Info
You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.
Availability and durability
Deployment options
The deployment options below are limited to those supported by the engine you selected above.
Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
Creates an Aurora Replica for fast failover and high availability.
Don't create an Aurora Replica
Settings
DB cluster snapshot ID
The identifier for the DB Snapshot.
rds:mydbcluster-cluster-2023-08-31-02-02
DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.
Enter a name for the DB cluster
The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

6. Wählen Sie unter Zum gewünschten Zeitpunkt wiederherstellen Datum und Uhrzeit angeben, um zu einem bestimmten Zeitpunkt wiederherzustellen.
7. Wählen Sie je nach Bedarf weitere Einstellungen für die Wiederherstellung des DB-Clusters, und wählen Sie dann Sicherung wiederherstellen.

   Die Seite Jobs wird angezeigt und zeigt das Fenster Wiederherstellungsjobs an. Eine Meldung am Anfang der Seite enthält Informationen zu dem Wiederherstellungsjob.

Nach der Wiederherstellung des DB-Clusters müssen Sie diesem die primäre (Writer)-DB-Instance hinzufügen. Rufen Sie den AWS CLI-Befehl create-db-instance auf, um die primäre Instance für Ihr DB-Cluster zu erstellen. Beziehen Sie den Namen des DB-Clusters als `--db-cluster-identifier`-Parameterwert mit ein.

CLI

Verwenden Sie den AWS CLI-Befehl start-restore-job, um den DB-Cluster zu einem bestimmten Zeitpunkt wiederherzustellen. Die folgenden Parameter sind erforderlich:

- `--recovery-point-arn` – Der Amazon-Ressourcenname (ARN) für den Wiederherstellungspunkt, von dem aus die Wiederherstellung durchgeführt werden soll.
- `--resource-type` – verwenden Sie `Aurora`.
- `--iam-role-arn` – Der ARN für die IAM-Rolle, die Sie für AWS Backup-Operationen verwenden.
- `--metadata` – Die Metadaten, die Sie zur Wiederherstellung des DB-Clusters verwenden. Die folgenden Parameter sind erforderlich:
  - `DBClusterIdentifier`
  - `Engine`
  - `RestoreToTime` oder `UseLatestRestorableTime`

Das folgende Beispiel zeigt, wie ein DB-Cluster zu einem bestimmten Zeitpunkt wiederhergestellt wird.

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
```

```
--metadata '{"DBClusterIdentifier":"backup-pitr-test","Engine":"aurora-
mysql","RestoreToTime":"2023-09-01T17:00:00.000Z"}'
```

Das folgende Beispiel zeigt, wie ein DB-Cluster zum letzten wiederherstellbaren Zeitpunkt wiederhergestellt wird.

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-latest","Engine":"aurora-
mysql","UseLatestRestorableTime":"true"}'
```

Nach der Wiederherstellung des DB-Clusters müssen Sie diesem die primäre (Writer)-DB-Instance hinzufügen. Rufen Sie den AWS CLI-Befehl create-db-instance auf, um die primäre Instance für Ihr DB-Cluster zu erstellen. Beziehen Sie den Namen des DB-Clusters als `--db-cluster-identifier`-Parameterwert mit ein.

Löschen eines DB-Cluster-Snapshots

Sie können DB-Cluster-Snapshots löschen, die mit Amazon RDS verwaltet werden, wenn Sie sie nicht mehr benötigen.

> **Note**
>
> Um mit AWS Backup verwaltete Backups zu löschen, verwenden Sie die AWS Backup-Konsole. Informationen zu AWS Backup finden Sie im AWS Backup-Developer-Handbuch.

Löschen eines DB-Cluster-Snapshots

Sie können einen DB-Cluster-Snapshot mit der Konsole, der AWS CLI oder der RDS-API löschen.

Um einen freigegebenen oder öffentlichen Snapshot zu löschen, müssen Sie sich bei dem AWS-Konto anmelden, in dessen Besitz sich der Snapshot befindet.

Konsole

So löschen Sie einen DB-Cluster-Snapshot

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.
3. Wählen Sie den DB-Cluster-Snapshot aus, den Sie löschen möchten.
4. Wählen Sie unter Actions (Aktionen) die Option Delete Snapshot (Snapshot löschen) aus.
5. Wählen Sie auf der Bestätigungsseite die Option Delete (Löschen) aus.

AWS CLI

Sie können einen DB-Cluster-Snapshot mithilfe des AWS CLI-Befehls delete-db-cluster-snapshot löschen.

Zum Löschen eines DB-Cluster-Snapshots werden die folgenden Optionen verwendet.

- `--db-cluster-snapshot-identifier` – Der Bezeichner des DB-Cluster-Snapshots.

Example

Der folgende Code löscht den DB-Cluster-Snapshot `mydbclustersnapshot`.

Für Linux, macOS oder Unix:

```
aws rds delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Für Windows:

```
aws rds delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS-API

Sie können einen DB-Cluster-Snapshot mit der Amazon-RDS-API-Operation DeleteDBSnapshot löschen.

Zum Löschen eines DB-Cluster-Snapshots werden die folgenden Parameter verwendet.

- `DBClusterSnapshotIdentifier` – Der Bezeichner des DB-Cluster-Snapshots.

Tutorial: Wiederherstellen eines DB-Clusters von Amazon Aurora aus einem DB-Cluster-Snapshot

Bei der Arbeit mit Amazon Aurora ist häufig eine DB-Instance vorhanden, mit der Sie gelegentlich arbeiten, aber die Sie nicht ständig brauchen. Sie können beispielsweise einen DB-Cluster verwenden, um die Daten für einen Bericht zu speichern, den Sie nur vierteljährlich erstellen. Eine Möglichkeit, in einem solchen Szenario Geld zu sparen, besteht darin, nach Abschluss des Berichts einen DB-Cluster-Snapshot des DB-Clusters zu verwenden. Anschließend löschen Sie den DB-Cluster und stellen ihn wieder her, wenn Sie neue Daten hochladen und den Bericht im nächsten Quartal erstellen müssen.

Wenn Sie einen DB-Cluster wiederherstellen, können Sie den Namen des DB-Cluster-Snapshots für die Wiederherstellung angeben. Anschließend vergeben Sie einen Namen für den neuen DB-Cluster, der durch den Wiederherstellungsvorgang erstellt wird. Weitere Informationen zum Wiederherstellen von DB-Cluster aus Snapshots finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.

In diesem Tutorial führen wir ein Upgrade des wiederhergestellten DB-Clusters von Aurora MySQL Version 2 (mit MySQL 5.7 kompatibel) auf Aurora MySQL Version 3 (mit MySQL 8.0 kompatibel) durch.

Stellen Sie einen DB-Clusters zu einer bestimmten Zeit aus einem DB-Cluster-Snapshot über die Amazon-RDS-Konsole oder die AWS CLI wieder her.

Weitere Informationen zur AWS KMS-Schlüsselverwaltung für Amazon RDS finden Sie unter AWS KMS key-Verwaltung.

Themen

- Tutorial: Wiederherstellen eines DB-Clusters aus einem DB-Cluster-Snapshot über die Amazon-RDS-Konsole
- Tutorial: Wiederherstellen eines DB-Clusters aus einem DB-Cluster-Snapshot mithilfe der AWS CLI

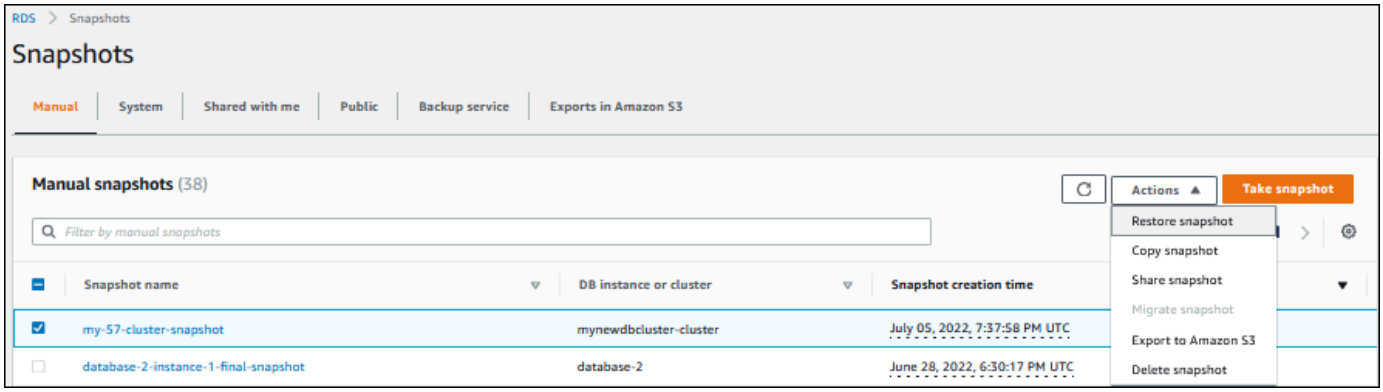
Tutorial: Wiederherstellen eines DB-Clusters aus einem DB-Cluster-Snapshot über die Amazon-RDS-Konsole

In diesem Tutorial stellen Sie einen DB-Cluster aus einem DB-Cluster-Snapshot über die Amazon-RDS-Konsole wieder her. Wenn Sie einen DB-Cluster aus einem Snapshot über die AWS-Managementkonsole wiederherstellen, wird auch die primäre (Writer-)DB-Instance erstellt.

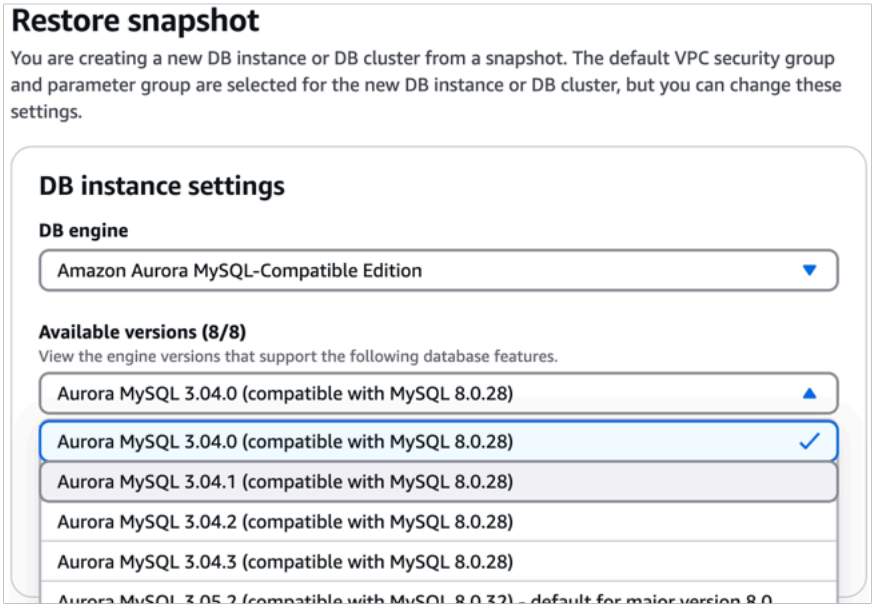
> **Note**
>
> Während die primäre DB-Instance erstellt wird, erscheint sie als Reader-Instance, wird aber nach der Erstellung zur Writer-Instance.

So stellen Sie einen DB-Cluster aus einem DB-Cluster-Snapshot wieder her

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Snapshots.
3. Wählen Sie den DB-Cluster-Snapshot für die Wiederherstellung aus.
4. Wählen Sie in Actions (Aktionen) die Option Restore Snapshot (Snapshot wiederherstellen) aus.



   Anschließend wird die Seite Restore snapshot (Snapshot wiederherstellen) angezeigt.
5. Führen Sie unter DB instance settings (DB-Instance-Einstellungen) folgende Schritte aus:
   a. Verwenden Sie die Standardeinstellung für DB engine (DB-Engine).
   b. Wählen Sie für Verfügbare Versionen eine mit MySQL 8.0 kompatible Version aus, z. B. Aurora MySQL 3.04.0 (mit MySQL 8.0.23 kompatibel).



6. Geben Sie unter Settings (Einstellungen) bei DB instance identifier (DB-Instance-Kennung) den eindeutigen Namen ein, den Sie für die wiederhergestellte DB-Instance verwenden möchten, zum Beispiel **`my-80`**.

   > **Note**
   >
   > Um die DB-Cluster-ID zu erstellen, hängt Amazon RDS `-cluster` an die von Ihnen angegebene DB-Instance-ID an.

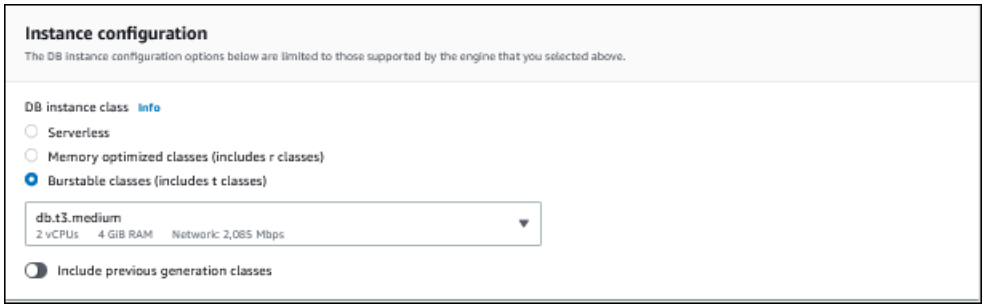
7. Verwenden Sie unter Konnektivität die Standardeinstellungen für folgende Optionen:

   - Virtual Private Cloud (VPC)
   - DB-Subnetzgruppe
   - Öffentlicher Zugriff
   - VPC security group (firewall) (VPC-Sicherheitsgruppe (Firewall))

8. Wählen Sie die DB instance class (DB-Instance-Klasse) aus.

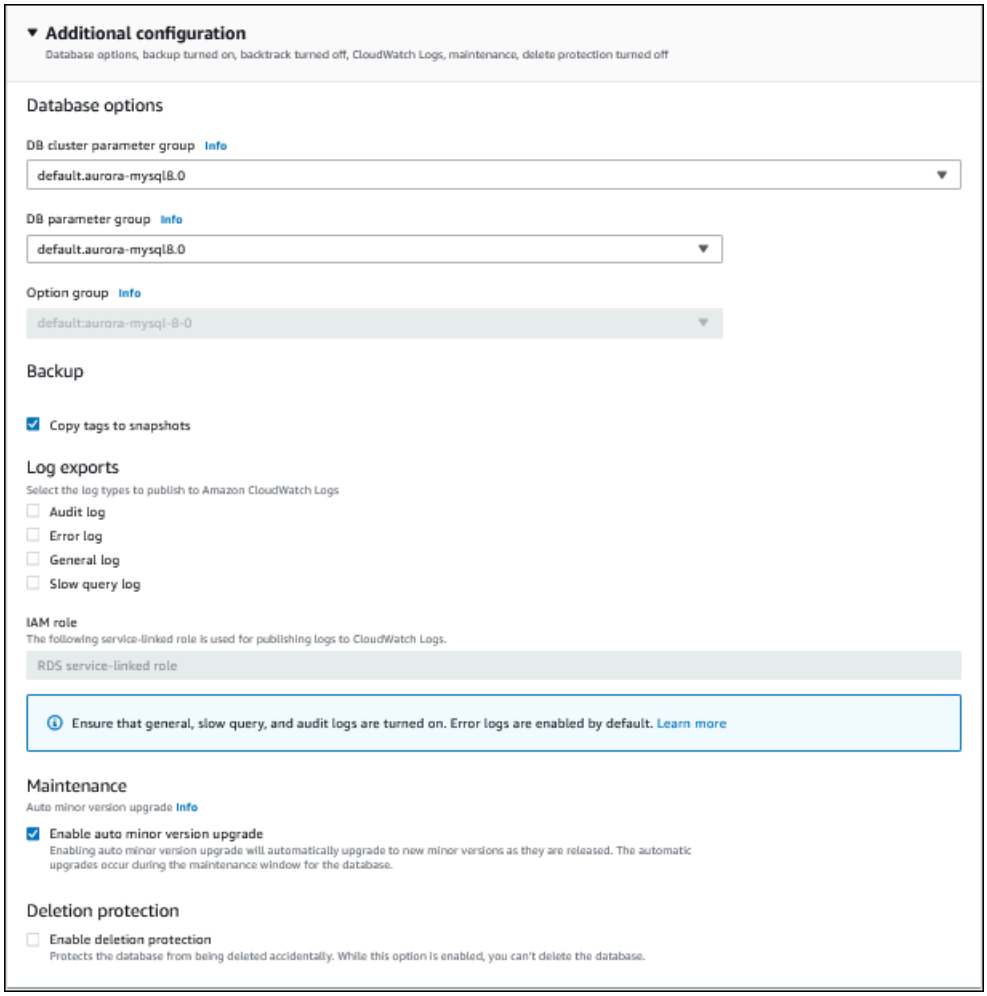
   Wählen Sie für dieses Tutorial Burstable classes (includes t classes) (Burst-fähige Klassen (einschließlich t-Klassen)) und dann db.t3.medium aus.

> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-Instance-Klassenarten.



9. Verwenden Sie für Database authentication (Datenbankauthentifizierung) die Standardeinstellung.
10. Für Encryption (Verschlüsselung) verwenden Sie die Standardeinstellungen.

    Wenn der Quell-DB-Cluster für den Snapshot verschlüsselt wurde, wird der wiederhergestellte DB-Cluster ebenfalls verschlüsselt. Die Verschlüsselung kann nicht aufgehoben werden.
11. Erweitern Sie Additional configuration (Zusätzliche Konfiguration) unten auf der Seite.



12. Nehmen Sie die folgenden Einstellungen vor:

    a. Verwenden Sie für dieses Tutorial den Standardwert für DB cluster parameter group (DB-Cluster-Parametergruppe).

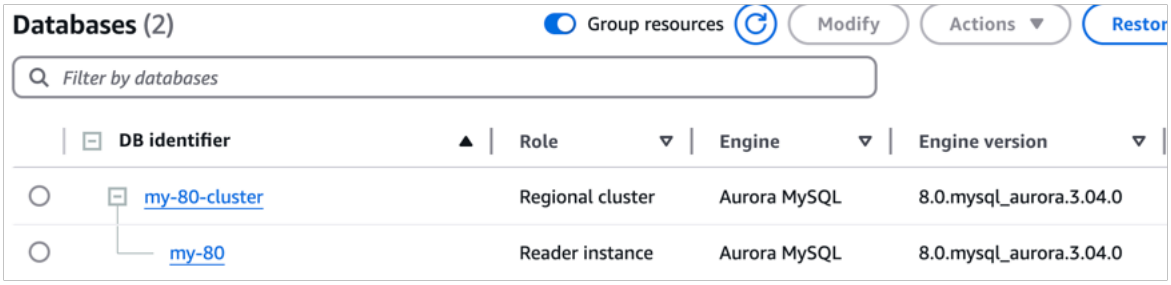
    b. Verwenden Sie für dieses Tutorial den Standardwert für DB parameter group (DB-Parametergruppe).

    c. Aktivieren Sie für Log exports (Protokollexporte) alle Kontrollkästchen.

    d. Aktivieren Sie für Deletion protection (Löschschutz) das Kontrollkästchen Enable deletion protection (Löschschutz aktivieren).

13. Klicken Sie auf Restore DB Instance (DB-Instance wiederherstellen).

Auf der Seite Databases (Datenbanken) wird der wiederhergestellte DB-Cluster mit dem Status `Creating` angezeigt.



Während die primäre DB-Instance erstellt wird, erscheint sie als Reader-Instance, wird aber nach der Erstellung zur Writer-Instance.

Tutorial: Wiederherstellen eines DB-Clusters aus einem DB-Cluster-Snapshot mithilfe der AWS CLI

In diesem Tutorial stellen Sie einen DB-Cluster mithilfe der AWS CLI aus einem DB-Cluster-Snapshot wieder her. Das Wiederherstellen eines DB-Clusters aus einem Snapshot mithilfe der AWS CLI erfolgt in zwei Schritten:

1. Wiederherstellen eines DB-Clusters unter Verwendung des Befehls restore-db-cluster-from-snapshot
2. Erstellen der primären (Writer-)DB-Instance unter Verwendung des create-db-instance-Befehls

Wiederherstellen eines DB-Clusters

Verwenden Sie den Befehl `restore-db-cluster-from-snapshot`. Die folgenden Optionen sind erforderlich:

- `--db-cluster-identifier` – Der Name des wiederhergestellten DB-Clusters.
- `--snapshot-identifier` – Der Name des DB-Snapshots, der für die Wiederherstellung verwendet wird.
- `--engine` – Die Datenbank-Engine des wiederhergestellten DB-Clusters. Sie muss mit der Datenbank-Engine des Quell-DB-Clusters kompatibel sein.

  Es stehen folgende Optionen zur Verfügung:

  - `aurora-mysql` – mit Aurora MySQL 5.7 und 8.0 kompatibel
  - `aurora-postgresql` – mit Aurora PostgreSQL kompatibel

  In diesem Beispiel verwenden wir `aurora-mysql`.

- `--engine-version` – die Version des zu erstellenden DB-Clusters In diesem Beispiel verwenden wir eine mit MySQL 8.0 kompatible Version.

Im folgenden Beispiel wird ein mit Aurora MySQL 8.0 kompatibler DB-Cluster mit dem Namen `my-new-80-cluster` aus einem DB-Cluster-Snapshot namens `my-57-cluster-snapshot` wiederhergestellt.

So stellen Sie den DB-Cluster wieder her

- Verwenden Sie einen der folgenden Befehle.

  Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-80-cluster \
    --snapshot-identifier my-57-cluster-snapshot \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.02.0
```

  Für Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier my-new-80-cluster ^
    --snapshot-identifier my-57-cluster-snapshot ^
    --engine aurora-mysql ^
    --engine-version 8.0.mysql_aurora.3.02.0
```

Die Ausgabe sieht in etwa folgendermaßen aus.

```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-central-1b",
            "eu-central-1c",
            "eu-central-1a"
        ],
        "BackupRetentionPeriod": 14,
        "DatabaseName": "",
        "DBClusterIdentifier": "my-new-80-cluster",
```

```
        "DBClusterParameterGroup": "default.aurora-mysql8.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "my-new-80-cluster.cluster-############.eu-
central-1.rds.amazonaws.com",
        "ReaderEndpoint": "my-new-80-cluster.cluster-ro-############.eu-
central-1.rds.amazonaws.com",
        "MultiAZ": false,
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "Port": 3306,
        "MasterUsername": "admin",
        "PreferredBackupWindow": "01:55-02:25",
        "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z1RLNU0EXAMPLE",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#######-5ccc-49cc-8aaa-
############",
        "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
        "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-
cluster",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": []
    }
}
```

Erstellen der primären (Writer-)DB-Instance

Verwenden Sie den Befehl `create-db-instance`, um die primäre (Writer-)DB-Instance zu erstellen. Die folgenden Optionen sind erforderlich:

- `--db-cluster-identifier` – Der Name des wiederhergestellten DB-Clusters.
- `--db-instance-identifier` – Der Name der primären DB-Instance.
- `--db-instance-class` – Die Instance-Klasse der primären DB-Instance. In diesem Beispiel verwenden wir `db.t3.medium`.

  > **Note**
  >
  > Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-Instance-Klassenarten.

- `--engine` – Die Datenbank-Engine der primären DB-Instance. Es muss dieselbe Datenbank-Engine sein, die der wiederhergestellte DB-Cluster verwendet.

  Es stehen folgende Optionen zur Verfügung:

  - `aurora-mysql` – mit Aurora MySQL 5.7 und 8.0 kompatibel
  - `aurora-postgresql` – mit Aurora PostgreSQL kompatibel

  In diesem Beispiel verwenden wir `aurora-mysql`.

Im folgenden Beispiel wird eine primäre (Writer-)DB-Instance mit dem Namen `my-new-80-cluster-instance` im wiederhergestellten DB-Cluster, der mit Aurora MySQL 8.0 kompatibel ist, namens `my-new-80-cluster` erstellt.

So erstellen Sie die primäre DB-Instance

- Verwenden Sie einen der folgenden Befehle.

  Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-80-cluster \
    --db-instance-identifier my-new-80-cluster-instance \
    --db-instance-class db.t3.medium \
```

```
    --engine aurora-mysql
```

Für Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-80-cluster ^
    --db-instance-identifier my-new-80-cluster-instance ^
    --db-instance-class db.t3.medium ^
    --engine aurora-mysql
```

Die Ausgabe sieht in etwa folgendermaßen aus.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-new-80-cluster-instance",
        "DBInstanceClass": "db.t3.medium",
        "Engine": "aurora-mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 1,
        "PreferredBackupWindow": "01:55-02:25",
        "BackupRetentionPeriod": 14,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.aurora-mysql8.0",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2305ca49",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
```

```
                "SubnetIdentifier": "subnet-########",
                "SubnetAvailabilityZone": {
                    "Name": "eu-central-1a"
                },
                "SubnetOutpost": {},
                "SubnetStatus": "Active"
            },
            {
                "SubnetIdentifier": "subnet-########",
                "SubnetAvailabilityZone": {
                    "Name": "eu-central-1b"
                },
                "SubnetOutpost": {},
                "SubnetStatus": "Active"
            },
            {
                "SubnetIdentifier": "subnet-########",
                "SubnetAvailabilityZone": {
                    "Name": "eu-central-1c"
                },
                "SubnetOutpost": {},
                "SubnetStatus": "Active"
            }
        ]
    },
    "PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
    "PendingModifiedValues": {},
    "MultiAZ": false,
    "EngineVersion": "8.0.mysql_aurora.3.02.0",
    "AutoMinorVersionUpgrade": true,
    "ReadReplicaDBInstanceIdentifiers": [],
    "LicenseModel": "general-public-license",
    "OptionGroupMemberships": [
        {
            "OptionGroupName": "default:aurora-mysql-8-0",
            "Status": "in-sync"
        }
    ],
    "PubliclyAccessible": false,
    "StorageType": "aurora",
    "DbInstancePort": 0,
    "DBClusterIdentifier": "my-new-80-cluster",
    "StorageEncrypted": true,
```

```
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#######-5ccc-49cc-8aaa-
############",
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
        "TagList": []
    }
}
```

Überwachung von Metriken in einem Amazon-Aurora-Cluster

Amazon Aurora verwendet einen Cluster replizierter Datenbankserver. Das Überwachen eines Aurora-Clusters erfordert in der Regel die Überprüfung des Zustands mehrerer DB-Instances. Die Instances haben möglicherweise spezielle Rollen, die hauptsächlich Schreiboperationen, nur Lesevorgänge oder eine Kombination verarbeiten. Sie überwachen auch den Gesamtzustand des Clusters, indem Sie die Replikationsverzögerung messen. Dabei handelt es sich um die Zeitspanne, in der Änderungen, die von einer DB-Instance vorgenommen wurden, für die anderen Instances verfügbar sind.

Themen

- Überwachungsplan
- Leistungsbasislinie
- Richtlinien zur Leistung
- Überwachungstools für Amazon Aurora
- Anzeigen des Status der Cluster
- Empfehlungen von Amazon Aurora
- Anzeigen von Metriken in der Amazon-RDS-Konsole
- Anzeigen kombinierter Metriken mit dem Performance-Insights-Dashboard
- Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch
- Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights
- Überwachung mit Performance Insights auf
- Analysieren von Aurora-Leistungsanomalien mit Amazon DevOps Guru für Amazon RDS
- Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung)
- Amazon Aurora-Referenz für Metriken

Überwachungsplan

Bevor Sie mit der Überwachung von Amazon Aurora beginnen, sollten Sie einen Überwachungsplan erstellen. Dieser Plan sollte die folgenden Fragen beantworten:

- Was sind Ihre Ziele bei der Überwachung?

- Welche Ressourcen möchten Sie überwachen?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungs-Tools möchten Sie verwenden?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Leistungsbasislinie

Um Ihre Überwachungsziele zu erreichen, müssen Sie eine Basislinie erstellen. Messen Sie dazu die Leistung unter unterschiedlichen Lastbedingungen zu verschiedenen Zeiten in Ihrer Amazon Aurora-Umgebung. Sie können Metriken wie die folgenden überwachen:

- Netzwerkdurchsatz
- Client-Verbindungen
- I/O für Lese-, Schreib- oder Metadatenoperationen
- Burst-Guthaben für Ihre DB-Instances

Es wird empfohlen, historische Leistungsdaten für Amazon Aurora zu speichern.. Mithilfe der gespeicherten Daten können Sie die aktuelle Leistung mit früheren Trends vergleichen. Sie können so auch normale Leistungsmuster von Anomalien unterscheiden und Methoden zur Behebung von Problemen entwickeln.

Richtlinien zur Leistung

Die zulässigen Werte für Leistungsmetriken sind im Allgemeinen davon abhängig, wie die Basislinie aussieht, und wofür die Anwendung gedacht ist. Prüfen Sie, ob dauerhafte oder tendenzielle Abweichungen von Ihrer Ausgangsbasis vorliegen. Die folgenden Metriken sind häufig die Ursache von Leistungsproblemen:

- Hohe CPU- oder RAM-Nutzung – Hohe Werte für die CPU- oder RAM-Nutzung können angemessen sein, wenn sie der Zielsetzung Ihrer Anwendung entsprechen (z. B. in Bezug auf Durchsatz oder Gleichzeitigkeit) und erwartet werden.
- Nutzung des Datenträgerplatzes – Überprüfen Sie die Nutzung des Datenträgerplatzes, wenn konsistent 85 Prozent oder mehr des gesamten Datenträgerplatzes belegt werden. Prüfen Sie,

ob Daten in der Instance gelöscht oder auf einem anderen System archiviert werden können, um Speicherplatz freizugeben.

- Netzwerkdatenverkehr – Wenden Sie sich an Ihren Systemadministrator, um zu erfahren, welcher Durchsatz für Ihr Domänennetzwerk und Ihre Internetverbindung erwartet wird. Überprüfen Sie den Netzwerkdatenverkehr, wenn der Durchsatz dauerhaft unter dem erwarteten Wert liegt.
- Datenbankverbindungen – Sie sollten eine Einschränkung der Datenbankverbindungen in Betracht ziehen, wenn eine große Anzahl von Benutzerverbindungen sowie auch eine Abnahme der Instance-Leistung und -Reaktionszeit zu erkennen sind. Die optimale Anzahl der Benutzerverbindungen für Ihre DB-Instance ist von der Instance-Klasse und der Komplexität der ausgeführten Operationen abhängig. Um die Anzahl der Datenbankverbindungen zu bestimmen, ordnen Sie Ihre DB-Instance einer Parametergruppe zu, für die der Parameter `User Connections` auf einen anderen Wert als „0“ (unbegrenzt) gesetzt ist. Sie können eine entweder eine vorhandene Parametergruppe verwenden oder eine neue erstellen. Weitere Informationen finden Sie unter Parametergruppen für Amazon Aurora.
- IOPS-Metriken – Die erwarteten Werte für IOPS-Metriken sind von der Datenträgerspezifikation und der Serverkonfiguration abhängig. Verwenden Sie die Basiswerte als typische Werte. Prüfen Sie, ob dauerhafte Abweichungen von den Werten Ihrer Ausgangsbasis vorliegen. Für eine optimale IOPS-Leistung stellen Sie sicher, dass Ihr typisches Working Set in den Speicher passt, um Lese- und Schreibvorgänge zu minimieren.

Wenn die Leistung außerhalb der festgelegten Baseline liegt, müssen Sie möglicherweise Änderungen vornehmen, um die Datenbankverfügbarkeit für Ihre Workload zu optimieren. Beispielsweise müssen Sie möglicherweise die Instance-Klasse Ihrer DB-Instance ändern. Oder Sie müssen möglicherweise die Anzahl der DB-Instance und Read Replicas ändern, die für Clients verfügbar sind.

Überwachungstools für Amazon Aurora

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit und Leistung von Amazon Aurora und Ihren anderen AWS-Lösungen. AWS bietet verschiedene Überwachungswerkzeuge, um Amazon Aurora zu beobachten, Probleme zu melden und ggf. automatisch Maßnahmen ergreifen zu können.

Themen

- Automatisierte Überwachungstools
- Manuelle Überwachungstools

Automatisierte Überwachungstools

Wir empfehlen, dass Sie die Überwachungsaufgaben möglichst automatisieren.

Themen

- Amazon-Aurora-Cluster-Status und Empfehlungen
- Amazon-CloudWatch-Metriken für Amazon Aurora
- Amazon RDS Performance Insights und Betriebssystemüberwachung
- Integrierte Services

Amazon-Aurora-Cluster-Status und Empfehlungen

Sie können die folgenden automatisierten Tools zur Überwachung von Amazon Aurora verwenden und auftretende Probleme melden:

- Amazon Aurora--Cluster-Status – Zeigen Sie über die Amazon-RDS-Konsole, den AWS CLI oder die RDS-API-Details zum aktuellen Status Ihres -Clusters an.
- Amazon Aurora Empfehlungen — Reagieren Sie auf automatisierte Empfehlungen für Datenbankressourcen wie DB-Instances, DB-Cluster, und DB-Cluster-Parametergruppen. Weitere Informationen finden Sie unter Empfehlungen von Amazon Aurora.

Amazon-CloudWatch-Metriken für Amazon Aurora

Amazon Aurora integriert sich mit Amazon CloudWatch für zusätzliche Überwachungsfunktionen.

- Amazon CloudWatch – überwacht Ihre AWS-Ressourcen und -Anwendungen, die Sie auf AWS ausführen, in Echtzeit. Mit Amazon Aurora können Sie die folgenden Amazon CloudWatch-Funktionen verwenden.
  - Amazon CloudWatch Metriken sendet – automatisch Metriken an CloudWatch jede Minute für jede aktive Datenbank. Sie erhalten keine zusätzlichen Gebühren für Amazon-RDS-Metriken in CloudWatch. Weitere Informationen finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora.
  - Amazon CloudWatch Alarme – Sie können eine einzelne Metrik über einen bestimmten Zeitraum ansehen. Anschließend können Sie eine oder mehrere Aktionen basierend auf dem Wert der Metrik relativ zu einem von Ihnen festgelegten Schwellenwert ausführen.

Amazon RDS Performance Insights und Betriebssystemüberwachung

Sie können die folgenden automatisierten Tools zum Überwachen der Amazon Aurora-Leistung verwenden:

- Amazon RDS Performance Insights – Bewerten Sie die Belastung Ihrer Datenbank und bestimmen Sie, wann und wo Maßnahmen ergriffen werden sollen. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .
- Amazon RDS Enhanced Monitoring – Sehen Sie sich in Echtzeit Metriken für das Betriebssystem an. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).

Integrierte Services

Die folgenden AWS-Services sind in Amazon Aurora integriert:

- Amazon EventBridge ist ein Serverless-Ereignisbus-Service, über den Sie Ihre Anwendungen einfach mit Daten aus einer Vielzahl von Quellen verbinden können. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora-Ereignissen.
- Mit Amazon CloudWatch Logs können Sie Ihre Protokolldateien von Amazon Aurora-Instances, CloudTrail oder anderen Quellen überwachen, speichern und darauf zugreifen. Weitere Informationen finden Sie unter Überwachen von Amazon Aurora-Protokolldateien.
- AWS CloudTrail erfasst API-Aufrufe und zugehörige Ereignisse, die von oder im Namen Ihres AWS-Konto-Kontos erfolgten, und übermittelt die Protokolldateien an einen von Ihnen

angegebenen Amazon-S3-Bucket. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora-API-Aufrufen in AWS CloudTrail.

- Database Activity Streams (Datenbankaktivitäts-Streams) ist eine Amazon Aurora-Funktion, die einen nahezu Echtzeitdatenstrom der Aktivität in Ihrer DB-Cluster- bietet. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.
- DevOps Guru for RDS ist eine Funktion von Amazon DevOps Guru, die Machine Learning auf Performance-Insights-Metriken für Amazon-Aurora-Datenbanken anwendet. Weitere Informationen finden Sie unter Analysieren von Aurora-Leistungsanomalien mit Amazon DevOps Guru für Amazon RDS.

Manuelle Überwachungstools

Sie müssen die Elemente manuell überwachen, die von den CloudWatch-Alarmen nicht abgedeckt werden. Die Dashboards der Amazon-RDS-, Cloudwatch-, AWS Trusted Advisor- und AWS-Konsole bieten eine Übersicht über den Zustand Ihrer AWS-Umgebung. Wir empfehlen, auch die Protokolldateien auf Ihrer DB-Instance zu überprüfen.

- In der Amazon-RDS-Konsole können Sie die folgenden Elemente für Ihre Ressourcen überwachen:
  - Die Anzahl der Verbindungen zu einer DB-Instance
  - Die Anzahl der Lese- und Schreibvorgänge in einer DB-Instance
  - Der von einer DB-Instance aktuell verwendete Speicherplatz
  - Der aktuell verwendete Arbeitsspeicher und die aktuelle CPU-Nutzung einer DB-Instance
  - Der Netzwerkdatenverkehr von und zu einer DB-Instance
- Über das Trusted Advisor-Dashboard können Sie die folgenden Prüfungen in Bezug auf die Kostenoptimierung, Sicherheit, Fehlertoleranz und Leistungssteigerung einsehen:
  - Amazon-RDS-DB-Instances im Leerlauf
  - Zugriffsrisiko für Amazon-RDS-Sicherheitsgruppen
  - Amazon-RDS-Backups
  - Amazon-RDS-Multi-AZ
  - Barrierefreiheit von Aurora-DB-Instances

  Weitere Informationen zu diesen Prüfungen finden Sie unter Trusted Advisor – bewährte Methoden (Prüfungen).
- CloudWatch-Homepage zeigt folgende Informationen an:

- Aktuelle Alarme und Status
- Diagramme mit Alarmen und Ressourcen
- Servicestatus

Zusätzlich können Sie CloudWatch für folgende Aufgaben nutzen:

- Erstellen Sie benutzerdefinierte Dashboards zur Überwachung der Services, die Ihnen wichtig sind.
- Aufzeichnen von Metrikdaten, um Probleme zu beheben und Trends zu erkennen.
- Suchen und durchsuchen aller AWS-Ressourcenmetriken
- Erstellen und Bearbeiten von Alarmen, um über Probleme benachrichtigt zu werden.

Anzeigen des Status der Cluster

Mit der Amazon-RDS-Konsole können Sie schnell auf den Status Ihrer DB-Cluster- zugreifen.

Themen

- Anzeigen eines DB-Clusters von Amazon Aurora
- Anzeigen des DB-Clusterstatus
- Anzeigen von DB-Instance-Status in einem Aurora-Cluster

Anzeigen eines DB-Clusters von Amazon Aurora

Es gibt mehrere Möglichkeiten, Informationen zu Ihren Amazon Aurora-DB-Clustern und DB-Instances in Ihren DB-Clustern anzuzeigen.

- Sie können DB-Cluster und DB-Instances in der Amazon-RDS-Konsole durch Auswahl von Databases (Datenbanken) in der Navigationsebene anzeigen.
- Sie können DB-Cluster- und DB-Instance-Informationen mit AWS Command Line Interface (AWS CLI) abrufen.
- Sie können Informationen über DB-Cluster und DB-Instances mit der Amazon-RDS-API abrufen.

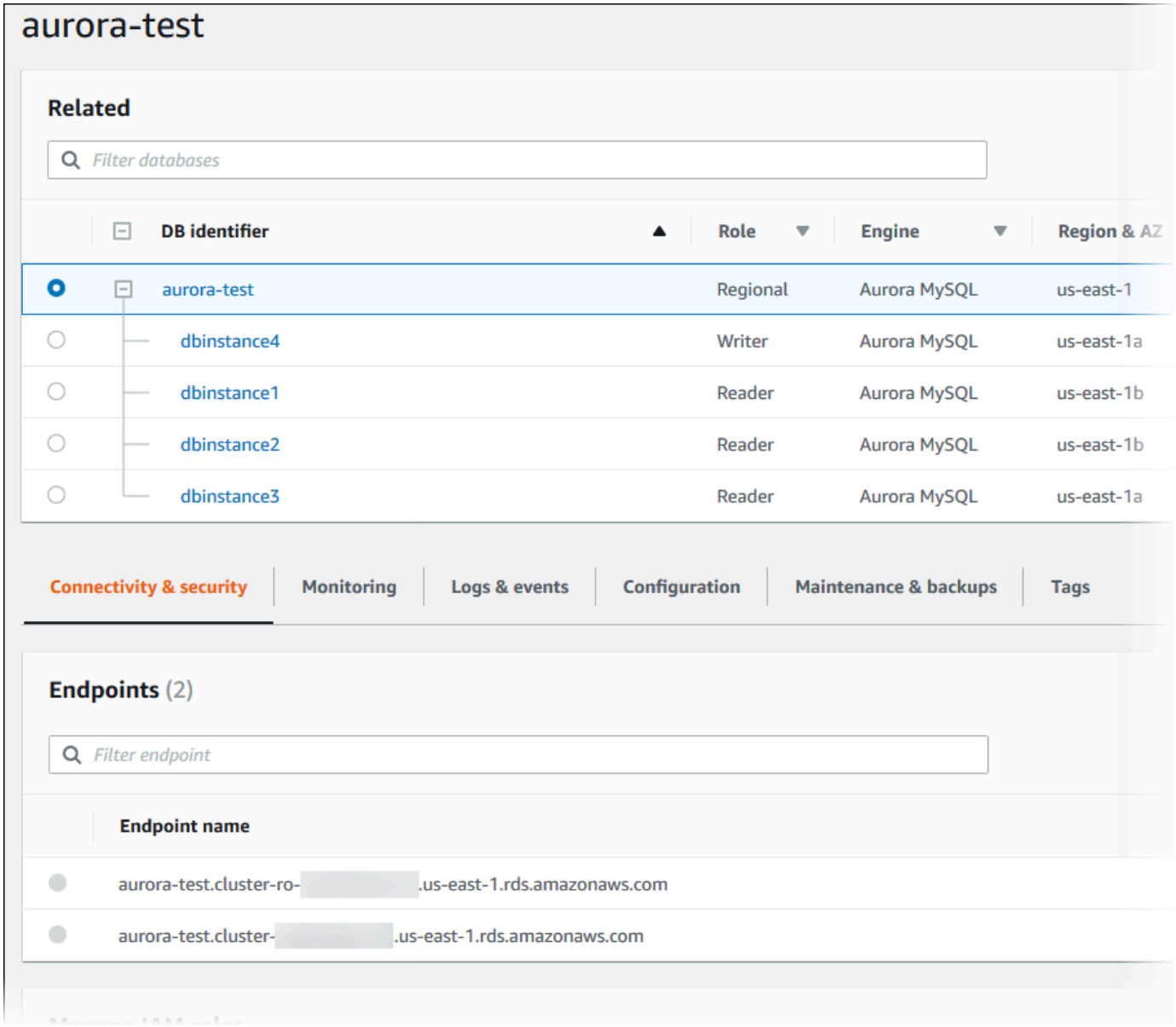
Konsole

In der Amazon RDS Konsole können Sie Details zu einem DB-Cluster anzeigen, indem Sie im Navigationsbereich der Konsole die Option Datenbanken auswählen. Details zu DB-Instances, die einem Amazon Aurora-DB-Cluster angehören, können Sie auch anzeigen.

So zeigen Sie DB-Cluster in der Amazon-RDS-Konsole an bzw. ändern sie

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie in der Liste den Namen des Aurora-DB-Clusters aus, den Sie anzeigen möchten.

   In der folgenden Abbildung wird beispielsweise die Detailseite für den DB-Cluster mit der Bezeichnung `aurora-test` angezeigt. Für den DB-Cluster werden in der Liste DB identifier (DB-Kennung) vier DB-Instances angezeigt. Die Writer-DB-Instance, `dbinstance4`, ist die primäre DB-Instance für den DB-Cluster.



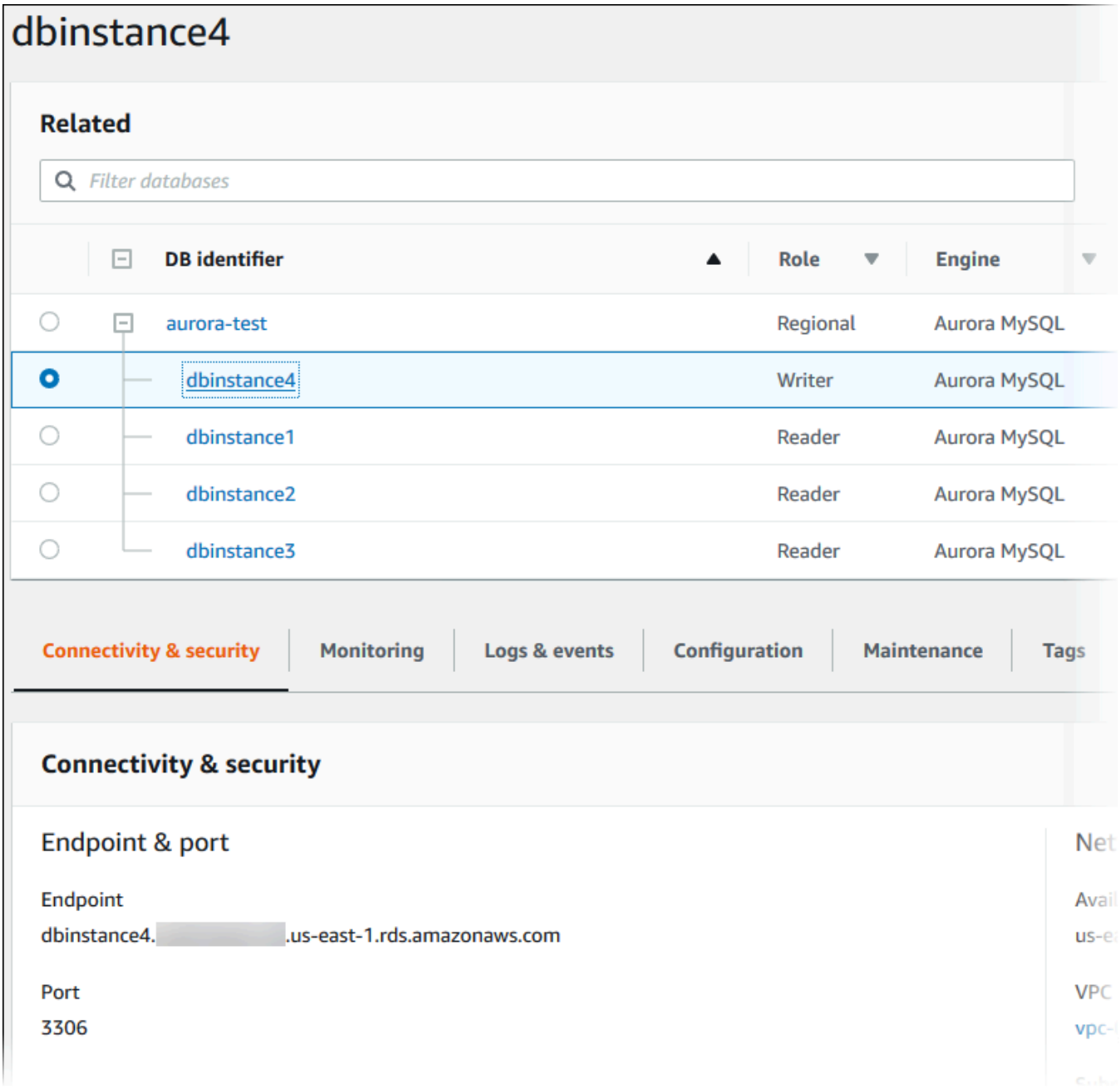
4. Um einen DB-Cluster zu ändern, wählen Sie ihn in der Liste aus gefolgt von Modify (Ändern).

So zeigen Sie DB-Instances eines DB-Clusters in der Amazon-RDS-Konsole an bzw. ändern sie

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Führen Sie eine der folgenden Aktionen aus:

   - Um eine DB-Instance anzuzeigen, wählen Sie eine aus der Liste aus, die Mitglied des Aurora-DB-Clusters ist.

Wenn Sie zum Beispiel auf die `dbinstance4` DB-Instance-Kennung klicken, zeigt die Konsole die Detailseite für die `dbinstance4` DB-Instance an, wie in der folgenden Abbildung dargestellt.



- Um eine DB-Instance zu ändern, wählen Sie die DB-Instance aus der Liste und Modify (Ändern) aus. Weitere Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

AWS CLI

Verwenden Sie den describe-db-clustersBefehl AWS CLI, um DB-Cluster-Informationen mithilfe von anzuzeigen. Der folgende AWS CLI Befehl listet beispielsweise die DB-Clusterinformationen für alle DB-Cluster in der `us-east-1` Änderungsregion für das konfigurierte AWS Konto auf.

```
aws rds describe-db-clusters --region us-east-1
```

Der Befehl gibt die folgende Ausgabe zurück, wenn Ihre für die JSON-Ausgabe konfiguriert AWS CLI ist.

```
{
    "DBClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "sample-cluster1.cluster-123456789012.us-
east-1.rds.amazonaws.com"
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "sample-cluster1",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T03:35:42.563Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-replica"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-primary"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "03:34-04:04",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-ddb65fec"
                }
            ],
            "DBSubnetGroup": "default",
```

```
            "StorageEncrypted": false,
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:06:08.903Z",
            "PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
        },
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "aurora-sample.cluster-123456789012.us-
east-1.rds.amazonaws.com",
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "aurora-sample-cluster",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T10:21:34.826Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-replica-sample"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-sample"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "10:20-10:50",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-55da224b"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
```

```
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:00:11.491Z",
            "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
        }
    ]
}
```

RDS-API

Verwenden Sie den DBClusters Vorgang Describe, um DB-Cluster-Informationen mithilfe der Amazon RDS-API anzuzeigen.

Anzeigen des DB-Clusterstatus

Der Status eines DB-Clusters gibt seinen aktuellen Betriebsstatus an. Sie können den Status eines DB-Clusters und der Cluster-Instances mithilfe der Amazon RDS-Konsole AWS CLI, der oder der API anzeigen.

> **Note**
>
> Aurora führt daneben noch einen anderen Statuswert, der als Wartungsstatus bezeichnet wird, und der in der Spalte Wartung in der Amazon-RDS-Konsole angezeigt wird. Dieser Wert gibt den Status der Wartungspatches an, die auf dem DB-Cluster installiert werden müssen. Der Wartungsstatus ist von dem Status des DB-Clusters unabhängig. Weitere Informationen zum Wartungsstatus finden Sie unter Anwenden von Updates auf einen DB-Cluster.

Die möglichen Statuswerte für DB-Cluster sind in der folgenden Tabelle aufgeführt.

DB-Cluster-Status	Berechnet	Beschreibung
Verfügbar	Berechnet	Der DB-Cluster ist für Änderungen verfügbar. Wenn ein Aurora Serverless Cluster verfügbar und angehalten ist, wird Ihnen nur der Speicher berechnet.
Backing-up	Berechnet	Der DB-Cluster wird derzeit gesichert.
backtracking	Berechnet	Der DB-Cluster wird derzeit rückverfolgt. Dieser Status gilt nur für Aurora MySQL.
Cloning-failed	Nicht berechnet	Das Klonen eines DB-Clusters ist fehlgesch lagen.
Erstellen	Nicht berechnet	Der DB-Cluster wird gerade erstellt. Während der DB-Cluster erstellt wird, kann nicht darauf zugegriffen werden.
Wird gelöscht	Nicht berechnet	Der DB-Cluster wird gerade gelöscht.

DB-Cluster-Status	Berechnet	Beschreibung
Failing-over	Berechnet	Es wird gerade ein Failover von der primären Instance in ein Aurora-Replica durchgeführt.
Inaccessible-encryption-credentials	Nicht berechnet	Auf das zum Verschlüsseln oder Entschlüsseln des DB-Clusters AWS KMS key verwendet e kann nicht zugegriffen oder es kann nicht wiederhergestellt werden.
Inaccessible-encryption-credentials-recoverable	Berechnet für Speicher	Es ist kein Zugriff auf den KMS-Schlüssel möglich, der zum Ver- oder Entschlüsseln des DB-Clusters verwendet wird. Wenn der KMS-Schlüssel jedoch aktiv ist, kann ein Neustart des DB-Clusters ihn wiederherstellen. Weitere Informationen finden Sie unter Verschlüsseln eines Amazon-Aurora-DB-Clusters.
Wartung	Berechnet	Amazon RDS installiert auf dem DB-Cluster ein Wartungsupdate. Dieser Status wird bei Wartungen auf der Ebene des DB-Clusters verwendet, die von RDS lange im Voraus geplant werden.
Migrating	Berechnet	Es wird gerade ein DB-Cluster-Snapshot in einen DB-Cluster wiederhergestellt.
Migration-failed	Nicht berechnet	Eine Migration ist fehlgeschlagen.
Ändern	Berechnet	Der DB-Cluster wird aufgrund einer Kundenanfrage, den DB-Cluster zu ändern, geändert.
Promoting	Berechnet	Ein Lesereplikat wird gerade zu einem eigenständigen DB-Cluster hochgestuft.
Preparing-data-migration	Berechnet	Amazon RDS bereitet die Migration von Daten zu Aurora vor.

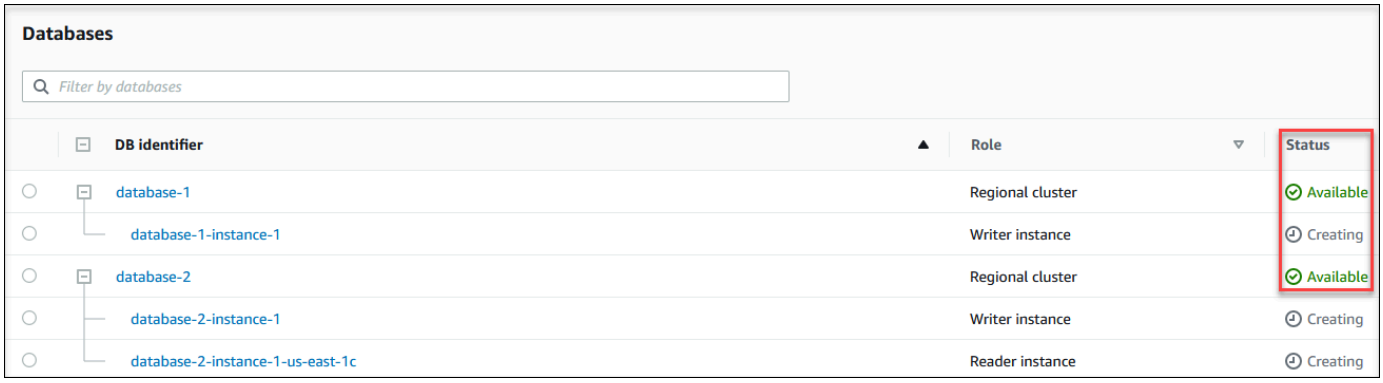
DB-Cluster-Status	Berechnet	Beschreibung
Wird umbenannt	Berechnet	Der DB-Cluster wird aufgrund einer Kundenanfrage auf Instance-Umbenennung umbenannt.
Resetting-master-credentials	Berechnet	Die Master-Anmeldeinformationen für den DB-Cluster werden auf eine Kundenanfrage auf Zurücksetzung hin zurückgesetzt.
Wird gestartet	Berechnet für Speicher	Der DB-Cluster wird gestartet.
Angehalten	Berechnet für Speicher	Der DB-Cluster wird gestoppt.
Wird angehalten	Berechnet für Speicher	Der DB-Cluster wird gerade angehalten.
Storage-optimization	Berechnet	Ihre DB-Instance wird modifiziert, um die Speichergröße oder den Speichertyp zu ändern. Die DB-Instance ist voll funktionsfähig. Während sich die DB-Instance aber im Zustand der Speicheroptimierung befindet, können Sie keine Änderungen am Speicher der DB-Instance anfordern. Der Prozess der Speicheroptimierung ist in der Regel kurz, kann aber manchmal bis zu und sogar über 24 Stunden dauern.
Update-iam-db-auth	Berechnet	Die IAM-Autorisierung für den DB-Cluster wird gerade aktualisiert.
Wird upgegradet	Berechnet	Die DB-Cluster-Engine- oder Betriebssystemversion wird aktualisiert.
upgrade_failed	Nicht berechnet	Der DB-Cluster konnte nicht auf eine unterstützte Version aktualisiert werden. Aurora erstellt einen endgültigen Snapshot mit dem Präfix`rds-final` .

Konsole

So zeigen Sie den Status eines DB-Clusters an

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.

   Die Seite „Datenbanken“ wird zusammen mit der Liste der DB-Cluster angezeigt. Für die einzelnen DB-Cluster wird der Statuswert angezeigt.



CLI

Um nur den Status der DB-Cluster anzuzeigen, verwenden Sie die folgende Abfrage unter AWS CLI.

```
aws rds describe-db-clusters --query 'DBClusters[*].[DBClusterIdentifier,Status]' --
output table
```

Anzeigen von DB-Instance-Status in einem Aurora-Cluster

Der Status einer DB-Instance in einem Aurora-Cluster gibt ihren aktuellen Betriebsstatus an. Sie können die folgenden Verfahren verwenden, um den DB-Instance-Status eines Clusters in der Amazon RDS-Konsole, im AWS CLI Befehl oder im API-Vorgang anzuzeigen.

> **Note**
>
> Amazon RDS führt daneben noch einen anderen Statuswert, der als Wartungsstatus bezeichnet wird, und der in der Spalte Wartung in der Amazon-RDS-Konsole angezeigt wird. Dieser Wert gibt den Status der Wartungspatches an, die auf der DB-Instance installiert werden müssen. Der Wartungsstatus ist von dem Status der DB-Instance unabhängig. Weitere Informationen zum Wartungsstatus finden Sie unter Anwenden von Updates auf einen DB-Cluster.

In der folgenden Tabelle finden Sie die möglichen Statuswerte für DB-Instances. Diese Tabelle zeigt auch, ob Ihnen die DB-Instance und der Speicher in Rechnung gestellt werden, die nur für Speicher in Rechnung gestellt oder nicht in Rechnung gestellt werden. Für alle DB-Instance-Status wird immer die Sicherungsnutzung berechnet.

DB-Instance-Status	Berechnet	Beschreibung
available	Berechnet	Die DB-Instance ist für Änderungen verfügbar.
backing-up	Berechnet	Die DB-Instance wird derzeit gesichert.
backtracking	Berechnet	Die DB-Instance wird derzeit rückverfolgt. Dieser Status gilt nur für Aurora MySQL.
configuring-enhanced-monitoring	Berechnet	Für diese DB-Instance wird „Enhanced Monitoring“ (Erweiterte Überwachung) aktiviert oder deaktiviert.
configuring-iam-database-auth	Berechnet	AWS Identity and Access Management Die (IAM-) Datenbankauthentifizierung wird für diese DB-Instance aktiviert oder deaktiviert.

DB-Instance-Status	Berechnet	Beschreibung
configuring-log-exports	Berechnet	Das Veröffentlichen von Protokolldateien in Amazon CloudWatch Logs wird für diese DB-Instance aktiviert oder deaktiviert.
converting-to-vpc	Berechnet	Die sich nicht in einer Amazon Virtual Private Cloud (Amazon VPC) befindliche DB-Instance wird in eine DB-Instance umgewandelt, die sich in einer Amazon VPC befindet.
creating	Nicht berechnet (kein PITR) Berechnet (nur PITR)	Die DB-Instance wird gerade erstellt. Während die DB-Instance erstellt wird, kann nicht auf sie zugegriff en werden. Wenn Sie eine Datenbank während der point-in-time Wiederherstellung (PITR) wiederherstellen, wird Ihnen die Rechnung gestellt, während sich die Datenbank im Erstellungszustand befindet. Dies ist das einzige Szenario, in dem für den Status creating Gebühren anfallen.
Delete-precheck	Nicht berechnet	Amazon RDS überprüft, ob Read Replicas sicher gelöscht werden können.
deleting	Nicht berechnet	Die DB-Instance wird gerade gelöscht.
failed	Nicht berechnet	Die DB-Instance befindet sich im Fehlerzustand und Amazon RDS kann sie nicht wiederherstellen. Führen Sie eine point-in-time Wiederherstellung auf den letzten wiederherstellbaren Zeitpunkt der DB-Instance durch, um die Daten wiederherzustellen.
inaccessible-encry ption-credentials	Nicht berechnet	Die zum Verschlüsseln oder Entschlüsseln der DB-Instance AWS KMS key verwendete Datei kann nicht abgerufen oder wiederhergestellt werden.

DB-Instance-Status	Berechnet	Beschreibung
inaccessible-encryption-credentials-recoverable	Berechnet für Speicher	Es ist kein Zugriff auf den KMS-Schlüssel möglich, der zum Ver- oder Entschlüsseln der DB-Instance verwendet wird. Wenn der KMS-Schlüssel jedoch aktiv ist, kann ein Neustart der DB-Instance ihn wiederherstellen. Weitere Informationen finden Sie unter Verschlüsseln eines Amazon-Aurora-DB-Clusters.
incompatible-create	Nicht berechnet	Amazon RDS versucht, eine DB-Instance zu erstellen. Dies ist aber nicht möglich, weil die Ressourcen nicht mit Ihrer DB-Instance kompatibel sind. Dieser Status kann auftreten, wenn das Instance-Profil für Ihre DB-Instance nicht über die richtigen Berechtigungen verfügt.
incompatible-network	Nicht berechnet	Amazon RDS versucht, auf der DB-Instance eine Wiederherstellungsaktion durchzuführen, was bislang gescheitert ist, weil der Status der VPC die Durchführung der Aktion verhindert. Dieser Status kann beispielsweise eintreten, wenn alle IP-Adressen in einem Subnetz belegt sind und Amazon RDS keine IP-Adresse für die DB-Instance abrufen kann.
incompatible-option-group	Berechnet	Amazon RDS hat versucht, eine Optionsgruppenänderung zu übernehmen, konnte den Vorgang jedoch nicht abschließen, und Amazon RDS kann keinen Rollback auf den vorherigen Optionsgruppenstatus durchführen. Weitere Informationen enthält die Liste Recent Events (Aktuelle Ereignisse) für die DB-Instance. Dieser Status kann beispielsweise eintreten, wenn eine Optionsgruppe eine Option wie TDE und die DB-Instance keine verschlüsselten Daten enthält.

DB-Instance-Status	Berechnet	Beschreibung
incompatible-parameters	Berechnet	Amazon RDS kann die DB-Instance nicht hochfahren, weil in der DB-Parametergruppe der DB-Instance angegebene Parameter nicht mit der DB-Instance kompatibel sind. Machen Sie die letzten Änderungen an den Parametern der DB-Instance rückgängig, um wieder auf die DB-Instance zugreifen zu können. Beachten Sie für weitere Informationen zu den inkompatiblen Parametern die Liste Recent Events (Aktuelle Ereignisse) für die DB-Instance.
incompatible-restore	Nicht berechnet	Amazon RDS kann keine point-in-time Wiederherstellung durchführen.
unzureichende Kapazität	Nicht berechnet	Amazon RDS kann Ihre Instance nicht erstellen, da derzeit keine ausreichende Kapazität verfügbar ist. Um Ihre DB-Instance in derselben AZ mit demselben Instance-Typ zu erstellen, löschen Sie Ihre DB-Instance, warten Sie ein paar Stunden und versuchen Sie erneut zu erstellen. Alternativ können Sie eine neue Instance mit einer anderen Instance-Klasse oder AZ erstellen.
maintenance	Berechnet	Amazon RDS installiert auf der DB-Instance ein Wartungsupdate. Dieser Status wird bei Wartungen auf der Ebene der Instance verwendet, die von RDS lange im Voraus geplant werden.
modifying	Berechnet	Die DB-Instance wird aufgrund einer Kundenanfrage geändert.
moving-to-vpc	Berechnet	Die DB-Instance wird in eine neue Amazon Virtual Private Cloud (Amazon VPC) verschoben.
rebooting	Berechnet	Die DB-Instance wird aufgrund einer Kundenanfrage oder eines Amazon-RDS-Prozesses neu gestartet.

DB-Instance-Status	Berechnet	Beschreibung
resetting-master-credentials	Berechnet	Die Masteranmeldeinformationen für die DB-Instance werden auf eine Kundenanfrage hin zurückgesetzt.
renaming	Berechnet	Die DB-Instance wird aufgrund einer Kundenanfrage umbenannt.
restore-error	Berechnet	Bei dem Versuch, einen Snapshot point-in-time oder aus einem Snapshot wiederherzustellen, ist bei der DB-Instance ein Fehler aufgetreten.
starting	Berechnet für Speicher	Die DB-Instance wird gestartet.
stopped	Berechnet für Speicher	Die DB-Instance ist angehalten.
stopping	Berechnet für Speicher	Die DB-Instance wird gestoppt.
storage-config-upgrade	Berechnet	Die Konfiguration des Speicherdateisystems der DB-Instance wird aktualisiert. Dieser Status gilt nur für grüne Datenbanken innerhalb einer Blau/Grün-Bereitstellung oder für DB-Instance-Lesereplikate.
storage-full	Berechnet	Die DB-Instance hat die Speicherkapazitäts zuteilung erreicht. Dies ist ein kritischer Status, wir empfehlen deshalb eine umgehende Behebung dieses Problems. Vergrößern Sie zu diesem Zweck den verfügbaren Speicher, indem Sie die DB-Instance ändern. Um diese Situation zu vermeiden, stellen Sie CloudWatch Amazon-Alarme so ein, dass Sie gewarnt werden, wenn der Speicherplatz knapp wird.

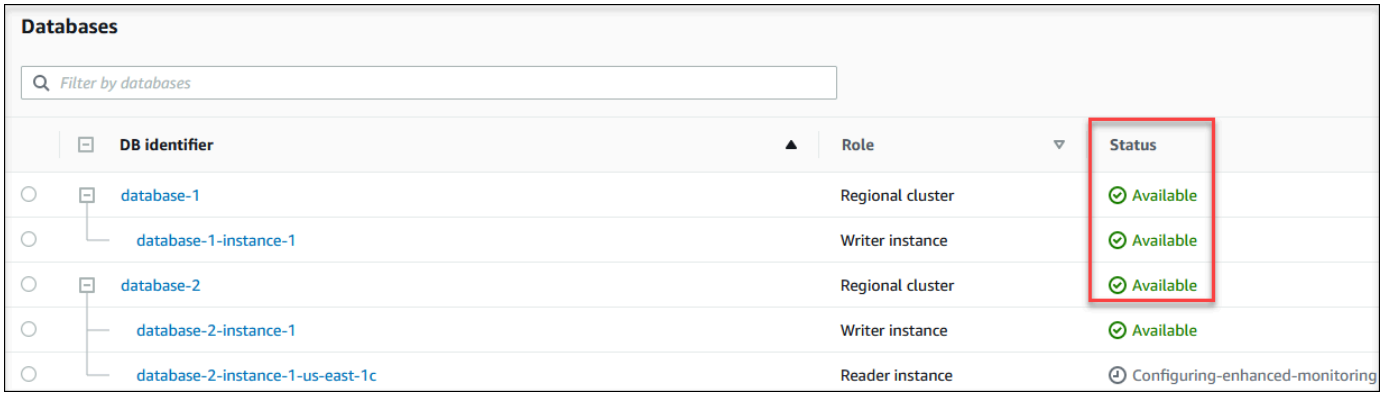
DB-Instance-Status	Berechnet	Beschreibung
storage-initialization	Berechnet	Die DB-Instance lädt Datenblöcke aus Amazon S3, um die Volume-Leistung nach der Wiederherstellung aus einem Snapshot zu optimieren. Sie bleibt für den Betrieb verfügbar, aber die Leistung ist möglicher weise erst dann voll ausgeschöpft, wenn die Initialis ierung abgeschlossen ist.
storage-optimization	Berechnet	Amazon RDS optimiert den Speicher Ihrer DB-Instan ce. Der Prozess der Speicheroptimierung ist in der Regel kurz, kann aber manchmal bis zu und sogar über 24 Stunden dauern. Während der Speicheroptimierung bleibt die DB-Instance verfügbar. Die Speicheroptimierung ist ein Hintergrundprozess, der die Verfügbarkeit der Instance nicht beeinträchtigt.
upgrading	Berechnet	Die Datenbank-Engine- oder die Betriebssystemvers ion wird aktualisiert.
upgrade_failed	Nicht berechnet	Die DB-Instance konnte nicht auf eine unterstützte Version aktualisiert werden. Aurora erstellt einen endgültigen Snapshot mit dem Präfix`rds-final` .

Konsole

So zeigen Sie den Status einer DB-Instance an

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.

   Die Seite „Datenbanken“ wird zusammen mit der Liste der DB-Instances angezeigt. Für jede DB-Instance in einem Cluster wird der Statuswert angezeigt.



CLI

Verwenden Sie den describe-db-instancesBefehl, um die AWS CLI DB-Instance und ihre Statusinformationen mithilfe von anzuzeigen. Der folgende AWS CLI Befehl listet beispielsweise alle Informationen zu DB-Instances auf.

```
aws rds describe-db-instances
```

Um eine bestimmte DB-Instance und ihren Status anzuzeigen, rufen Sie den describe-db-instancesBefehl mit der folgenden Option auf:

- `DBInstanceIdentifier` – der Name der DB-Instance

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

Um nur den Status aller DB-Instances anzuzeigen, verwenden Sie die folgende Abfrage in AWS CLI.

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

Rufen Sie den DBInstances Vorgang Describe auf, um den Status der DB-Instance mithilfe der Amazon RDS-API anzuzeigen.

Empfehlungen von Amazon Aurora

Amazon Aurora bietet automatisierte Empfehlungen für Datenbankressourcen wie DB-Instances, DB-Cluster, und DB-Cluster-Parametergruppen. Diese Empfehlungen bieten Anleitungen nach bewährten Methoden, indem sie die DB-Cluster-Konfiguration, DB-Instance-Konfiguration, die Nutzung und die Leistungsdaten analysieren.

Erkenntnisse zur Amazon-RDS-Leistung überwacht bestimmte Metriken und erstellt automatisch Schwellenwerte, indem analysiert wird, welche Werte für eine bestimmte Ressource als potenziell problematisch gelten. Wenn die neuen Metrikwerte über einen bestimmten Zeitraum einen vordefinierten Schwellenwert überschreiten, generiert Performance Insights eine proaktive Empfehlung. Diese Empfehlung trägt dazu bei, zukünftige Auswirkungen auf die Datenbankleistung zu verhindern. Beispielsweise wird die Empfehlung „Leerlauf in Transaktion“ für Aurora PostgreSQL-Instances generiert, wenn die mit der Datenbank verbundenen Sitzungen keine aktiven Aufgaben durchführen, aber Datenbankressourcen blockieren können. Um proaktive Empfehlungen zu erhalten, müssen Sie Performance Insights mit einer Aufbewahrungsfrist für ein kostenpflichtiges Kontingent aktivieren. Weitere Informationen zum Aktivieren von Performance Insights finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora. Weitere Informationen zur Preisgestaltung und Datenaufbewahrung für Performance Insights finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights.

DevOps Guru für RDS überwacht bestimmte Metriken, um zu erkennen, wann das Verhalten der Metrik sehr ungewöhnlich oder anomal wird. Diese Anomalien werden als reaktive Erkenntnisse mit Empfehlungen gemeldet. DevOps Guru for RDS empfiehlt Ihnen beispielsweise, die CPU-Kapazität zu erhöhen oder Warteereignisse zu untersuchen, die zur DB-Last beitragen. DevOps Guru für RDS bietet auch schwellenwertbasierte proaktive Empfehlungen. Für diese Empfehlungen müssen Sie DevOps Guru für RDS aktivieren. Weitere Informationen zum Aktivieren von DevOps Guru für RDS finden Sie unter DevOpsGuru einschalten und die Ressourcenabdeckung angeben.

Empfehlungen haben einen der folgenden Status: „Aktiv“, „Abgelehnt“, „Ausstehend“ oder „Gelöst“. Gelöste Empfehlungen sind 365 Tage lang verfügbar.

Sie können die Empfehlungen anzeigen oder verwerfen. Sie können eine konfigurationsbasierte aktive Empfehlung sofort ausführen, für das nächste Wartungsfenster einplanen oder verwerfen. Für auf Schwellenwerten basierende proaktive und auf Machine Learning basierende reaktive Empfehlungen müssen Sie die vorgeschlagene Ursache des Problems überprüfen und dann die empfohlenen Maßnahmen zur Behebung des Problems ergreifen.

Empfehlungen werden in den folgenden AWS-Regionen unterstützt:

- USA Ost (Ohio)
- USA Ost (Nord-Virginia)
- USA West (Nordkalifornien)
- USA West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asien-Pazifik (Singapur)
- Asien-Pazifik (Sydney)
- Asien-Pazifik (Tokio)
- Canada (Central)
- Europe (Frankfurt)
- Europa (Irland)
- Europa (London)
- Europe (Paris)
- Europa (Stockholm)
- Südamerika (São Paulo)

Erfahren Sie in den folgenden Abschnitten, wie Sie Empfehlungen von Amazon Aurora anzeigen, anwenden, ablehnen und ändern.

Themen

- Anzeigen von Amazon Aurora-Empfehlungen
- Anwenden von Empfehlungen von Amazon Aurora
- Verwerfen der Empfehlungen von Amazon Aurora
- Ändern abgelehnter Empfehlungen von Amazon Aurora in aktive Empfehlungen
- Empfehlungen aus der Referenz zu Amazon Aurora

Anzeigen von Amazon Aurora-Empfehlungen

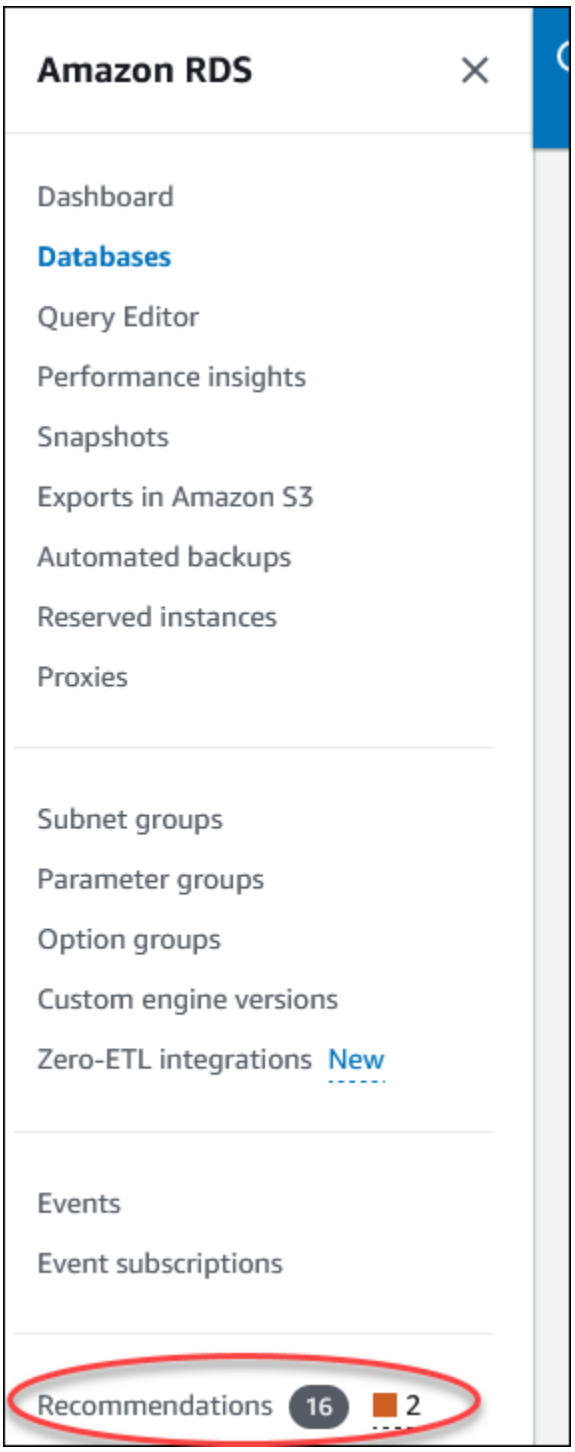
Mithilfe der Amazon-RDS-Konsole können Sie Amazon Aurora-Empfehlungen für Ihre Datenbankressourcen anzeigen. Bei einem DB-Cluster werden die Empfehlungen für den DB-Cluster und seine Instances angezeigt.

Konsole

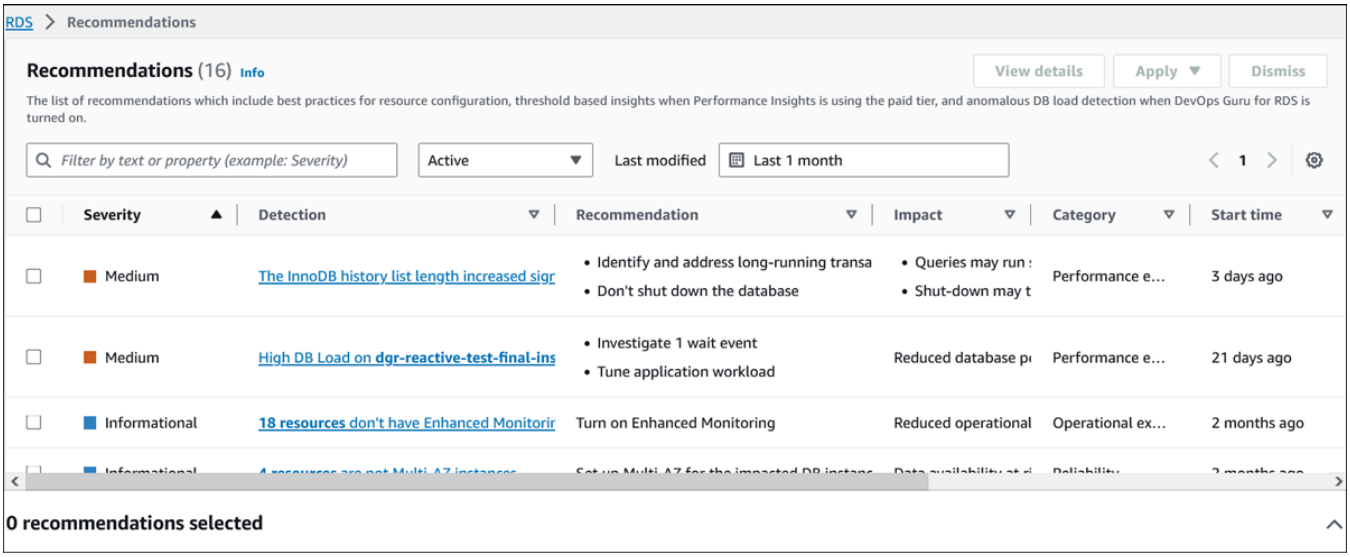
So zeigen Sie Amazon-Aurora-Empfehlungen an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Führen Sie im Navigationsbereich eine der folgenden Aktionen aus:

   - Wählen Sie Empfehlungen aus. Die Anzahl der aktiven Empfehlungen für Ihre Ressourcen und die Anzahl der Empfehlungen mit dem höchsten Schweregrad, die im letzten Monat generiert wurden, sind neben Empfehlungen sichtbar. Um die Anzahl der aktiven Empfehlungen für jeden Schweregrad zu ermitteln, wählen Sie die Zahl aus, die den höchsten Schweregrad anzeigt.

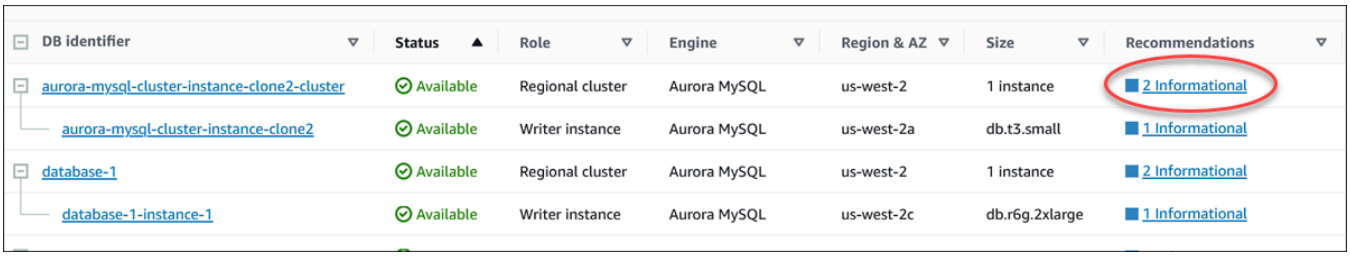


Standardmäßig wird auf der Seite Empfehlungen eine Liste der neuen Empfehlungen des letzten Monats angezeigt. Amazon Aurora gibt Empfehlungen für alle Ressourcen in Ihrem Konto und sortiert die Empfehlungen nach ihrem Schweregrad.

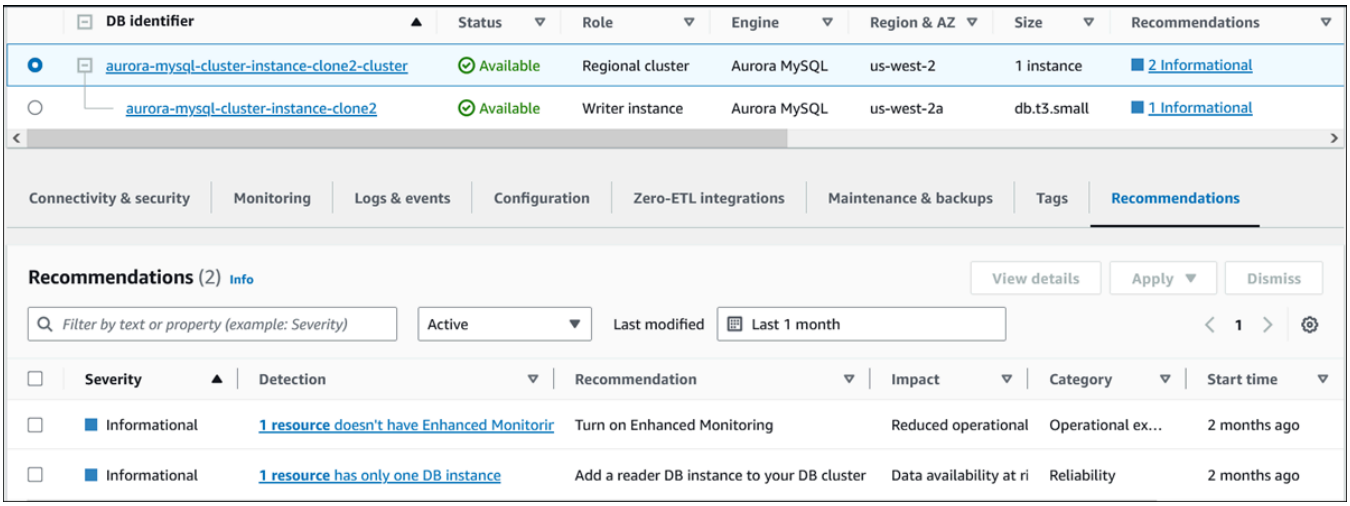


Sie können eine Empfehlung auswählen, um unten auf der Seite einen Abschnitt zu sehen, der die betroffenen Ressourcen und Einzelheiten zur Anwendung der Empfehlung enthält.

- Wählen Sie auf der Seite Datenbanken die Option Empfehlungen für eine Ressource aus.



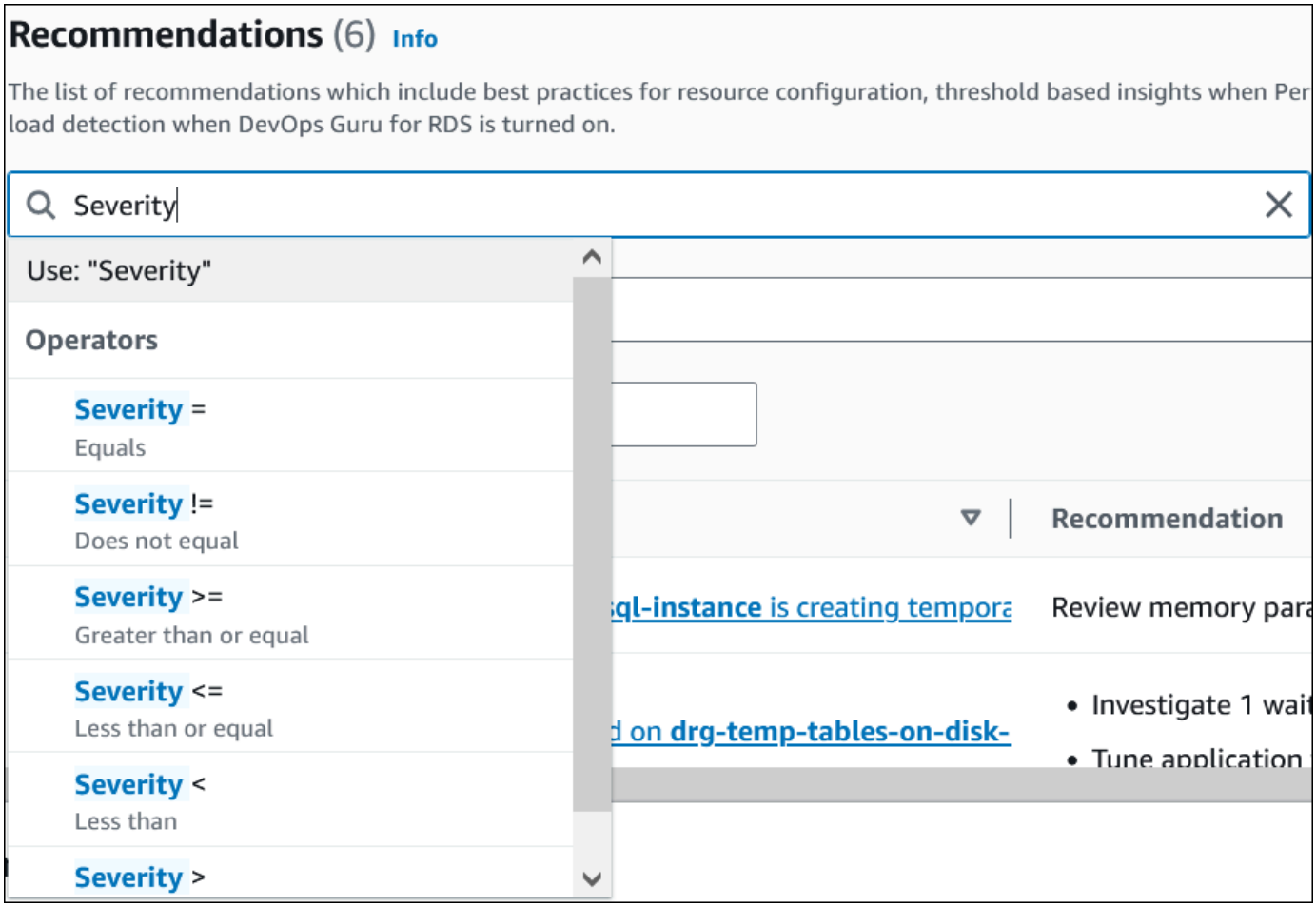
Die Registerkarte Empfehlungen zeigt die Empfehlungen und entsprechende Details für die ausgewählte Ressource an.



Die folgenden Informationen sind für die Empfehlungen verfügbar:

- Schweregrad – Das Ausmaß der Auswirkungen des Problems. Die Schweregrade lauten Hoch, Medium, Niedrig und Informativ.
- Erkennung – Die Anzahl der betroffenen Ressourcen und eine kurze Beschreibung des Problems. Wählen Sie diesen Link, um die Empfehlung und die Analysedetails anzuzeigen.
- Empfehlung – Eine kurze Beschreibung der empfohlenen Maßnahme, die angewendet werden soll.
- Auswirkung – Eine kurze Beschreibung der möglichen Auswirkung, wenn die Empfehlung nicht angewendet wird.
- Kategorie – Die Art der Empfehlung. Die Kategorien sind Leistungseffizienz, Sicherheit, Zuverlässigkeit, Kostenoptimierung, Operative Exzellenz und Nachhaltigkeit.
- Status – Der aktuelle Status der Empfehlung. Die möglichen Status sind Alle, Aktiv, Abgelehnt, Gelöst und Ausstehend.
- Startzeit – Die Zeit, zu der das Problem begann. Zum Beispiel Vor 18 Stunden.
- Letzte Änderung – Der Zeitpunkt, zu dem die Empfehlung aufgrund einer Änderung des Schweregrads zuletzt vom System aktualisiert wurde, oder der Zeitpunkt, zu dem Sie auf die Empfehlung reagiert haben. Zum Beispiel Vor 10 Stunden.
- Endzeit – Der Zeitpunkt, zu dem das Problem beendet wurde. Bei anhaltenden Problemen wird die Uhrzeit nicht angezeigt.
- Ressourcen-ID – Der Name einer oder mehrerer Ressourcen.

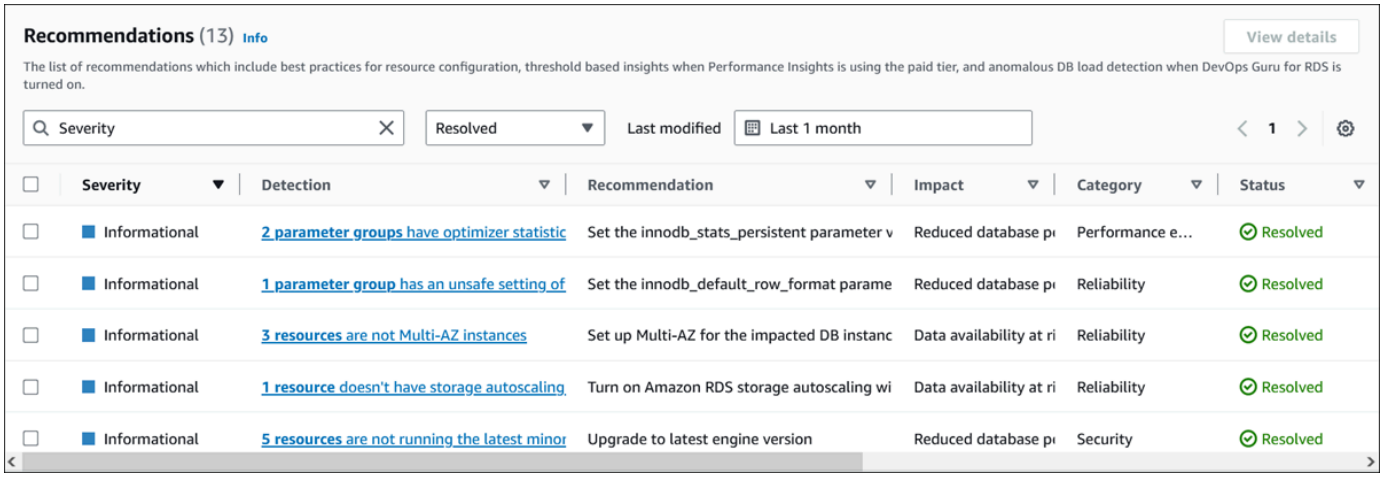
3. (Optional) Wählen Sie in dem Feld die Operatoren Schweregrad oder Kategorie aus, um die Liste der Empfehlungen zu filtern.



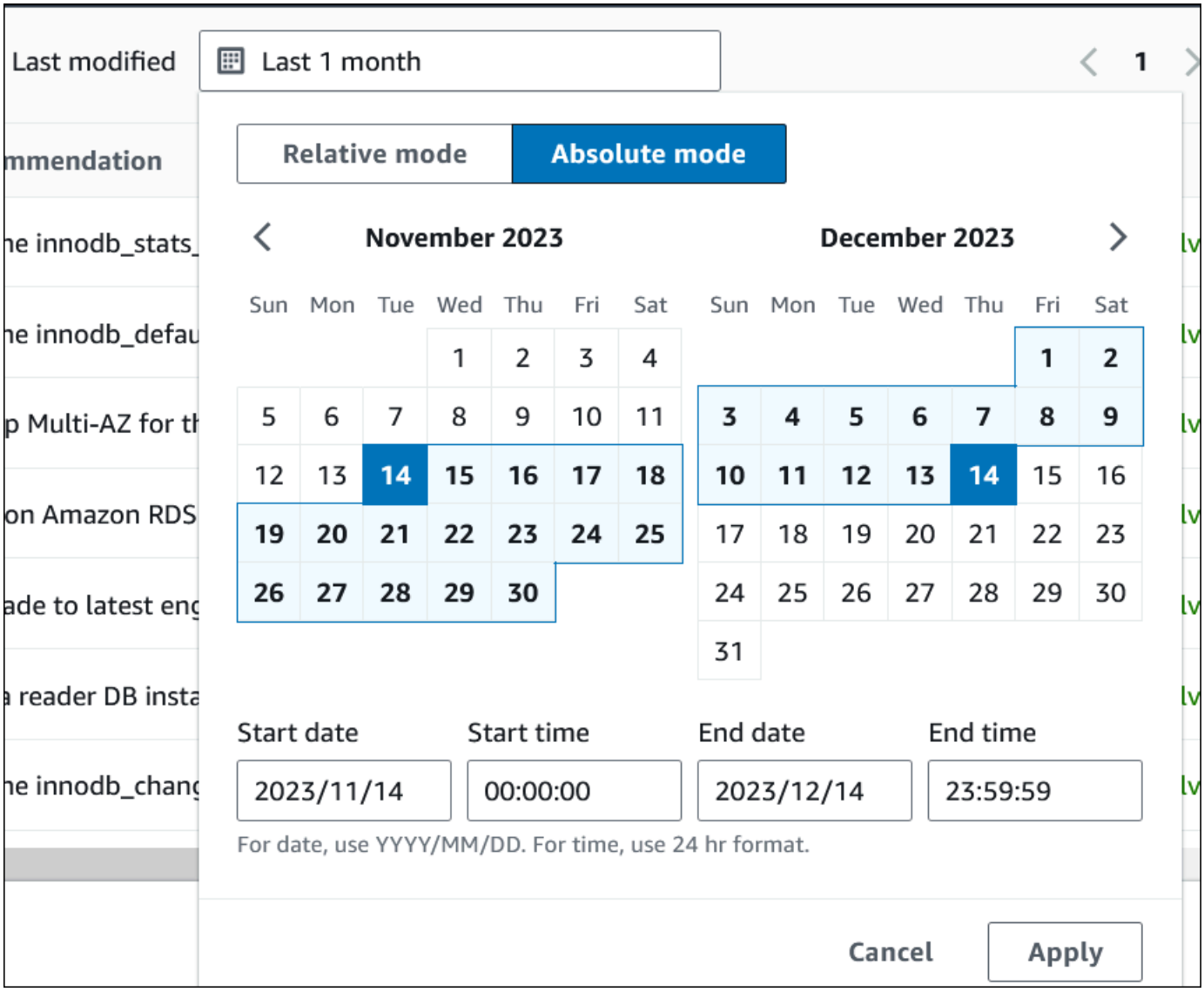
Die Empfehlungen für den ausgewählten Vorgang werden angezeigt.

4. (Optional) Sie können einen der folgenden Empfehlungsstatus auswählen:

   - Aktiv (Standard) – Zeigt die aktuellen Empfehlungen an, die Sie anwenden, für das nächste Wartungsfenster planen oder ablehnen können.
   - Alle – Zeigt alle Empfehlungen mit dem aktuellen Status an.
   - Abgelehnt – Zeigt die abgelehnten Empfehlungen an.
   - Gelöst – Zeigt die Empfehlungen an, die gelöst wurden.
   - Ausstehend – Zeigt die Empfehlungen an, deren empfohlene Maßnahmen derzeit ausgeführt werden oder für das nächste Wartungsfenster geplant sind.



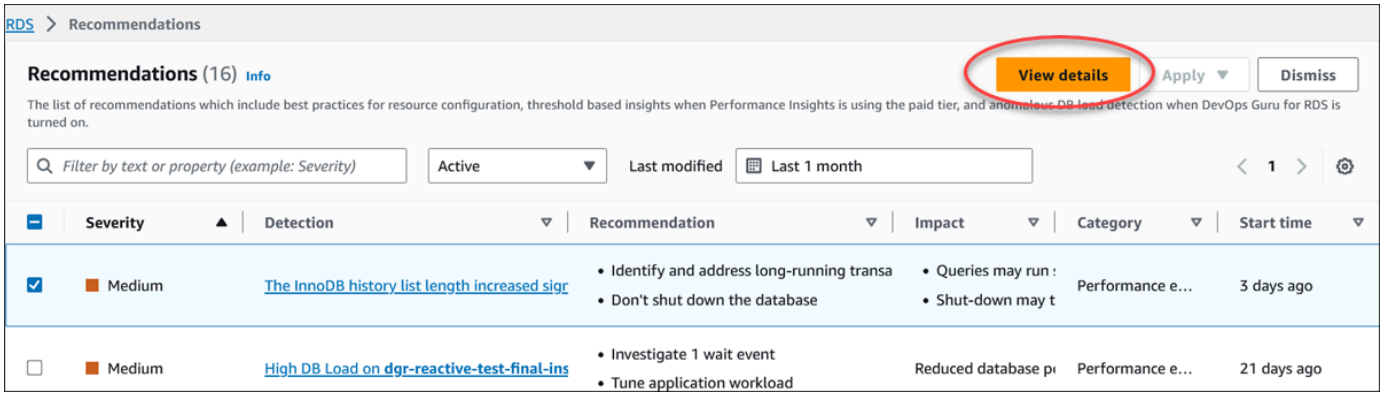
5. (Optional) Wählen Sie unter Letzte Änderung die Option Relativer Modus oder Absoluter Modus aus, um den Zeitraum zu ändern. Auf der Seite Empfehlungen werden die in diesem Zeitraum generierten Empfehlungen angezeigt. Der Standardzeitraum ist der letzte Monat. Unter Absoluter Modus können Sie den Zeitraum auswählen oder die Uhrzeit in die Felder Startdatum und Enddatum eingeben.



Die Empfehlungen für den festgelegten Zeitraum werden angezeigt.

Beachten Sie, dass Sie alle Empfehlungen für Ressourcen in Ihrem Konto anzeigen können, indem Sie den Bereich auf Alle festlegen.

6. (Optional) Wählen Sie auf der rechten Seite Einstellungen aus, um die anzuzeigenden Details anzupassen. Sie können ein Seitenformat auswählen, die Textzeilen umbrechen und die Spalten zulassen oder ausblenden.
7. (Optional) Wählen Sie eine Empfehlung und dann Details anzeigen aus.



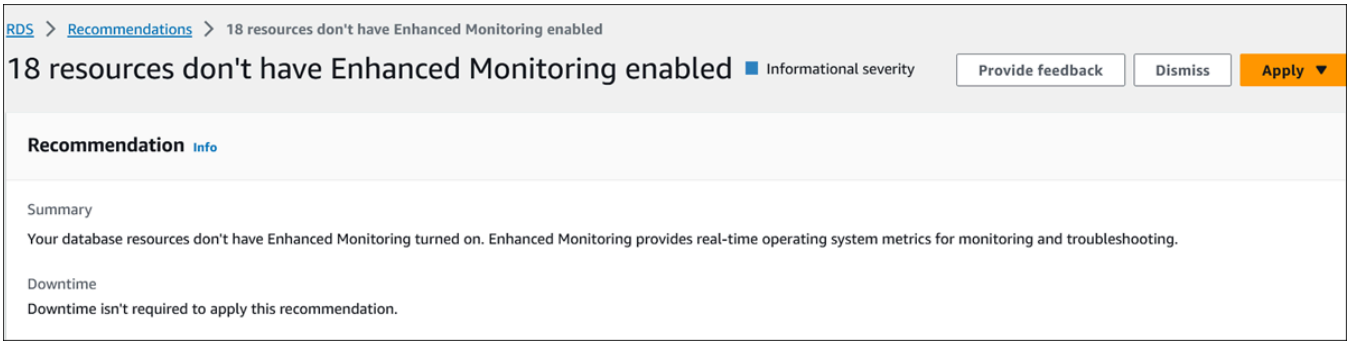
Die Detailseite zu den Empfehlungen wird angezeigt. Der Titel enthält die Gesamtzahl der Ressourcen mit dem erkannten Problem und dem Schweregrad.

Weitere Informationen zu den Komponenten auf der Detailseite für eine auf Anomalien basierende reaktive Empfehlung finden Sie unter Anzeigen reaktive Anomalien im Benutzerhandbuch zu Amazon DevOps Guru.

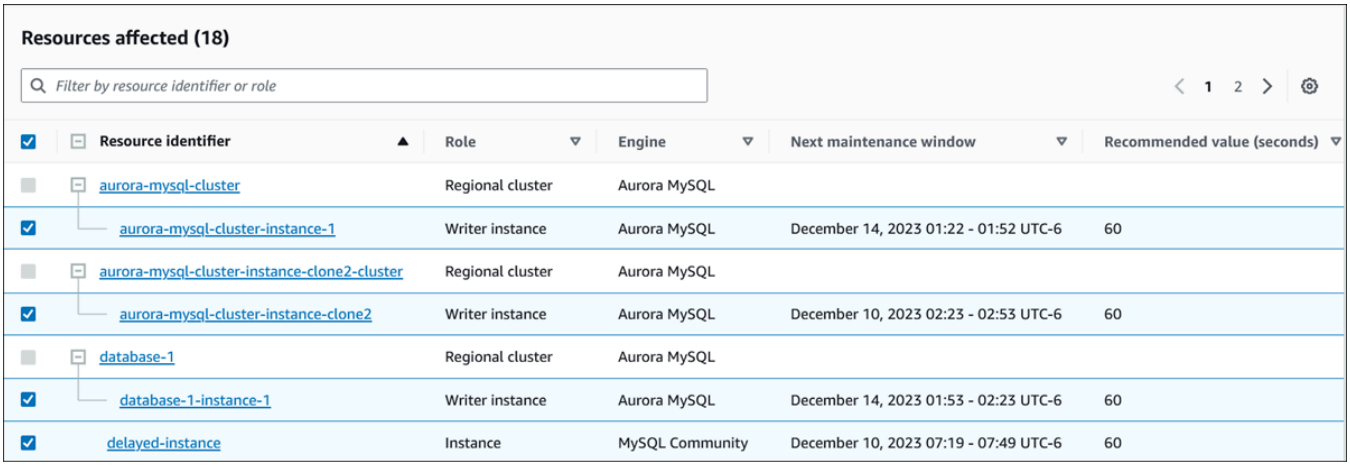
Informationen zu den Komponenten auf der Detailseite für eine auf Schwellenwerten basierende proaktive Empfehlung finden Sie unter Anzeigen proaktiver Empfehlungen von Performance Insights.

Bei den anderen automatisierten Empfehlungen werden auf der Detailseite der Empfehlungen die folgenden Komponenten angezeigt:

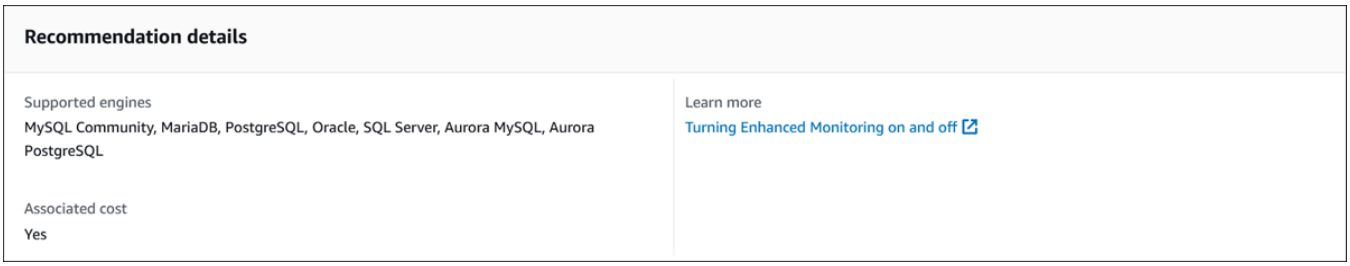
- Empfehlung – Eine Zusammenfassung der Empfehlung und Informationen dazu, ob Ausfallzeiten erforderlich sind, um die Empfehlung umzusetzen.



- Betroffene Ressourcen – Einzelheiten zu den betroffenen Ressourcen.



- Einzelheiten zur Empfehlung – Informationen zum unterstützten Modul, alle erforderlichen Kosten für die Umsetzung der Empfehlung und Link zur Dokumentation mit weiteren Informationen.



CLI

Um die Amazon-RDS-Empfehlungen der DB-Instances oder DB-Cluster anzuzeigen, verwenden Sie den folgenden Befehl in AWS CLI.

```
aws rds describe-db-recommendations
```

RDS-API

Um Amazon-RDS-Empfehlungen über die Amazon-RDS-API anzuzeigen, verwenden Sie den Vorgang DescribeDBRecommendations.

Anwenden von Empfehlungen von Amazon Aurora

Um Empfehlungen von Amazon Aurora mithilfe der Amazon-RDS-Konsole anzuwenden, wählen Sie auf der Detailseite eine konfigurationsbasierte Empfehlung oder eine betroffene Ressource aus. Wählen Sie dann aus, ob Sie die Empfehlung sofort anwenden oder für das nächste Wartungsfenster

planen möchten. Die Ressource muss möglicherweise neu gestartet werden, damit die Änderung wirksam wird. Für einige Empfehlungen zu DB-Parametergruppen müssen Sie möglicherweise die Ressourcen neu starten.

Die auf Schwellenwerten basierenden proaktiven oder anomaliebasierten reaktiven Empfehlungen verfügen nicht über die Option „Anwenden“ und müssen möglicherweise zusätzlich überprüft werden.

Konsole

So wenden Sie eine konfigurationsbasierte Empfehlung an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Führen Sie im Navigationsbereich eine der folgenden Aktionen aus:

   - Wählen Sie Empfehlungen aus.

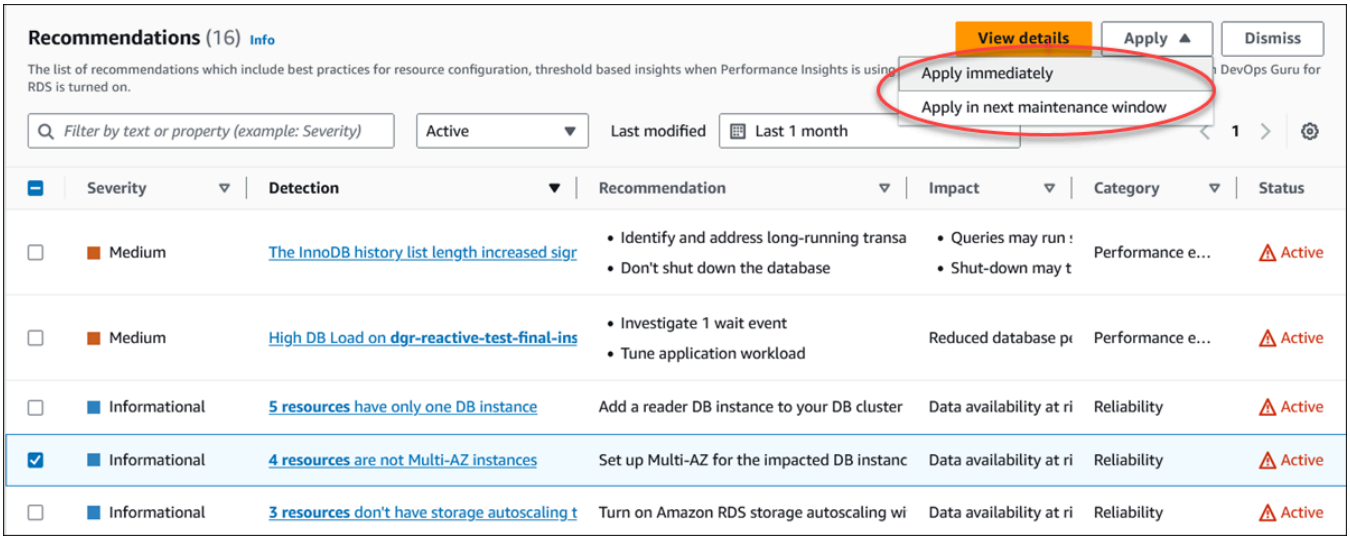
     Die Seite Empfehlungen mit der Liste aller Empfehlungen wird angezeigt.
   - Wählen Sie Datenbanken und dann Empfehlungen für eine Ressource auf der Datenbankseite aus.

     Die Details werden auf der Registerkarte Empfehlungen für die ausgewählte Empfehlung angezeigt.
   - Wählen Sie Erkennung für eine aktive Empfehlung auf der Seite Empfehlungen oder auf der Registerkarte Empfehlungen auf der Seite Datenbanken aus.

     Die Detailseite zu den Empfehlungen wird angezeigt.
3. Wählen Sie auf der Seite mit den Empfehlungsdetails eine Empfehlung oder eine oder mehrere betroffene Ressourcen aus und führen Sie einen der folgenden Schritte aus:

   - Wählen Sie Anwenden und anschließend Sofort anwenden, um die Empfehlung sofort anzuwenden.
   - Wählen Sie Anwenden und dann Im nächsten Wartungsfenster übernehmen aus, um die Änderungen im nächsten Wartungsfenster zu übernehmen.

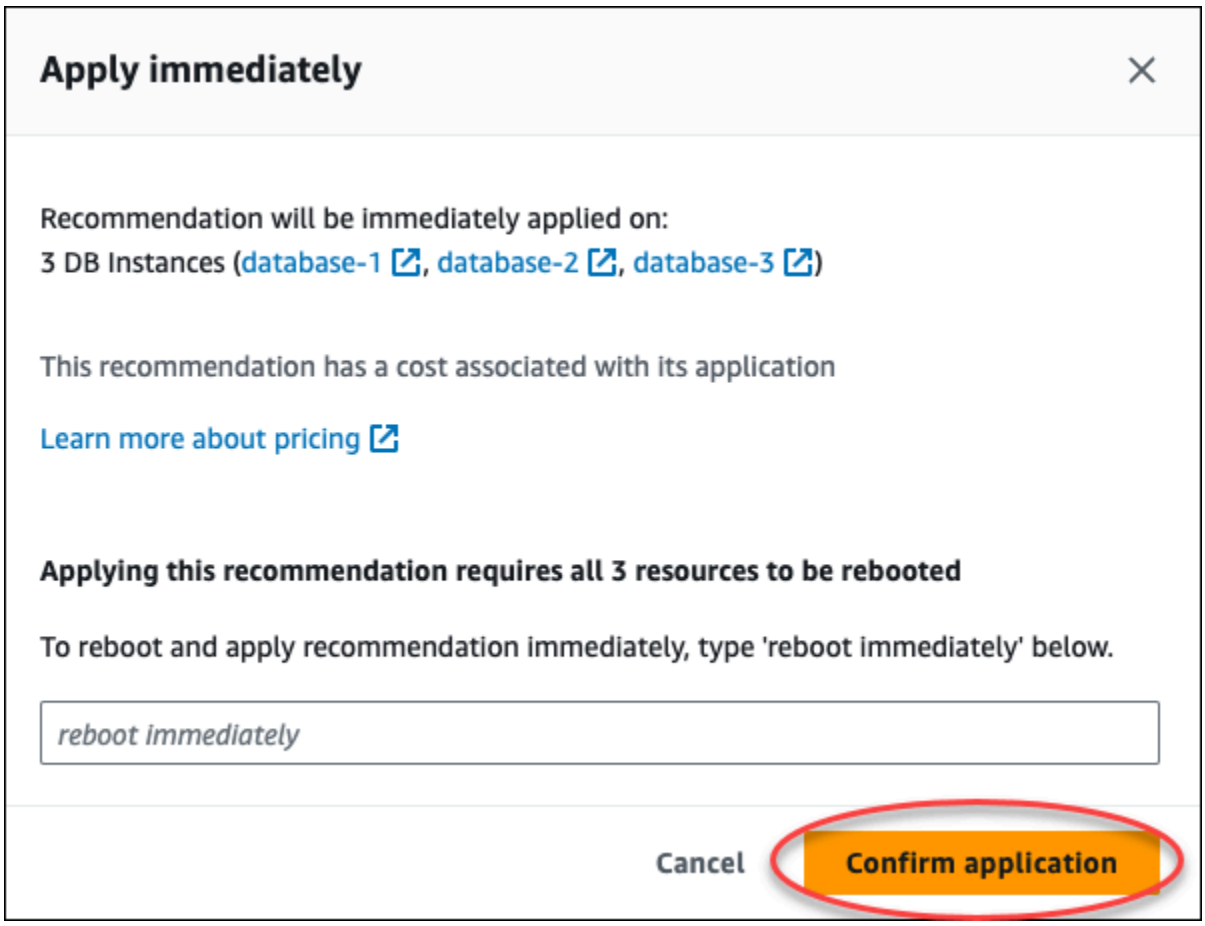
     Der Status der ausgewählten Empfehlung lautet bis zum nächsten Wartungsfenster „Ausstehend“.



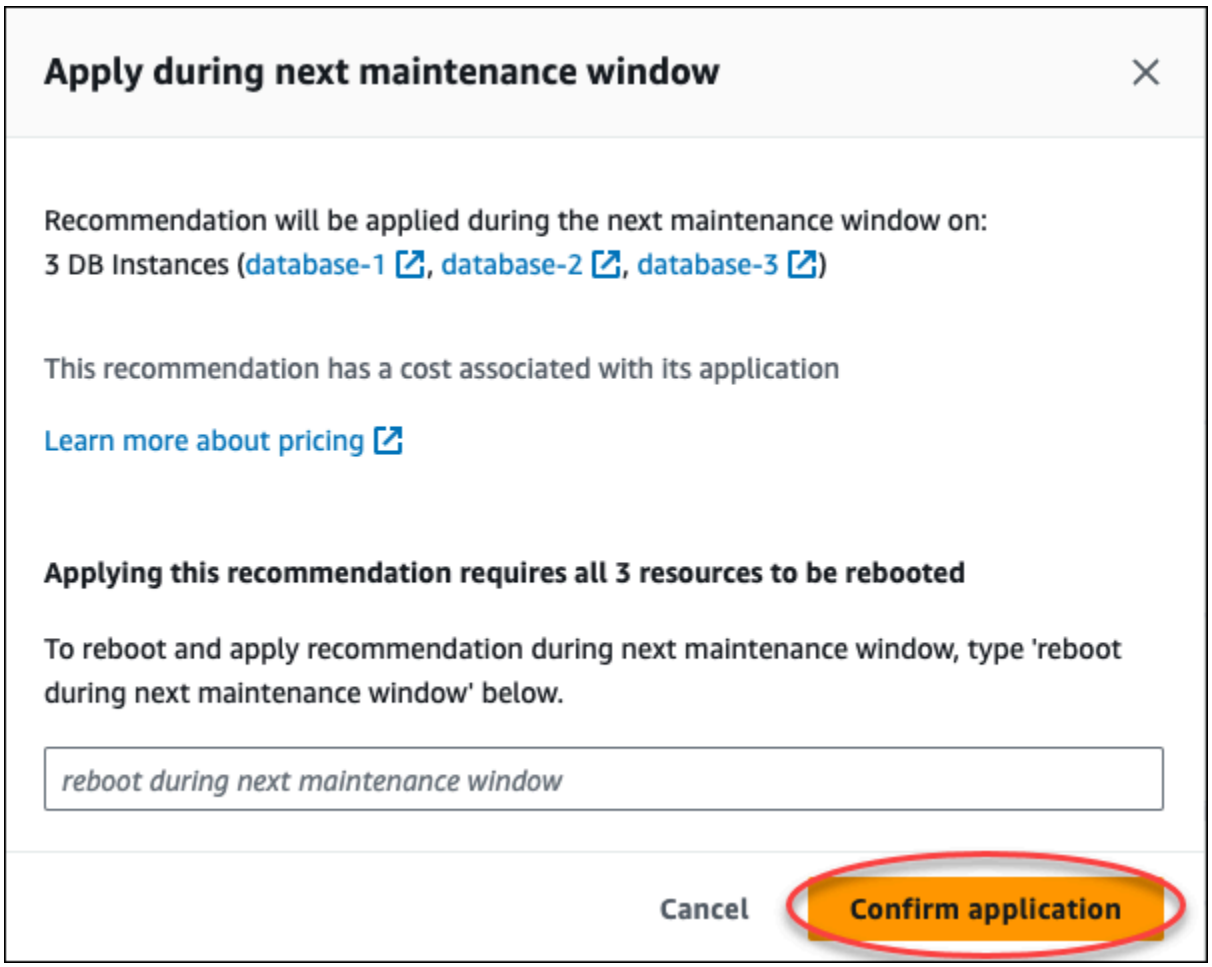
Ein Bestätigungsfenster wird angezeigt.

4. Wählen Sie Anwendung bestätigen, um die Empfehlung zu übernehmen. In diesem Fenster wird bestätigt, ob die Ressourcen automatisch oder manuell neu gestartet werden müssen, damit die Änderungen wirksam werden.

   Das folgende Beispiel zeigt das Bestätigungsfenster für die sofortige Anwendung der Empfehlung.



Das folgende Beispiel zeigt das Bestätigungsfenster, um die Anwendung der Empfehlung im nächsten Wartungsfenster zu planen.



In einem Banner wird eine Meldung angezeigt, wenn die Empfehlung erfolgreich anwendet wurde oder fehlgeschlagen ist.

Das folgende Beispiel zeigt das Banner mit der Erfolgsmeldung.

Das folgende Beispiel zeigt das Banner mit der Fehlermeldung.

RDS-API

So wenden Sie eine konfigurationsbasierte Aurora-Empfehlung mithilfe der Amazon-RDS-API an

1. Verwenden Sie den Vorgang DescribeDBRecommendations. Die `RecommendedActions` in der Ausgabe können eine oder mehrere empfohlene Aktionen enthalten.
2. Verwenden Sie das Objekt RecommendedAction für jede empfohlene Aktion aus Schritt 1. Die Ausgabe enthält `Operation` und `Parameters`.

   Das folgende Beispiel zeigt die Ausgabe mit einer empfohlenen Aktion.

```
   "RecommendedActions": [
       {
           "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
           "Title": "Turn on auto backup", // localized
           "Description": "Turn on auto backup for my-mysql-instance-1", //
localized
           "Operation": "ModifyDbInstance",
           "Parameters": [
               {
                   "Key": "DbInstanceIdentifier",
                   "Value": "my-mysql-instance-1"
               },
               {
                   "Key": "BackupRetentionPeriod",
                   "Value": "7"
               }
           ],
           "ApplyModes": ["immediately", "next-maintenance-window"],
           "Status": "applied"
       },
       ... // several others
   ],
```

3. Verwenden Sie `operation` für jede empfohlene Aktion aus der Ausgabe in Schritt 2 und geben Sie die Werte für die `Parameters` ein.
4. Nachdem der Vorgang in Schritt 2 erfolgreich war, verwenden Sie den Vorgang ModifyDBRecommendation, um den Empfehlungsstatus zu ändern.

Verwerfen der Empfehlungen von Amazon Aurora

Sie können eine oder mehrere Empfehlungen von Amazon Aurora mithilfe der Amazon-RDS-Konsole, der AWS CLI oder der Amazon-RDS-API verwerfen.

Konsole

So verwerfen Sie eine oder mehrere Empfehlungen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Führen Sie im Navigationsbereich eine der folgenden Aktionen aus:

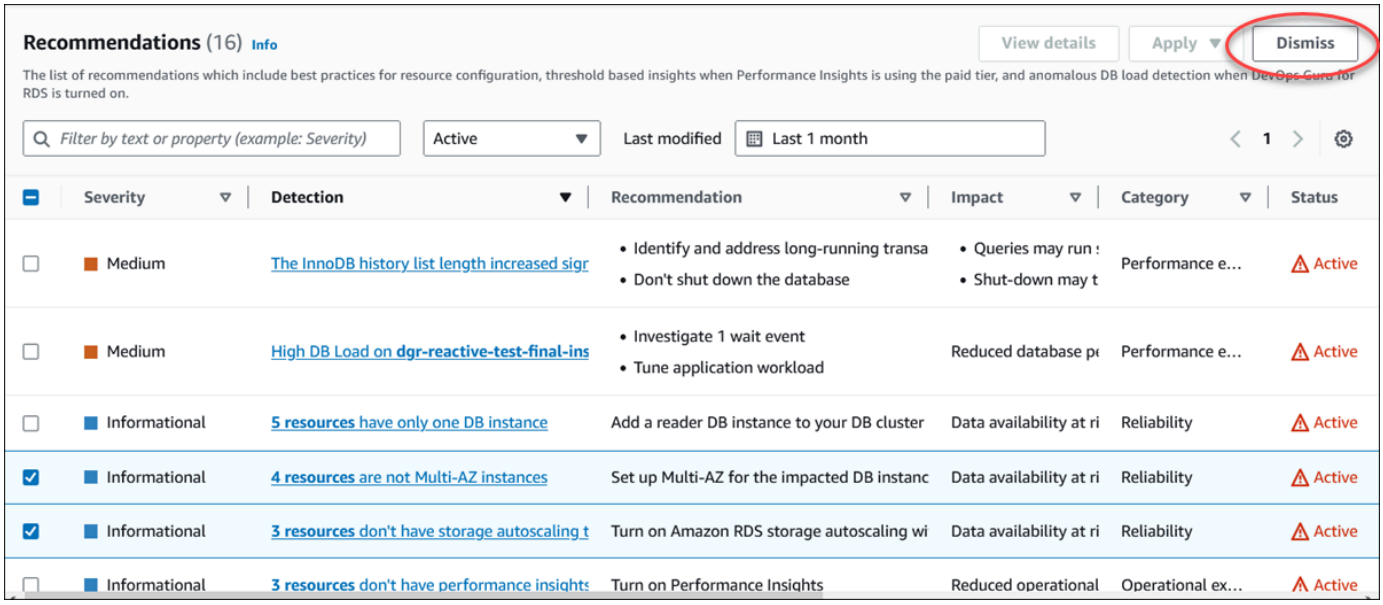
   - Wählen Sie Empfehlungen aus.

     Die Seite Empfehlungen mit der Liste aller Empfehlungen wird angezeigt.
   - Wählen Sie Datenbanken und dann Empfehlungen für eine Ressource auf der Datenbankseite aus.

     Die Details werden auf der Registerkarte Empfehlungen für die ausgewählte Empfehlung angezeigt.
   - Wählen Sie Erkennung für eine aktive Empfehlung auf der Seite Empfehlungen oder auf der Registerkarte Empfehlungen auf der Seite Datenbanken aus.

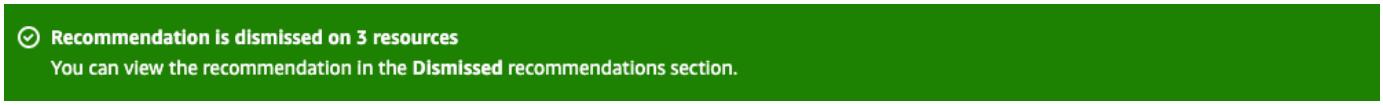
     Auf der Seite mit den Empfehlungsdetails wird die Liste der betroffenen Ressourcen angezeigt.
3. Wählen Sie auf der Seite mit den Empfehlungsdetails eine oder mehrere Empfehlungen oder eine oder mehrere betroffene Ressourcen aus und klicken Sie dann auf Ablehnen.

   Das folgende Beispiel zeigt die Seite mit den Empfehlungen, auf der mehrere aktive Empfehlungen zum Ablehnen ausgewählt wurden.



Ein Banner zeigt eine Meldung an, wenn die ausgewählten oder mehrere Empfehlungen abgelehnt werden.

Das folgende Beispiel zeigt das Banner mit der Erfolgsmeldung.



Das folgende Beispiel zeigt das Banner mit der Fehlermeldung.

CLI

So lehnen Sie eine Empfehlung von Aurora über die AWS CLI ab

1. Führen Sie den Befehl `aws rds describe-db-recommendations --filters "Name=status,Values=active"` aus.

   Die Ausgabe enthält eine Liste mit Empfehlungen mit dem Status `active`.

2. Suchen Sie die `recommendationId` für die Empfehlung aus Schritt 1, die abgelehnt werden soll.

3. Führen Sie den Befehl `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` mit der `recommendationId` aus Schritt 2 aus, um die Empfehlung abzulehnen.

RDS-API

Um eine Empfehlung von Aurora über die Amazon-RDS-API abzulehnen, verwenden Sie den Vorgang ModifyDBRecommendation.

Ändern abgelehnter Empfehlungen von Amazon Aurora in aktive Empfehlungen

Sie können eine oder mehrere abgelehnte Empfehlungen von Amazon Aurora mithilfe der Amazon-RDS-Konsole, der AWS CLI oder der Amazon-RDS-API in aktive Empfehlungen ändern.

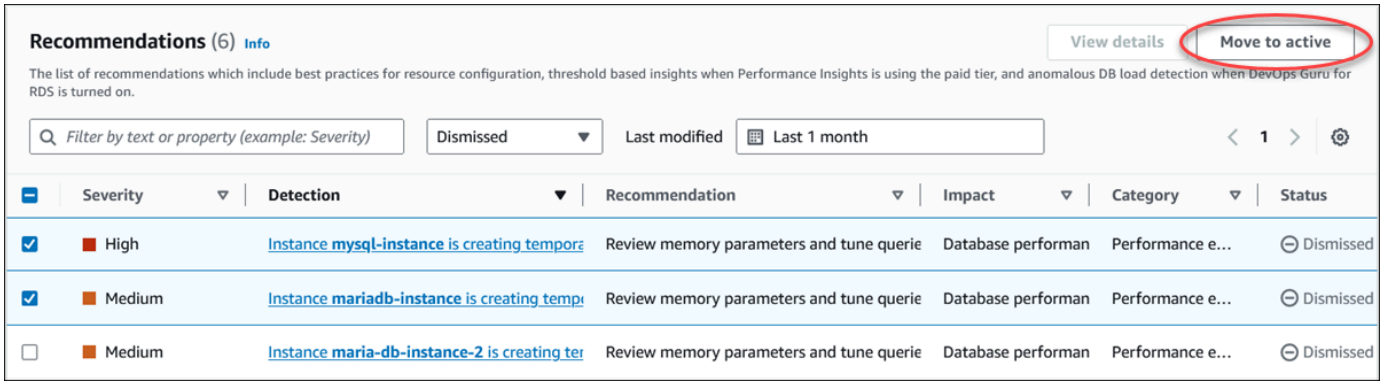
Konsole

So ändern Sie eine oder mehrere abgelehnte Empfehlungen in aktive Empfehlungen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Führen Sie im Navigationsbereich eine der folgenden Aktionen aus:
   - Wählen Sie Empfehlungen aus.

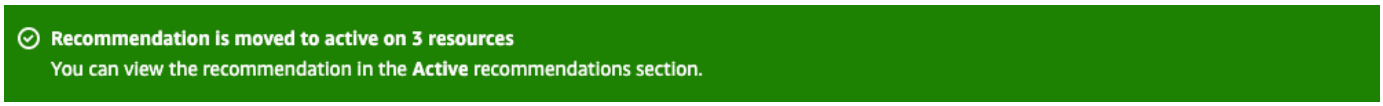
     Auf der Seite Empfehlungen wird eine nach Schweregrad sortierte Liste mit Empfehlungen für alle Ressourcen in Ihrem Konto angezeigt.
   - Wählen Sie Datenbanken und dann Empfehlungen für eine Ressource auf der Datenbankseite aus.

     Die Registerkarte Empfehlungen zeigt die Empfehlungen und entsprechende Details für die ausgewählte Ressource an.
3. Wählen Sie eine oder mehrere abgelehnte Empfehlungen aus der Liste und dann die Option In aktiv ändern aus.



Ein Banner zeigt eine Erfolgs- oder Fehlermeldung an, wenn die ausgewählten Empfehlungen vom Status „Abgelehnt“ in den Status „Aktiv“ geändert werden.

Das folgende Beispiel zeigt das Banner mit der Erfolgsmeldung.



Das folgende Beispiel zeigt das Banner mit der Fehlermeldung.

CLI

Ändern einer abgelehnten Empfehlung von Aurora in eine aktive Empfehlung über die AWS CLI

1. Führen Sie den Befehl `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"` aus.

   Die Ausgabe enthält eine Liste mit Empfehlungen mit dem Status `dismissed`.

2. Suchen Sie die `recommendationId` für die Empfehlung aus Schritt 1, deren Status Sie ändern möchten.

3. Führen Sie den Befehl `>aws rds modify-db-recommendation --status active --recommendationId <ID>` mit der `recommendationId` aus Schritt 2 aus, um eine Änderung in eine aktive Empfehlung vorzunehmen.

RDS-API

Um eine abgelehnte Empfehlung von Aurora über die Amazon-RDS-API in eine aktive Empfehlung zu ändern, verwenden Sie den Vorgang ModifyDBRecommendation.

Empfehlungen aus der Referenz zu Amazon Aurora

Amazon Aurora erstellt Empfehlungen für eine Ressource, wenn die Ressource erstellt oder geändert wird. In der folgenden Tabelle finden Sie Beispiele für Empfehlungen von Amazon Aurora.

Type (Typ)	Beschreibung	Empfehlung	Ausfall: it erforde ich	Zusätzliche Informati onen
Automatis ierte Ressource nsicherungen sind deaktivie rt	Automatisierte Sicherungen sind für Ihre DB-Instan ces nicht aktiviert . Automatisierte Backups werden empfohlen, da sie die point-in-time Wiederherstellung Ihrer DB-Instances ermöglichen.	Aktivieren Sie automatisierte Backups mit einer Aufbewahrungsfrist von bis zu 14 Tagen.	Ja	Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters Demystifying Amazon RDS backup storage costs im AWS - Datenbank-Blog
Ein Upgrade der Engine-Unterversio n ist erforderl ich	Ihre Datenbank ressourcen führen nicht die neueste DB-Engine-Unterver sion aus. Die neueste Unterversion enthält die neuesten Sicherheitskorrekt uren sowie andere Verbesserungen.	Führen Sie ein Upgrade auf die neueste Engine-Ve rsion durch.	Ja	Warten eines Amazon Aurora-DB-Clusters

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Enhanced Monitoring ist deaktiviert	Für Ihre Datenbankressourcen ist Enhanced Monitoring nicht aktiviert. „Erweiterte Überwachung“ bietet Echtzeit-Betriebssystemmetriken für die Überwachung und Fehlerbehebung.	Aktivieren Sie die Option „Enhanced Monitoring“.	Nein	Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“ (Erweiterte Überwachung)

Type (Typ)	Beschreibung	Empfehlung	Ausfall: it erforde ich	Zusätzliche Informati onen
Die Speicherv erschlüss elung ist deaktiviert	Amazon RDS unterstützt Verschlüs selung im Ruhezusta nd für alle Datenbank -Engines mithilfe der Schlüssel, die Sie im AWS Key Management Service (AWS KMS) verwalten . Auf einer aktiven DB-Instance mit Amazon-RDS-Verschl üsselung werden die Daten, die im Ruhezustand im Speicher gespeiche rt werden, ähnlich wie bei automatis ierten Sicherungen, Lesereplikaten und Snapshots. Wenn die Verschlüs selung beim Erstellen eines Aurora-DB-Clusters nicht aktiviert ist, müssen Sie einen entschlüsselten Snapshot in einem verschlüsselten DB-	Aktivieren Sie die Verschlüsselung der Daten im Ruhezusta nd für Ihren DB-Cluste r.	Ja	Sicherheit in Amazon Aurora

Type (Typ)	Beschreibung	Empfehlung	Ausfall: it erforde ich	Zusätzliche Informati onen
	Cluster wiederher stellen.			
DB-Cluster mit allen DB-Instances in derselben Availability Zone	Die DB-Cluster befinden sich derzeit in einer einzigen Availability Zone. Verwenden Sie mehrere Availabil ity Zones, um die Verfügbarkeit zu verbessern.	Fügen Sie die DB-Instances mehreren Availability Zones in Ihrem DB-Cluster hinzu.	Nein	Hohe Verfügbarkeit für Amazon Aurora
DB-Instan ces in den Clustern mit heterogen en Instance-Größen	Wir empfehlen, dass Sie für alle DB-Instan ces in Ihrem DB-Cluster dieselbe Klasse und Größe für DB-Instances verwenden.	Verwenden Sie für alle DB-Instances in Ihrem DB-Cluster dieselbe Klasse und Größe für DB-Instan ces.	Ja	Replikation mit Amazon Aurora
DB-Instan ces in den Clustern mit heterogen en Instance-Klassen	Wir empfehlen, dass Sie für alle DB-Instan ces in Ihrem DB–Cluster dieselbe Klasse und Größe für DB-Instances verwenden.	Verwenden Sie für alle DB-Instances in Ihrem DB-Cluster dieselbe Klasse und Größe für DB-Instan ces.	Ja	Replikation mit Amazon Aurora

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
DB-Instances in den Clustern mit heterogenen Parametergruppen	Wir empfehlen, dass alle DB-Instances im DB-Cluster dieselbe DB-Parametergruppe verwenden.	Ordnen Sie die DB-Instance der DB-Parametergruppe zu, die mit der Writer-Instance in Ihrem DB-Cluster verknüpft ist.	Nein	Parametergruppen für Amazon Aurora
DB-Cluster von Amazon-RDS verfügen über eine DB-Instance	Fügen Sie Ihrem DB-Cluster mindestens eine weitere DB-Instance hinzu, um die Verfügbarkeit und Leistung zu verbessern.	Fügen Sie Ihrem DB-Cluster eine Reader-DB-Instance hinzu.	Nein	Hohe Verfügbarkeit für Amazon Aurora
Performance Insights ist deaktiviert	Performance Insights überwacht die DB-Instance-Last, damit Sie Probleme mit der Datenbankleistung analysieren und Fehler beheben können. Wir empfehlen Ihnen, Performance Insights zu aktivieren.	Aktivieren Sie Performance Insights.	Nein	Überwachung mit Performance Insights auf

Type (Typ)	Beschreibung	Empfehlung	Ausfall: it erforde ich	Zusätzliche Informati onen
Aktualisi erung der Hauptvers ionen von RDS-Resso urcen ist erforderlich	Datenbanken mit der aktuellen Hauptvers ion für die DB-Engine werden nicht unterstüt zt. Wir empfehlen Ihnen ein Upgrade auf die neueste Hauptversion, die neue Funktionen und Verbesserungen enthält.	Führen Sie ein Upgrade auf die neueste Hauptvers ion für die DB-Engine durch.	Ja	Amazon-Aurora-Upda tes Eine blue/green Bereitstellung in — Amazon Aurora erstellen
Maximale Volume-Gr öße des DB-Clusters	Neuere Engine-Ve rsionen unterstützen größere Speicherv olumen für Ihren DB-Cluster.	Wir empfehlen, dass Sie die Engine-Ve rsion Ihres DB-Cluste rs auf die neueste Version aktualisieren, um von der erhöhten Speicherkapazität zu profitieren.	Ja	Amazon Aurora-Gr ößenbeschränkungen

Type (Typ)	Beschreibung	Empfehlung	Ausfall: it erforde ich	Zusätzliche Informati onen
DB-Cluste r mit allen Reader-In stances in derselben Availability Zone	Availability Zones (AZs) sind Standorte , die sich voneinander unterscheiden, um bei Ausfällen innerhalb der einzelnen AWS Regionen eine Isolierung zu gewährleisten. Wir empfehlen, dass Sie die primäre Instance und die Reader-In stances in Ihrem DB-Cluster auf mehrere verteilen AZs , um die Verfügbarkeit Ihres DB-Clusters zu verbessern. Sie können einen Multi-AZ-Cluster mithilfe der AWS Managemen t Console, der AWS CLI oder der Amazon RDS-API erstellen, wenn Sie den Cluster erstellen. Sie können auch den vorhanden en Aurora-Cluster in einen Multi-AZ-Cluster konvertie ren, indem Sie eine	Alle Reader-Instances Ihres DB-Cluste rs befinden sich in derselben Availability Zone. Wir empfehlen Ihnen, die Reader-In stances über mehrere Availability Zones zu verteilen. Die Verteilung erhöht die Verfügbarkeit und verbessert die Reaktionszeit, indem die Netzwerklatenz zwischen Clients und Datenbank reduziert wird.	Nein	Hohe Verfügbarkeit für Amazon Aurora

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
	neue Reader-Instance hinzufügen und eine andere Availability Zone angeben.			
DB-Speicherparameter weichen vom Standard ab	Die Speicherparameter der DB-Instances unterscheiden sich erheblich von den Standardwerten. Diese Einstellungen können die Leistung beeinträchtigen und Fehler verursachen. Wir empfehlen, die benutzerdefinierten Speicherparameter für die DB-Instance in der DB-Parametergruppe auf ihre Standardwerte zurückzusetzen.	Setzen Sie die Speicherparameter auf die Standardwerte zurück.	Nein	Parametergruppen für Amazon Aurora

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Abfrage-Cache-Parameter ist aktiviert	Wenn Änderungen erfordern, dass Ihr Abfrage-Cache geleert wird, scheint Ihre DB-Instance zum Stillstand zu kommen. Die meisten Workloads profitieren nicht von einem Abfrage-Cache. Der Abfrage-Cache wurde aus MySQL 8.0 und höheren Versionen entfernt. Wir empfehlen, den Parameter „query_cache_type“ auf „0“ festzulegen.	Legen Sie den Wert des Parameters `query_cache_type` in Ihren DB-Parametergruppen auf `0` fest.	Ja	Parametergruppen für Amazon Aurora

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Parameter `log_output` ist auf „table“ festgelegt	Wenn `log_output` auf `TABLE` festgelegt ist, wird mehr Speicherplatz verwendet, als wenn `log_output` auf `FILE` festgelegt ist. Es wird empfohlen, den Parameter auf `FILE` zu setzen, um zu vermeiden, dass das Größenlimit des Speichers überschritten wird. Legen Sie den Wert in MySQL 8.4 und höheren Versionen standardmäßig auf `FILE` fest.	Legen Sie den Wert des Parameters `log_output` in Ihren DB-Parametergruppen auf `FILE` fest.	Nein	Aurora-MySQL-Datenbank-Protokolldateien

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Parameter `autovacuum` ist deaktiviert	Der Selbstbereinigungsparameter ist für Ihre DB-Cluster deaktiviert. Das Deaktivieren der Selbstbereinigung führt zu einer Überlastung von Tabellen und Indizes und beeinträchtigt die Leistung. Wir empfehlen Ihnen, die Selbstbereinigung in Ihren DB-Parametergruppen zu aktivieren.	Aktivieren Sie den Parameter für die Selbstbereinigung in Ihren DB-Cluster-Parametergruppen.	Nein	Grundlegendes zum Thema Autovacuum in Amazon RDS for PostgreSQL PostgreSQL-Umgebungen im Datenbank-Blog AWS

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Der Parameter `synchronous_commit` ist deaktiviert.	Wenn der Parameter `synchronous_commit` deaktiviert ist, können Daten bei einem Datenbankabsturz verloren gehen. Die Haltbarkeit der Datenbank ist gefährdet. Wir empfehlen Ihnen, den `synchronous_commit` -Parameter zu aktivieren.	Legen Sie den Parameter `synchronous_commit` in Ihren DB-Parametergruppen.	Ja	Amazon Aurora PostgreSQL-Parameter: Replikation, Sicherheit und Protokollierung im AWS Datenbank-Blog
Der Parameter `track_counts` ist deaktiviert.	Wenn der Parameter `track_counts` deaktiviert ist, erfasst die Datenbank die Datenbankaktivitätsstatistiken nicht. Die Selbstbereinigung erfordert, dass diese Statistiken korrekt funktionieren. Wir empfehlen , den Parameter `track_counts` auf 1 zu setzen.	Legen Sie den Parameter `track_counts` auf 1 fest.	Nein	Laufzeitstatistiken für PostgreSQL

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Der Parameter `enable_indexonlyscan` ist deaktiviert.	Der Abfrageplaner oder -optimierer kann den reinen Index-Scan-Plantyp nicht verwenden, wenn er deaktiviert ist. Wir empfehlen, den Wert des Parameters `enable_indexonlyscan` auf 1 zu setzen.	Legen Sie den Wert des Parameters `enable_indexonlyscan` auf 1 fest.	Nein	Konfiguration von Planermethoden für PostgreSQL
Der Parameter `enable_indexscan` ist deaktiviert.	Der Abfrageplaner oder -optimierer kann den Index-Scan-Plantyp nicht verwenden, wenn er deaktiviert ist. Wir empfehlen Ihnen, den Wert `enable_indexscan` auf 1 festzulegen.	Legen Sie den Wert des Parameters `enable_indexscan` auf 1 fest.	Nein	Konfiguration von Planermethoden für PostgreSQL

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Der Parameter `innodb_flush_log_at_trx` ist deaktiviert.	Der Wert des Parameters `innodb_flush_log_at_trx` Ihrer DB-Instance ist kein sicherer Wert. Dieser Parameter steuert die Persistenz von Commit-Operationen auf der Festplatte. Wir empfehlen, den Parameter `innodb_flush_log_at_trx` auf 1 zu setzen.	Legen Sie den Wert des Parameters `innodb_flush_log_at_trx` auf 1 fest.	Nein	Konfigurieren, wie oft der Protokollpuffer geleert wird

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Der Parameter `innodb_stats_persistent` ist deaktiviert.	Ihre DB-Instance ist nicht dafür konfiguriert, die InnoDB-Statistiken auf der Festplatte zu speichern. Wenn die Statistiken nicht gespeichert werden, werden sie bei jedem Neustart der Instance und jedem Zugriff auf die Tabelle neu berechnet. Dies führt zu Abweichungen im Abfrageausführungsplan. Sie können den Wert dieses globalen Parameters auf Tabellenebene ändern. Wir empfehlen, den Wert des Parameters `innodb_stats_persistent` auf `ON` zu setzen.	Legen Sie den Wert des Parameters `innodb_stats_persistent` auf `ON` fest.	Nein	Parametergruppen für Amazon Aurora

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Der Wert des Parameters `innodb_open_files` ist niedrig	Der Parameter `innodb_open_files` steuert die Anzahl der Dateien, die InnoDB gleichzeitig öffnen kann. InnoDB öffnet alle Protokoll- und System-Tablespace-Dateien, wenn mysqld ausgeführt wird. Ihre DB-Instance hat einen niedrigen Wert für die maximale Anzahl von Dateien, die InnoDB gleichzeitig öffnen kann. Wir empfehlen, den Parameter `innodb_open_files` auf einen Mindestwert von 65 zu setzen.	Legen Sie den Wert des Parameters `innodb_open_files` mindestens auf 65 fest.	Ja	InnoDB open files für MySQL

Type (Typ)	Beschreibung	Empfehlung	Ausfall: it erforde ich	Zusätzliche Informati onen
Der Wert des Parameters `max_user_ connectio ns` ist niedrig.	Ihre DB-Instance hat einen niedrigen Wert für die maximale Anzahl gleichzeitiger Verbindungen für jedes Datenbank konto. Wir empfehlen , den Parameter `max_user_ connections` auf eine Zahl größer als 5 festzulegen.	Erhöhen Sie den Wert des Parameter s `max_user_ connections` auf eine Zahl größer als 5.	Ja	Setting Account Resource Limits for MySQL
Leserepli kate sind im beschreib baren Modus geöffnet	Ihre DB-Instance verfügt über ein Lesereplikat im beschreibbaren Modus, das Aktualisi erungen von Clients ermöglicht. Es wird empfohlen , den Parameter `read_only` auf `TrueIfRep lica` festzuleg en, damit sich die Lesereplikate nicht im beschreibbaren Modus befinden.	Legen Sie den Wert des Parameters `read_only` auf `TrueIfReplica` fest.	Nein	Parametergruppen für Amazon Aurora

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Die Einstellung des Parameters `innodb_default_row_format` ist unsicher	Ihre DB-Instance stößt auf ein bekanntes Problem: Eine Tabelle, die in einer MySQL-Version niedriger als 8.0.26 erstellt wurde und bei der `row_format` auf `COMPACT` oder `REDUNDANT` festgelegt wurde, ist nicht mehr zugänglich und kann nicht wiederhergestellt werden, wenn der Index 767 Byte überschreitet. Wir empfehlen, den Wert des Parameters `innodb_default_row_format` auf `DYNAMIC` zu setzen.	Legen Sie den Wert des Parameters `innodb_default_row_format` auf `DYNAMIC` fest.	Nein	Änderungen in MySQL 8.0.26

Type (Typ)	Beschreibung	Empfehlung	Ausfall: it erforde ich	Zusätzliche Informati onen
Parameter `general_l ogging` ist aktiviert	Die allgemeine Protokollierung ist für Ihre DB-Instan ce aktiviert. Diese Einstellung ist bei der Behebung von Datenbankproblemen hilfreich. Die Aktivieru ng der allgemeinen Protokollierung erhöht jedoch die Anzahl der I/O Vorgänge und den zugewiese nen Speicherplatz, was zu Konflikten und Leistungseinbußen führen kann. Überprüfen Sie Ihre Anforderungen für die allgemeine Protokoll ierungsnutzung. Wir empfehlen, den Wert des Parameters `general_logging` auf `0` zu setzen.	Überprüfen Sie Ihre Anforderungen für die allgemeine Protokollierungsnu tzung. Wenn dies nicht verpflichtend ist, empfehlen wir Ihnen, den Wert des Parameters `general_logging` auf `0` festzulegen.	Nein	Überblick über Aurora-MySQL-Datenbankprotokolle

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Die für Lese-Workloads des DB-Clusters bereitgestellte Kapazität ist unzureichend	Wir empfehlen Ihnen, Ihrem DB-Cluster eine Reader-DB-Instance mit derselben Instance-Klasse und -Größe hinzuzufügen wie die Writer-DB-Instance im Cluster. Die aktuelle Konfiguration umfasst eine DB-Instance mit einer kontinuierlich hohen Datenbanklast, die hauptsächlich durch Lesevorgänge verursacht wird. Verteilen Sie diese Vorgänge, indem Sie dem Cluster eine weitere DB-Instance hinzufügen und den Lese-Workload an den schreibgeschützten Endpunkt des DB-Clusters weiterleiten.	Fügen Sie dem Cluster eine Reader-DB-Instance hinzu.	Nein	Hinzufügen von Aurora-Replicas zu einem DB-Cluster Verwalten von Performance und Skalierung für einen Aurora-DB-Cluster Amazon-RDS-Preise

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Die RDS-Instance ist in Bezug auf die Systemspeicherkapazität unzureichend	Wir empfehlen Ihnen, Ihre Abfragen so anzupassen, dass sie weniger Speicher verwenden, oder einen DB-Instance-Typ zu nutzen, dem mehr Speicher zugewiesen ist. Wenn der Instance nur noch wenig Arbeitsspeicher zur Verfügung steht, wird die Datenbankleistung beeinträchtigt.	Verwenden Sie eine DB-Instance mit höherer Speicherkapazität.	Ja	Vertikale und horizontale Skalierung Ihrer Amazon RDS-Instance im AWS Datenbank-Blog Amazon-RDS-Instance-Typen Amazon-RDS-Preise

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
Die RDS-Instance ist in Bezug auf die System-CPU-Kapazität unzureichend	Wir empfehlen Ihnen, Ihre Abfragen so zu optimieren, dass sie weniger CPU verbrauchen, oder Ihre DB-Instance so zu modifizieren, dass sie eine DB-Instance-Klasse mit einer höheren zugewiesenen V verwendet CPUs. Die Datenbankleistung kann sinken, wenn die CPU-Auslastung einer DB-Instance knapp wird.	Verwenden Sie eine DB-Instance mit höherer CPU-Kapazität.	Ja	Vertikale und horizontale Skalierung Ihrer Amazon RDS-Instance im AWS Datenbank-Blog Amazon-RDS-Instance-Typen Amazon-RDS-Preise

Type (Typ)	Beschreibung	Empfehlung	Ausfallzeit erforderlich	Zusätzliche Informationen
RDS-Ressourcen nutzen das Verbindungspooling nicht korrekt	Wir empfehlen Ihnen, Amazon-RDS-Proxy zu aktivieren, um vorhandene Datenbankverbindungen effizient zu bündeln und freizugeben. Wenn Sie bereits einen Proxy für Ihre Datenbank verwenden, konfigurieren Sie ihn korrekt, um das Verbindungspooling und den Lastenausgleich zwischen mehreren DB-Instances zu verbessern. RDS-Proxy kann dazu beitragen, das Risiko von Verbindungsausfällen und Ausfallzeiten zu verringern und gleichzeitig die Verfügbarkeit und Skalierbarkeit zu verbessern.	Aktivieren Sie RDS-Proxy oder ändern Sie Ihre bestehende Proxykonfiguration.	Nein	Vertikale und horizontale Skalierung Ihrer Amazon RDS-Instance im AWS Datenbank-Blog Amazon-RDS-Proxy für Aurora Preise von Amazon-RDS-Proxy

Anzeigen von Metriken in der Amazon-RDS-Konsole

Amazon RDS lässt sich in Amazon CloudWatch integrieren, um eine Vielzahl von Aurora-DB-Cluster-Metriken in der RDS-Konsole anzuzeigen. Einige Metriken gelten auf Cluster-Ebene, während andere auf Instance-Ebene gelten.Für Beschreibungen (dieser Metriken)the instance-level and cluster-level metrics (der Instance-Ebene- und Cluster-Ebene-Metriken), finden Sie unter Amazon Aurora-Referenz für Metriken.

Für Ihren Aurora-DB-Cluster werden die folgenden Metrikkategorien überwacht:

- CloudWatch – zeigt die Amazon-CloudWatch-Metriken für Aurora , auf die Sie in der RDS-Konsole zugreifen können. Diese Metriken können Sie auch in der CloudWatch-Konsole aufrufen. Jede Metrik beinhaltet ein Diagramm, das die überwachte Metrik für ein bestimmtes Zeitintervall anzeigt. Eine Liste der CloudWatch-Metriken finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora.
- Enhanced Monitoring (Erweiterte Überwachung) – Zeigt eine Übersicht über Metriken des Betriebssystems an, wenn Ihre Aurora-DB-Cluster das Enhanced Monitoring aktiviert hat. RDS liefert die Metriken von „Enhanced Monitoring“·(Erweiterte·Überwachung) in Ihr Amazon-CloudWatch-Logs-Konto. Jede Betriebssystem-Metrik beinhaltet ein Diagramm, das die überwachte Metrik für ein bestimmtes Zeitintervall anzeigt. Eine Übersicht finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung). Eine Liste der Metriken für „Enhanced Monitoring“·(Erweiterte·Überwachung) finden Sie unter Betriebssystemmetriken in „Erweiterte Überwachung“.
- Betriebssystem-Prozessliste – Zeigt Detailinformationen über alle Prozesse, die innerhalb Ihres DB--Clusters ausgeführt werden.
- Performance Insights – Öffnet das Amazon-RDS-Performance-Insights-Dashboard für eine DB-Instance in Ihrem Aurora-DB-Cluster. Performance Insights wird auf Cluster-Ebene nicht unterstützt.Eine Übersicht über Performance Insights finden Sie unter Überwachung mit Performance Insights auf . Eine Liste der Peformance-Insights-Metriken finden Sie unter Amazon-CloudWatch-Metriken für Erkenntnisse zur Amazon-RDS-Leistung.

Amazon RDS bietet jetzt im Performance-Insights-Dashboard eine konsolidierte Ansicht der Performance-Insights- und CloudWatch-Metriken. Performance Insights muss aktiviert sein, damit Ihr DB-Cluster diese Ansicht verwenden kann. Sie können die neue Überwachungsansicht auf der Registerkarte Überwachung oder Performance Insights im Navigationsbereich auswählen.

Anleitungen zur Auswahl dieser Ansicht finden Sie unter Anzeigen kombinierter Metriken mit dem Performance-Insights-Dashboard.

Anzeigen kombinierter Metriken mit dem Performance-Insights-Dashboard

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.
Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.
Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

Amazon RDS bietet im Performance Insights-Dashboard eine konsolidierte Ansicht von Performance Insights und CloudWatch Metriken für Ihre DB-Instance. Sie können das vorkonfigurierte Dashboard verwenden oder ein benutzerdefiniertes Dashboard erstellen. Das vorkonfigurierte Dashboard enthält die am häufigsten verwendeten Metriken zur Diagnose von Leistungsproblemen für eine Datenbank-Engine. Alternativ können Sie ein benutzerdefiniertes Dashboard mit den Metriken für

eine Datenbank-Engine erstellen, die Ihren Analyseanforderungen entsprechen. Verwenden Sie dann dieses Dashboard für alle DB-Instances dieses Datenbank-Engine-Typs in Ihrem AWS Konto.

Sie können die Überwachungsansicht auf der Registerkarte Überwachung oder Performance Insights im Navigationsbereich auswählen.

Performance Insights muss für Ihren DB-Cluster aktiviert sein, um die kombinierten Metriken im Performance-Insights-Dashboard anzeigen zu können. Weitere Informationen zum Aktivieren von Performance Insights finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora.

In den folgenden Abschnitten erfahren Sie, wie Sie Performance Insights und CloudWatch Metriken anzeigen.

Themen

- Auswählen der neuen Überwachungsansicht auf der Registerkarte „Überwachung“
- Auswählen der neuen Überwachungsansicht auf der Performance-Insights-Seite
- Erstellen eines benutzerdefinierten Dashboards mit Performance Insights
- Auswählen des vorkonfigurierten Dashboards mit Performance Insights

Auswählen der neuen Überwachungsansicht auf der Registerkarte „Überwachung“

> **Important**
>
> AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
> Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.

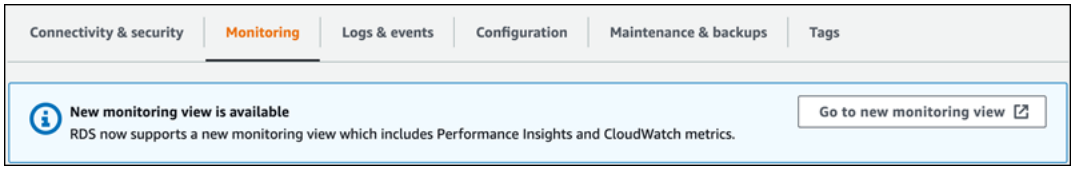
Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.
Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

In der Amazon RDS-Konsole können Sie die neue Überwachungsansicht wählen, um Performance Insights und CloudWatch Metriken für Ihre DB-Instance anzuzeigen.

So wählen Sie die neue Überwachungsansicht auf der Registerkarte „Überwachung“ aus

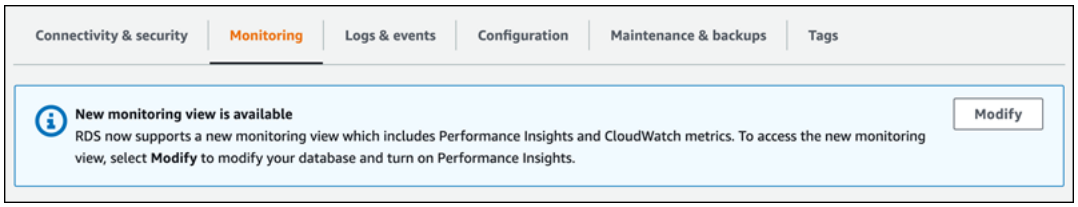
1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Datenbanken aus.
3. Wählen Sie den Aurora-DB-Cluster zur Überwachung aus.
4. Scrollen Sie nach unten und wählen Sie die Registerkarte Überwachung aus.

   Es erscheint ein Banner mit der Option, die neue Überwachungsansicht auszuwählen. Das folgende Beispiel veranschaulicht das Banner zur Auswahl der neuen Überwachungsansicht.



5. Wählen Sie Gehe zu neuer Überwachungsansicht, um das Performance Insights-Dashboard mit Performance Insights und CloudWatch Metriken für Ihre zu öffnen.
6. (Optional) Wenn Performance Insights für Ihre DB-Instance deaktiviert ist, wird ein Banner mit der Option angezeigt, Ihre DB-Instance zu ändern und Performance Insights zu aktivieren.

Das folgende Beispiel veranschaulicht das Banner zum Ändern der DB-Instance auf der Registerkarte Überwachung.



Wählen Sie Ändern aus, um Ihre DB-Instance zu ändern und Performance Insights zu aktivieren. Weitere Informationen zum Aktivieren von Performance Insights finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora.

Auswählen der neuen Überwachungsansicht auf der Performance-Insights-Seite

Important

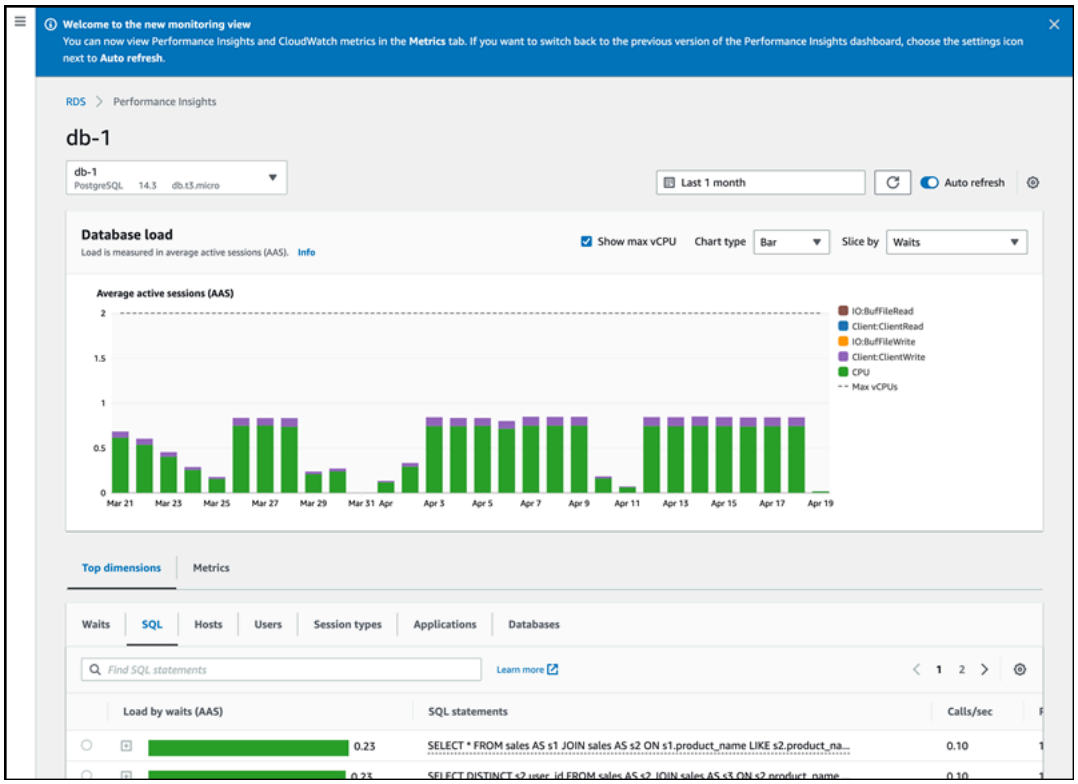
AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.
Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.

Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

In der Amazon RDS-Konsole können Sie die neue Überwachungsansicht wählen, um Performance Insights und CloudWatch Metriken für Ihre DB-Instance anzuzeigen.

So wählen Sie die neue Überwachungsansicht mit Performance Insights im Navigationsbereich aus

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus, um das Performance Insights-Dashboard aufzurufen, das sowohl Performance Insights als auch CloudWatch Metriken für Ihre DB-Instance anzeigt.



Erstellen eines benutzerdefinierten Dashboards mit Performance Insights

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.
Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.
Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

In der neuen Überwachungsansicht können Sie ein benutzerdefiniertes Dashboard mit den Metriken erstellen, die Sie für Ihre Analyseanforderungen benötigen.

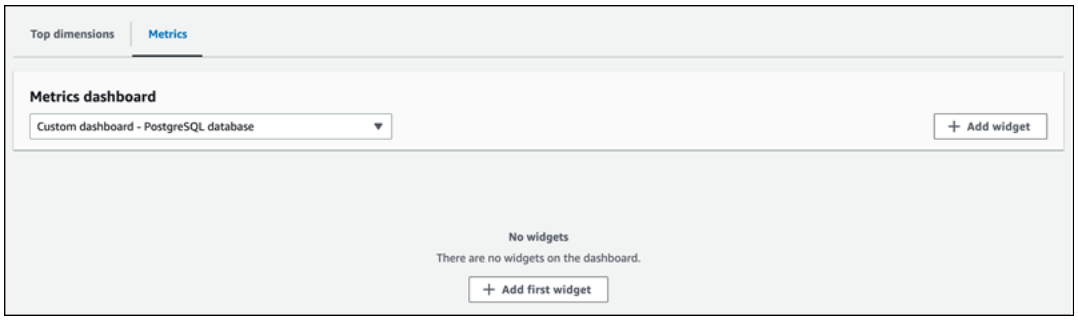
Sie können ein benutzerdefiniertes Dashboard erstellen, indem Sie Performance Insights und CloudWatch Metriken für Ihre DB-Instance auswählen. Sie können dieses benutzerdefinierte Dashboard für andere DB-Instances desselben Datenbank-Engine-Typs in Ihrem AWS Konto verwenden.

> **Note**
>
> Das benutzerdefinierte Dashboard unterstützt bis zu 50 Metriken.

Verwenden Sie das Widget-Einstellungsmenü, um das Dashboard zu bearbeiten oder zu löschen und das Widget-Fenster zu verschieben oder in der Größe zu ändern.

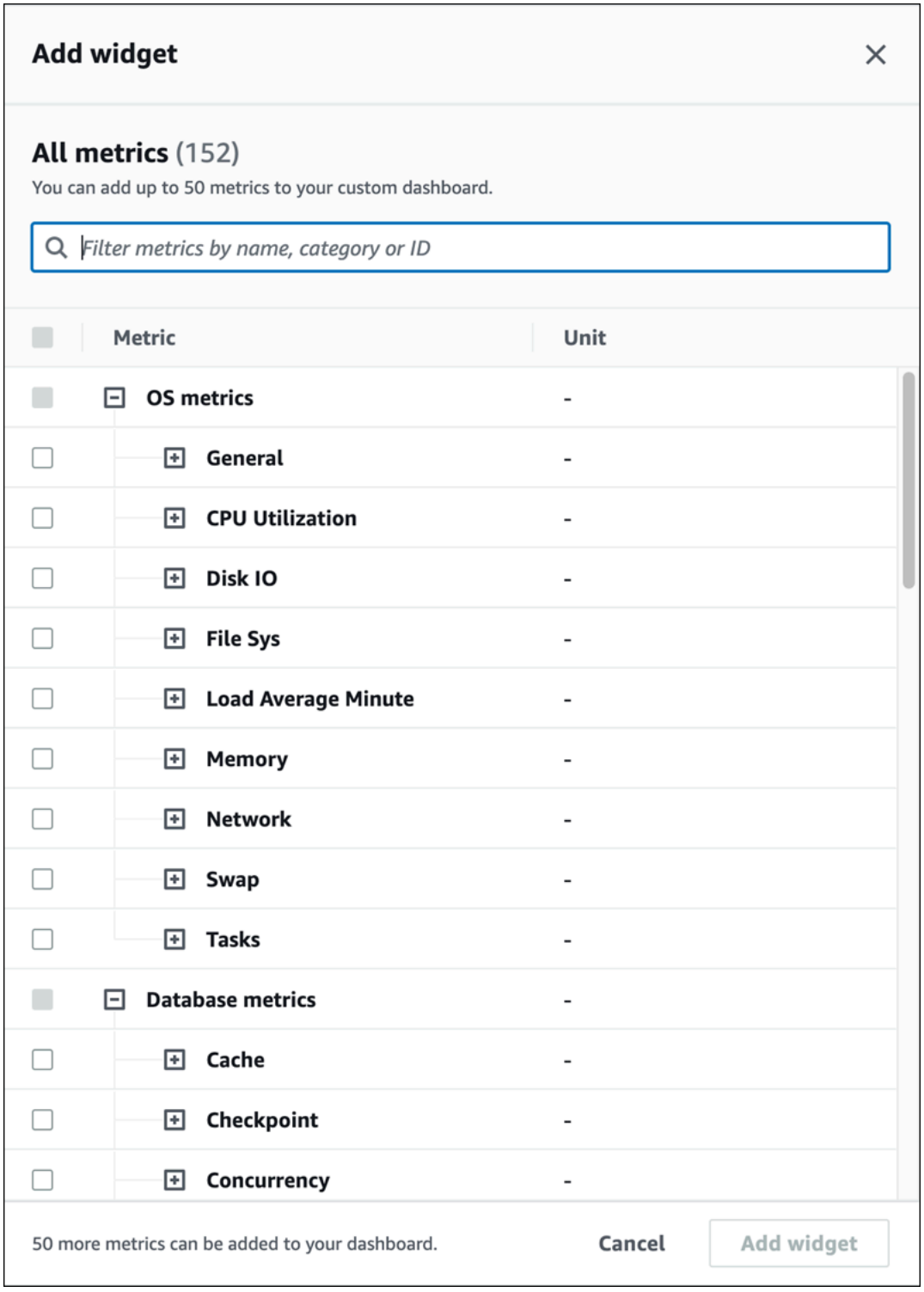
So erstellen Sie ein benutzerdefiniertes Dashboard mit Performance Insights im Navigationsbereich

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.
4. Scrollen Sie im Fenster nach unten zur Registerkarte Metriken.
5. Wählen Sie das benutzerdefinierte Dashboard aus der Dropdown-Liste aus. Das folgende Beispiel veranschaulicht die Erstellung eines benutzerdefinierten Dashboards.



6. Wählen Sie Widget hinzufügen aus, um das Fenster Widget hinzufügen zu öffnen. Sie können die verfügbaren Betriebssystem- (OS) -Metriken, Datenbank-Metriken und CloudWatch Metriken im Fenster öffnen und anzeigen.

   Das folgende Beispiel veranschaulicht das Fenster Widget hinzufügen mit den Metriken.



7. Wählen Sie die Metriken aus, die Sie im Dashboard anzeigen möchten. Klicken Sie dann auf Widget hinzufügen. Sie können über das Suchfeld eine bestimmte Metrik suchen.

   Die ausgewählten Metriken werden in Ihrem Dashboard angezeigt.

8. (Optional) Wenn Sie Ihr Dashboard ändern oder löschen möchten, wählen Sie das Einstellungensymbol oben rechts im Widget und dann eine der folgenden Aktionen im Menü aus.

   - Bearbeiten – Ändern Sie die Metrikenliste im Fenster. Wählen Sie Widget aktualisieren aus, nachdem Sie die Metriken für Ihr Dashboard ausgewählt haben.
   - Löschen – Löscht das Widget. Wählen Sie im Bestätigungsfenster Löschen aus.

Auswählen des vorkonfigurierten Dashboards mit Performance Insights

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.
Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.
Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

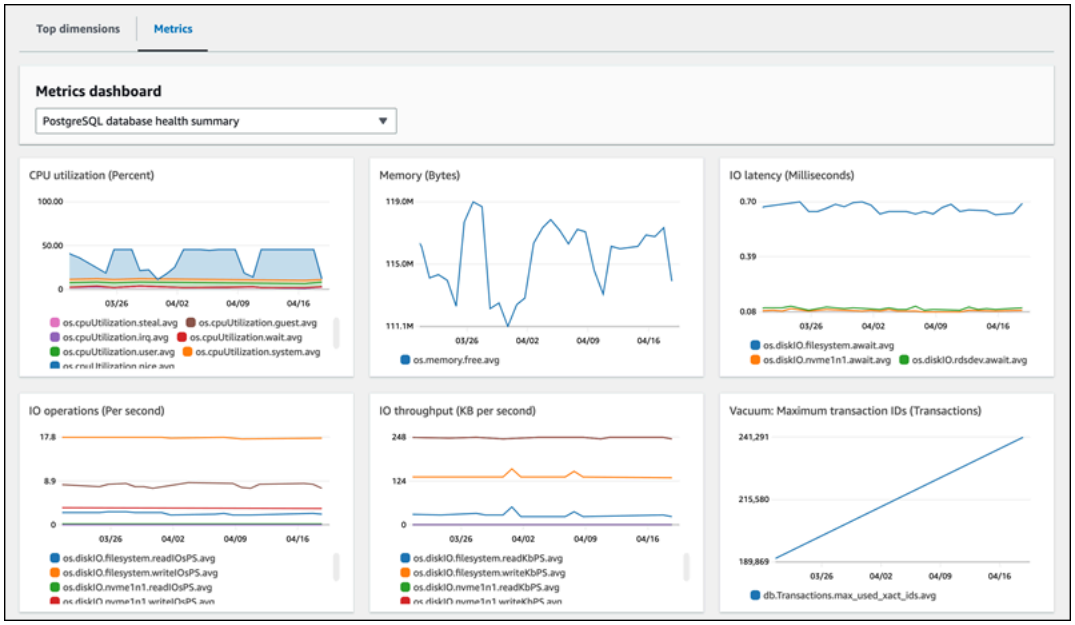
Sie können die am häufigsten verwendeten Metriken über das vorkonfigurierte Dashboard anzeigen. Dieses Dashboard hilft bei der Diagnose von Leistungsproblemen mit einer Datenbank-Engine und reduziert die durchschnittliche Wiederherstellungszeit von Stunden auf Minuten.

> **Note**
>
> Dieses Dashboard kann nicht bearbeitet werden.

So wählen Sie das vorkonfigurierte Dashboard mit Performance Insights im Navigationsbereich aus

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.
4. Scrollen Sie im Fenster nach unten zur Registerkarte Metriken.
5. Wählen Sie ein vorkonfiguriertes Dashboard aus der Dropdown-Liste aus.

   Sie können die Metriken für die DB-Instance im Dashboard anzeigen. Das folgende Beispiel veranschaulicht ein vorkonfiguriertes Metriken-Dashboard.



Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch

Amazon CloudWatch ist ein Repository für Metriken. Das Repository erfasst und verarbeitet Rohdaten aus Amazon Aurora, um nahezu in Echtzeit lesbare Metriken bereitzustellen. Eine vollständige Liste von Amazon Aurora- Metriken, die an CloudWatch gesendet werden, finden Sie unter Metrikreferenz für Amazon Aurora.

Verwenden Sie CloudWatch Database Insights, um die Leistung Ihrer Datenbanken in großem Maßstab zu analysieren und Fehler zu beheben.

Themen

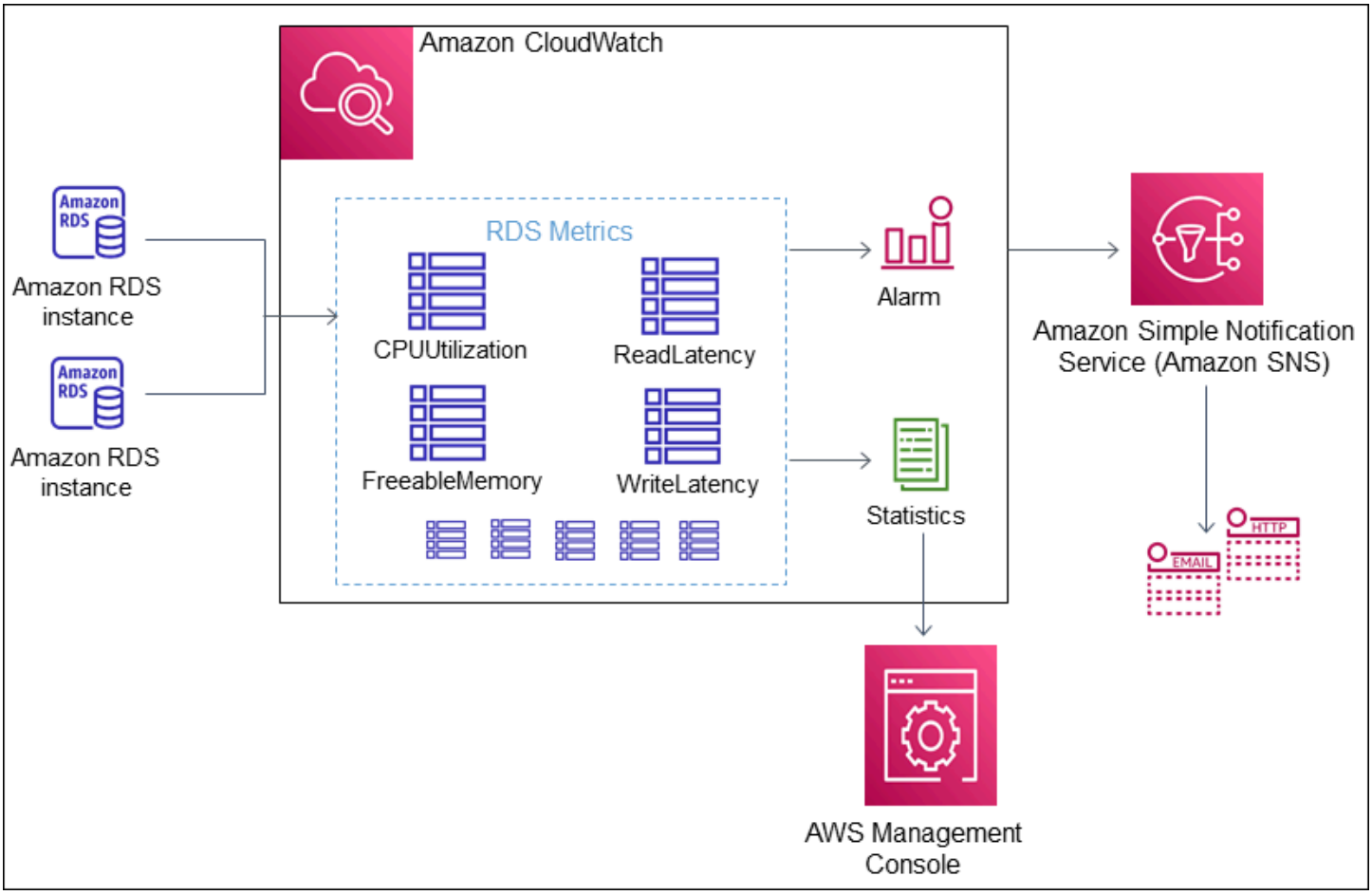
- Übersicht über Amazon Aurora und Amazon CloudWatch
- Anzeigen von DB-Cluster-Metriken in der CloudWatch-Konsole und AWS CLI
- Exportieren von Performance-Insights-Metriken nach CloudWatch
- Erstellen von CloudWatch-Alarmen zur Überwachung von Amazon Aurora

Übersicht über Amazon Aurora und Amazon CloudWatch

Standardmäßig werden Metrikdaten von Amazon Aurora in Abständen von 1 Minute automatisch an CloudWatch gesendet. Die Metrik `CPUUtilization` zeichnet z. B. den Prozentsatz der CPU-Auslastung für eine DB-Instance im Laufe der Zeit auf. Datenpunkte mit einem Zeitraum von 60 Sekunden (1 Minute) stehen 15 Tage lang zur Verfügung Dies bedeutet, dass Sie auf die Verlaufsdaten zugreifen und sehen können, wie die Leistung Ihrer Webanwendung oder Ihres Service ist.

Sie können jetzt Metrik-Dashboards von Performance Insights von Amazon RDS zu Amazon CloudWatch exportieren. Sie können die vorkonfigurierten oder benutzerdefinierten Metrik-Dashboards als neues Dashboard exportieren oder sie einem vorhandenen CloudWatch-Dashboard hinzufügen. Das exportierte Dashboard kann in der CloudWatch-Konsole angezeigt werden. Weitere Informationen zum Exportieren der Metrik-Dashboards von Performance Insights zu CloudWatch finden Sie unter Exportieren von Performance-Insights-Metriken nach CloudWatch.

Wie im folgenden Diagramm gezeigt, können Sie Alarme für Ihre CloudWatch-Metriken einrichten. Beispielsweise könnten Sie einen Alarm erstellen, der signalisiert, wenn die CPU-Auslastung für eine Instance über 70 % beträgt. Sie können Amazon Simple Notification Service so konfigurieren, dass Sie eine E-Mail erhalten, wenn der Schwellenwert überschritten wird.



Amazon RDS veröffentlicht die folgenden Arten von Metriken auf Amazon CloudWatch:

- Aurora-Metriken sowohl auf Cluster- als auch auf Instance-Ebene

  Eine Tabelle dieser Metriken finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora.
- Performance-Insights-Metriken

  Eine Tabelle dieser Metriken finden Sie unter Amazon-CloudWatch-Metriken für Erkenntnisse zur Amazon-RDS-Leistung und Performance-Insights-Zählermetriken.
- Enhanced-Monitoring-Metriken (veröffentlicht in Amazon CloudWatch Logs)

  Eine Tabelle dieser Metriken finden Sie unter Betriebssystemmetriken in „Erweiterte Überwachung“.
- Nutzungsmetriken für die Amazon-RDS-Service-Quotas in Ihrem AWS-Konto

Eine Tabelle dieser Metriken finden Sie unter CloudWatch Amazon-Nutzungsmetriken für Amazon Aurora. Weitere Informationen über Amazon-RDS-Kontingente finden Sie unter Kontingente und Beschränkungen für Amazon Aurora.

Weitere Informationen zu CloudWatch finden Sie unter Was ist Amazon CloudWatch? im Amazon CloudWatch-Benutzerhandbuch. Weitere Informationen zur Aufbewahrung von CloudWatch-Metriken finden Sie unter Aufbewahrung von Metriken.

Anzeigen von DB-Cluster-Metriken in der CloudWatch-Konsole und AWS CLI

Im Folgenden finden Sie Details zum Anzeigen von Metriken für Ihre DB-Instance mithilfe CloudWatch. Weitere Informationen zur Überwachung der Metriken zum Betriebssystem Ihrer DB-Instance in Echtzeit mit CloudWatch Logs finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).

Wenn Sie Amazon-Aurora-Ressourcen verwenden, sendet Amazon Aurora jede Minute Metriken und Dimensionen an Amazon CloudWatch.

Sie können jetzt Metrik-Dashboards von Performance Insights von Amazon RDS zu Amazon CloudWatch exportieren und diese Metriken in der CloudWatch-Konsole anzeigen. Weitere Informationen zum Exportieren der Metrik-Dashboards von Performance Insights zu CloudWatch finden Sie unter Exportieren von Performance-Insights-Metriken nach CloudWatch.

Nutzen Sie die folgenden Vorgehensweisen, um die Metriken für Amazon Aurora in der CloudWatch-Konsole und der CLI anzuzeigen.

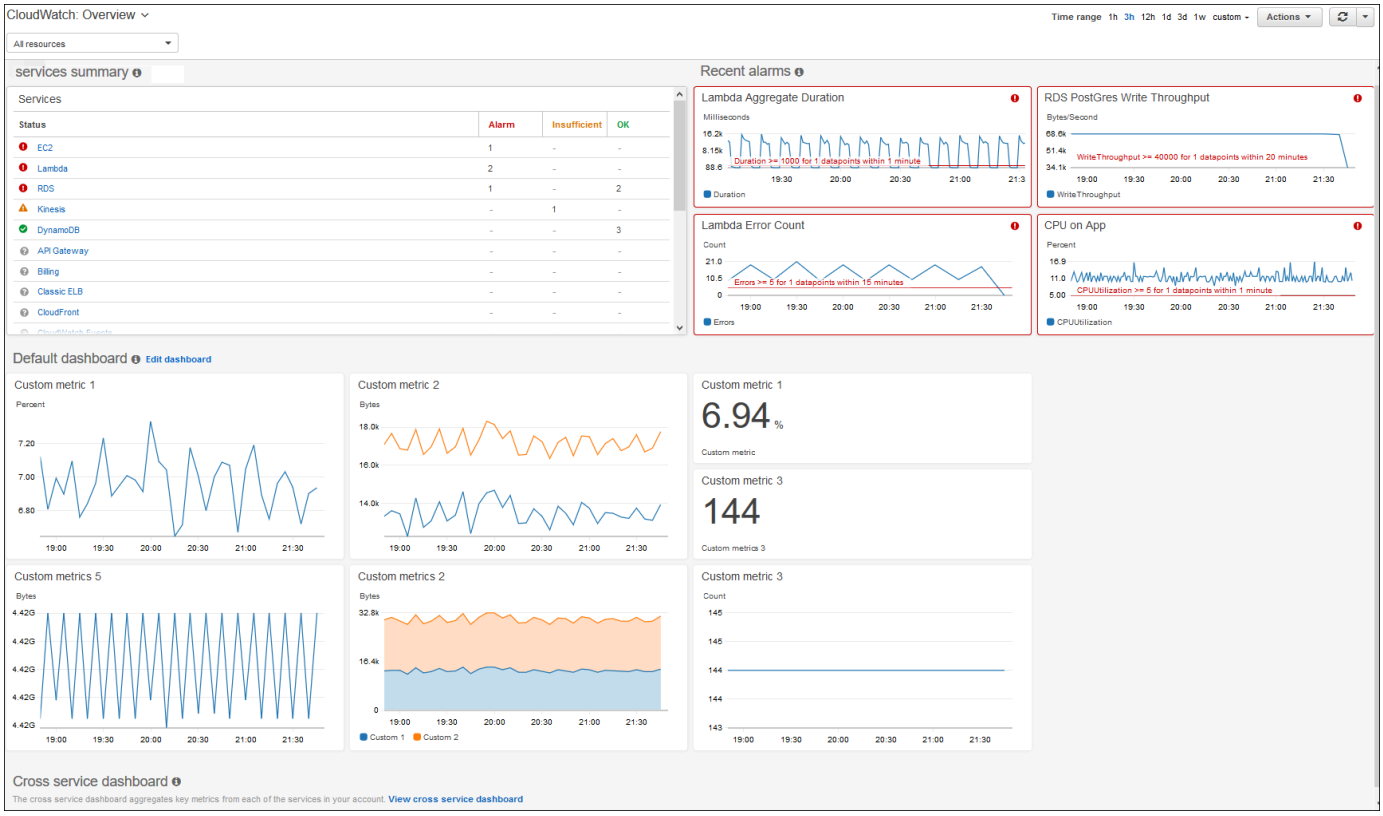
Konsole

So zeigen Sie Metriken mithilfe der Amazon CloudWatch-Konsole an

Metriken werden zunächst nach dem Service-Namespace und anschließend nach den verschiedenen Dimensionskombinationen in den einzelnen Namespaces gruppiert.

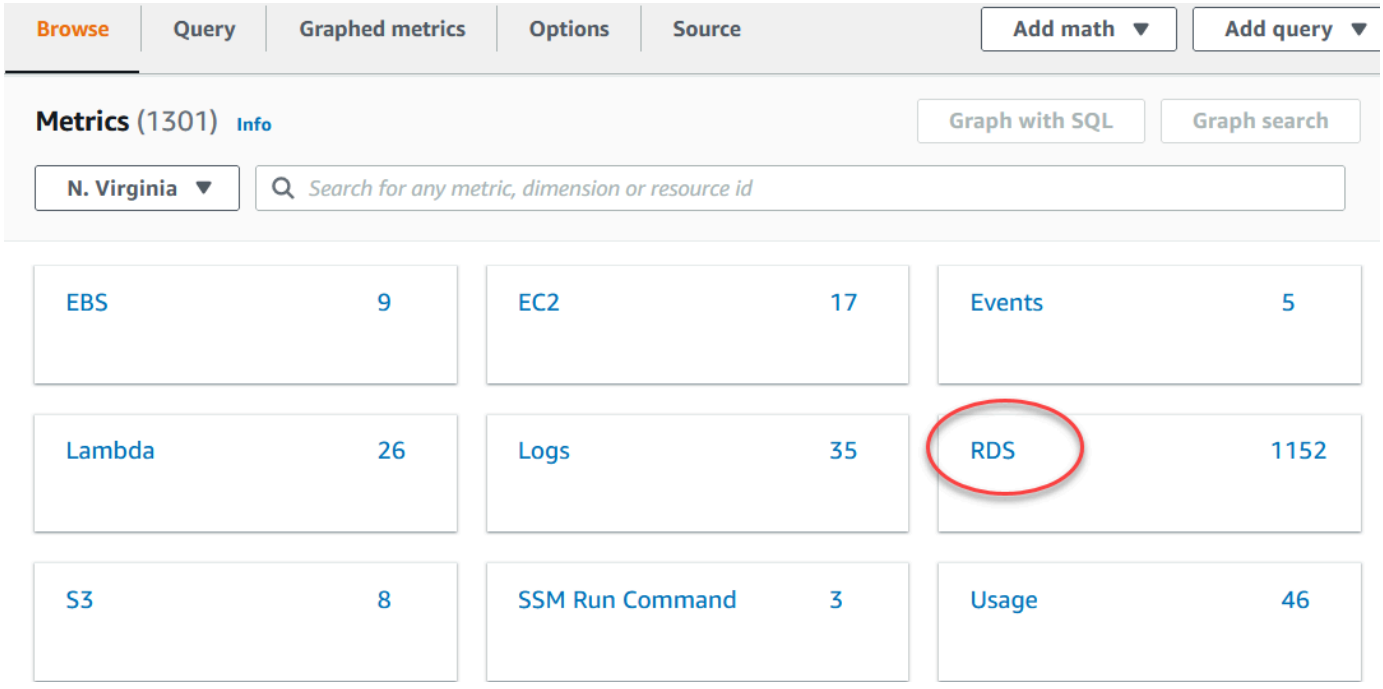
1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.

   Die Startseite mit der CloudWatch-Übersicht erscheint.



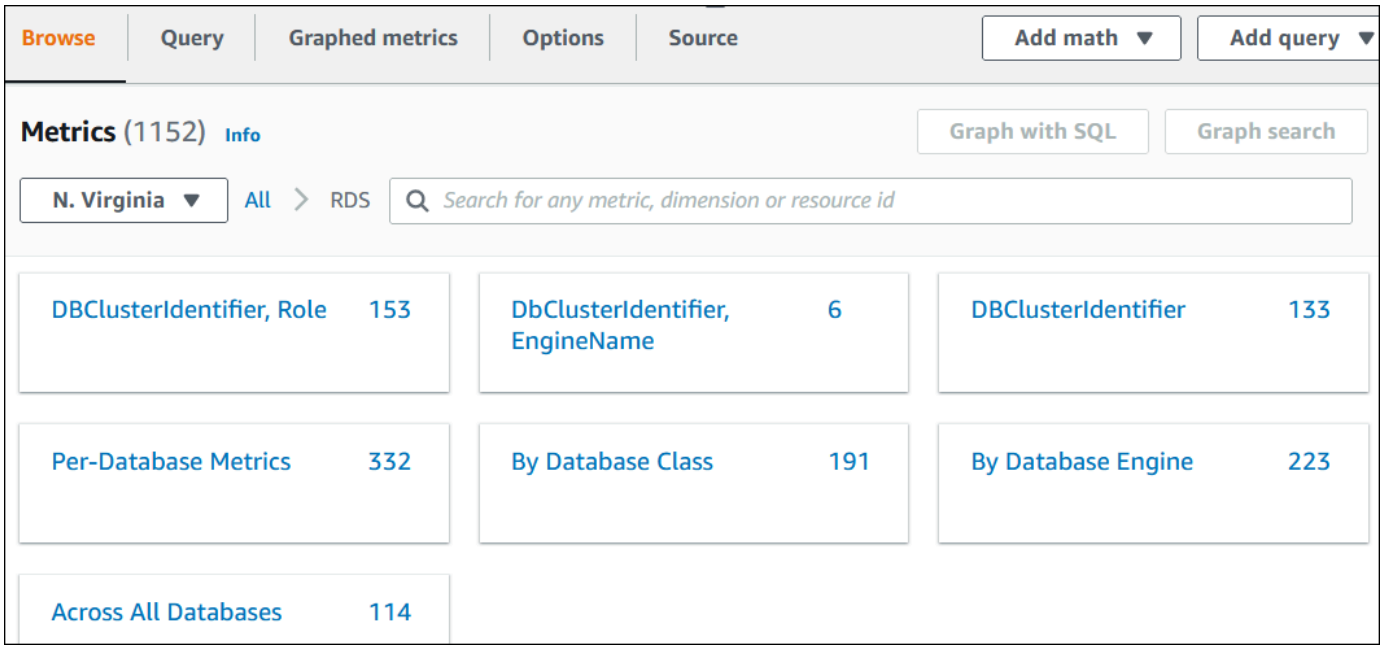
2. Ändern Sie, falls erforderlich, die AWS-Region. Wählen Sie in der Navigationsleiste die AWS-Region aus, in der sich Ihre AWS-Ressourcen befinden. Weitere Informationen finden Sie unter Regionen und Endpunkte.

3. Wählen Sie im Navigationsbereich Metrics (Metriken) und dann All metrics (Alle Metriken) aus.

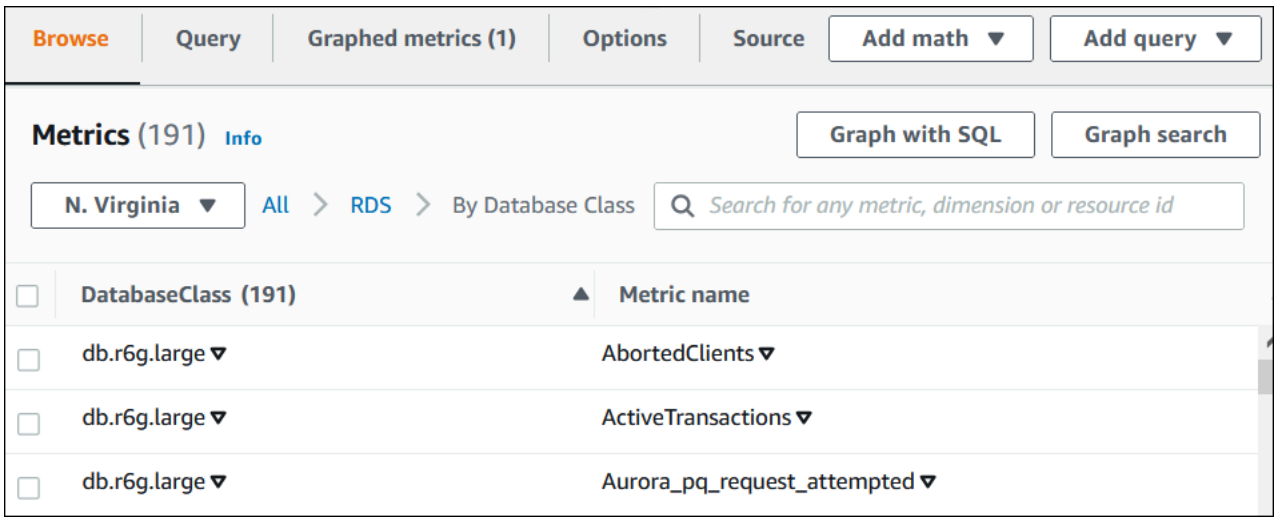


4. Scrollen Sie nach unten und wählen Sie den RDS-Namespace der Metrik aus.

   Auf der Seite werden die Amazon Aurora-Dimensionen angezeigt. Beschreibungen dieser Dimensionen finden Sie unter CloudWatch Amazon-Abmessungen für Aurora.



5. Wählen Sie eine Metrikdimension, z. B. By Database Class (Nach Datenbanken-Klasse).

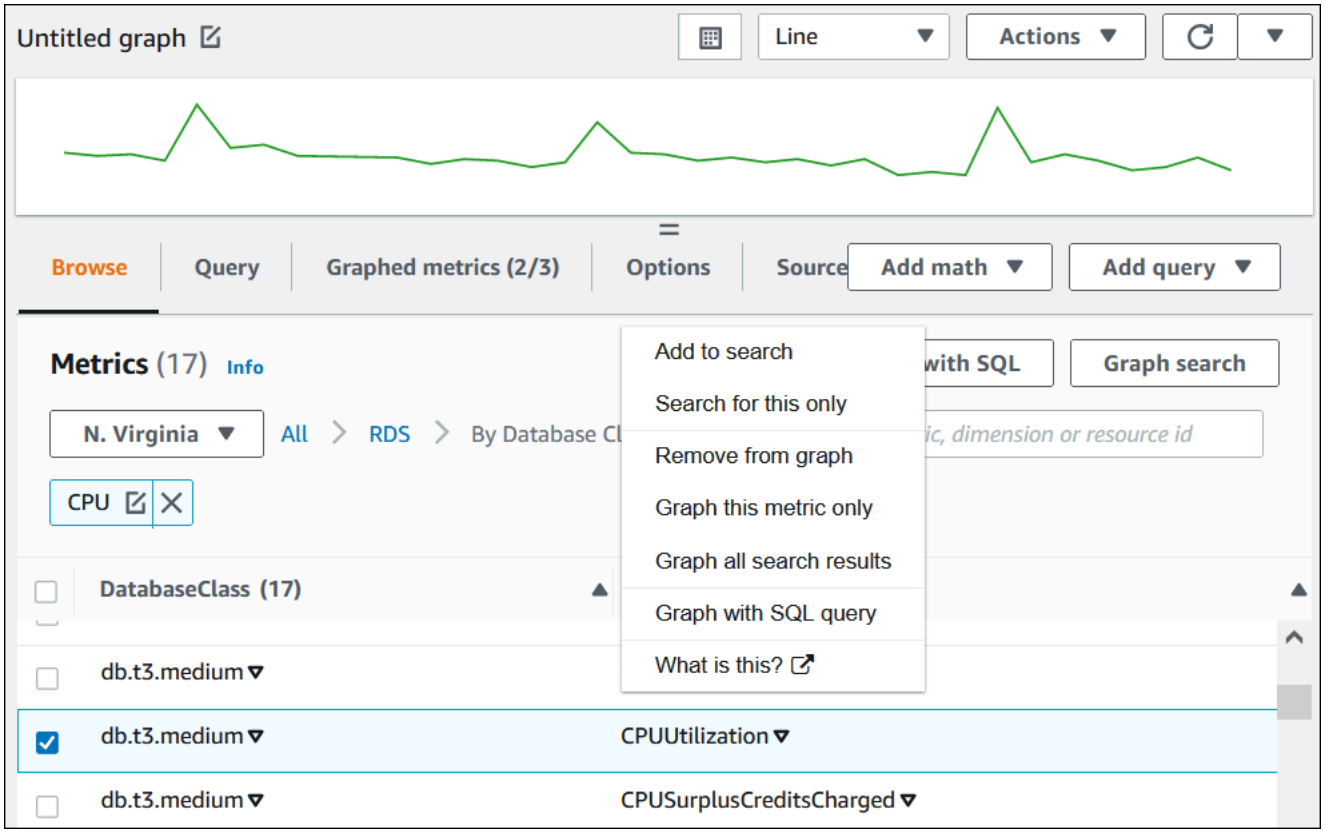


6. Führen Sie eine der folgenden Aktionen aus:

   - Verwenden Sie die Spaltenüberschrift, um die Metriken zu sortieren.
   - Um eine Metrik grafisch darzustellen, müssen Sie das Kontrollkästchen neben der Metrik aktivieren.

- Um nach Ressource zu filtern, müssen Sie zunächst die Ressourcen-ID und dann die Option Add to search (Zu Suche hinzufügen) wählen.
- Um nach Metrik zu filtern, müssen Sie den Metriknamen und anschließend Add to search (Zu Suche hinzufügen) wählen.

Das folgende Beispiel filtert die db.t3.medium-Klasse und stellt die CPUUtilization-Metrik grafisch dar.



Sie erhalten Details darüber, wie Sie die Ressourcennutzung für Aurora PostgreSQL mithilfe von CloudWatch-Metriken analysieren können. Weitere Informationen finden Sie unter Verwendung von CloudWatch Amazon-Metriken zur Analyse der Ressourcennutzung für Aurora PostgreSQL

AWS CLI

Um über die AWS CLI Metrikinformationen zu erhalten, verwenden Sie den CloudWatch-Befehl `list-metrics`. Im folgenden Beispiel listen Sie alle Metriken im `AWS/RDS`-Namespace auf.

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

Um Metrikdaten zu erhalten, verwenden Sie den Befehl `get-metric-data`.

Im folgenden Beispiel werden `CPUUtilization`-Statistiken für die Instance `my-instance` über den jeweiligen 24-Stunden-Zeitraum mit einer Granularität von 5 Minuten abgerufen.

Erstellen Sie eine JSON Datei mit dem Namen `CPU_metric.json` und dem folgenden Inhalt.

```
{
   "StartTime" : "2023-12-25T00:00:00Z",
   "EndTime" : "2023-12-26T00:00:00Z",
   "MetricDataQueries" : [{
     "Id" : "cpu",
     "MetricStat" : {
    "Metric" : {
        "Namespace" : "AWS/RDS",
        "MetricName" : "CPUUtilization",
        "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
    },
       "Period" : 360,
       "Stat" : "Minimum"
     }
   }]
}
```

Example

Für Linux, macOS oder Unix:

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

Für Windows:

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

Beispielausgabe:

```
{
    "MetricDataResults": [
        {
```

```
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
                "2023-12-15T23:24:00+00:00",
                ...
            ],
            "Values": [
                13.299778337027714,
                13.677507543049558,
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

Weitere Informationen finden Sie unter Abrufen von Statistiken für eine Metrik im Amazon-CloudWatch-Benutzerhandbuch.

Exportieren von Performance-Insights-Metriken nach CloudWatch

Mit Performance Insights können Sie das vorkonfigurierte oder benutzerdefinierte Metrik-Dashboard für Ihre DB-Instance zu Amazon CloudWatch exportieren. Sie können das Metrik-Dashboard als neues Dashboard exportieren oder es einem vorhandenen CloudWatch-Dashboard hinzufügen. Wenn Sie das Dashboard einem vorhandenen CloudWatch-Dashboard hinzufügen möchten, können Sie eine Header-Bezeichnung erstellen, sodass die Metriken in einem separaten Abschnitt im CloudWatch-Dashboard angezeigt werden.

Sie können das exportierte Metrik-Dashboard in der CloudWatch-Konsole anzeigen. Wenn Sie einem Dashboard mit Performance-Insights-Metriken nach dem Export neue Metriken hinzufügen, müssen Sie dieses Dashboard erneut exportieren, damit die neuen Metriken in der CloudWatch-Konsole angezeigt werden.

Sie können auch ein Metrik-Widget im Performance-Insights-Dashboard auswählen und die Metrikdaten in der CloudWatch-Konsole anzeigen.

Weitere Informationen zum Anzeigen der Metriken in der CloudWatch-Konsole finden Sie unter Anzeigen von DB-Cluster-Metriken in der CloudWatch-Konsole und AWS CLI.

Exportieren Sie in den folgenden Abschnitten Performance-Insights-Metriken als neues oder vorhandenes Dashboard nach CloudWatch und zeigen Sie Performance-Insights-Metriken in CloudWatch an.

Themen

- Exportieren von Performance-Insights-Metriken als neues Dashboard nach CloudWatch
- Hinzufügen von Performance-Insights-Metriken zu einem vorhandenen CloudWatch-Dashboard
- Anzeigen eines Metrik-Widgets von Performance Insights in CloudWatch

Exportieren von Performance-Insights-Metriken als neues Dashboard nach CloudWatch

Wählen Sie im Performance-Insights-Dashboard ein vorkonfiguriertes oder benutzerdefiniertes Metrik-Dashboard aus und exportieren Sie es als neues Dashboard nach CloudWatch. Sie können das exportierte Dashboard in der CloudWatch-Konsole anzeigen.

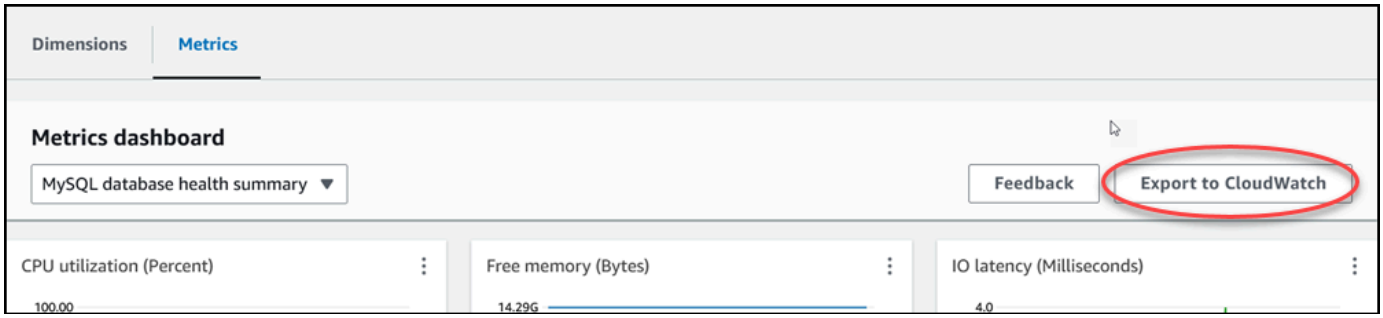
So exportieren Sie ein Dashboard mit Performance-Insights-Metriken als neues Dashboard nach CloudWatch

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.

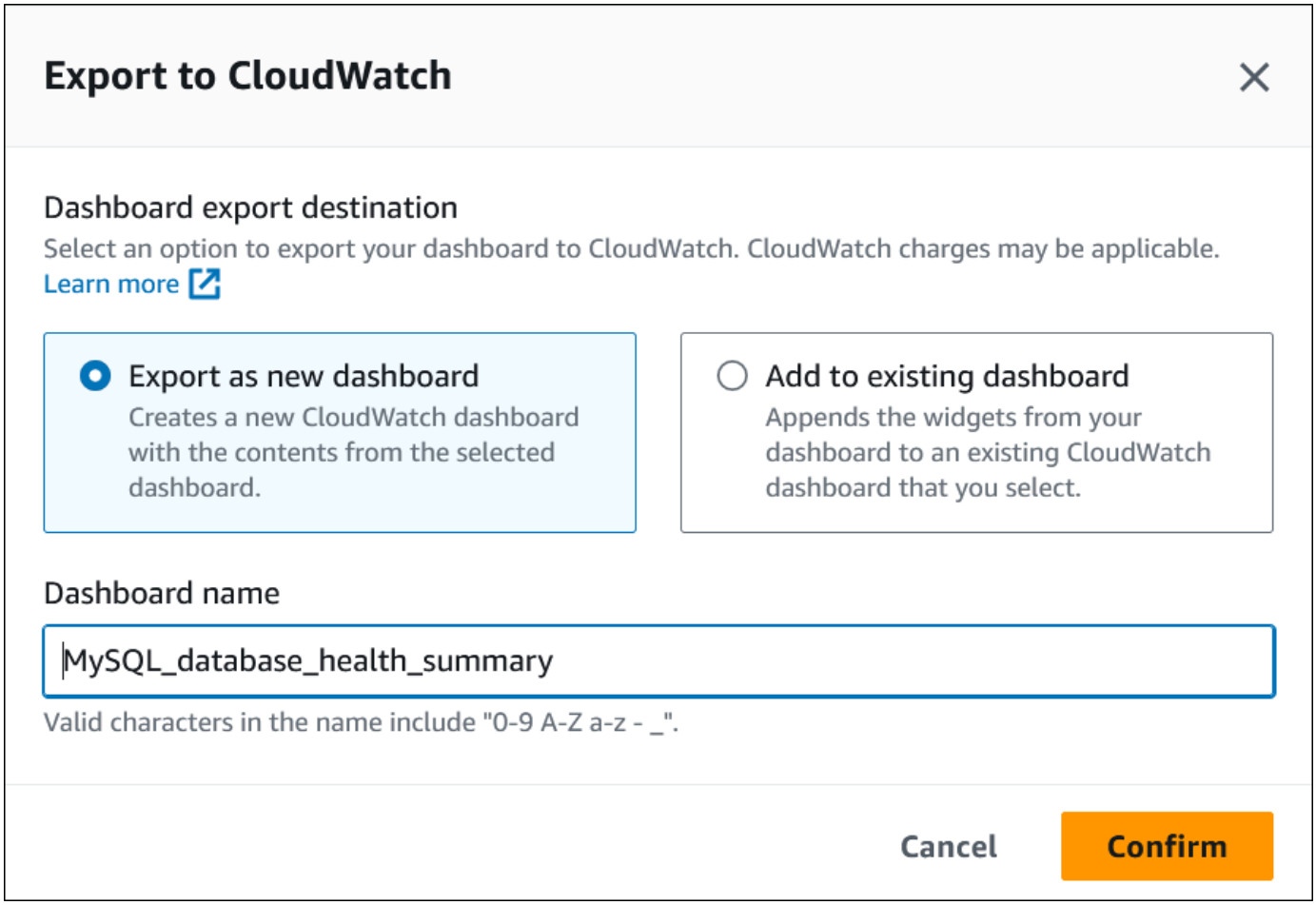
   Das Performance-Insights-Dashboard wird für die DB-Instance angezeigt.
4. Scrollen Sie nach unten und wählen Sie Metriken aus.

   Standardmäßig wird das vorkonfigurierte Dashboard mit Performance-Insights-Metriken angezeigt.
5. Wählen Sie ein vorkonfiguriertes oder benutzerdefiniertes Dashboard und dann Export nach CloudWatch aus.

   Das Fenster Export nach CloudWatch wird angezeigt.

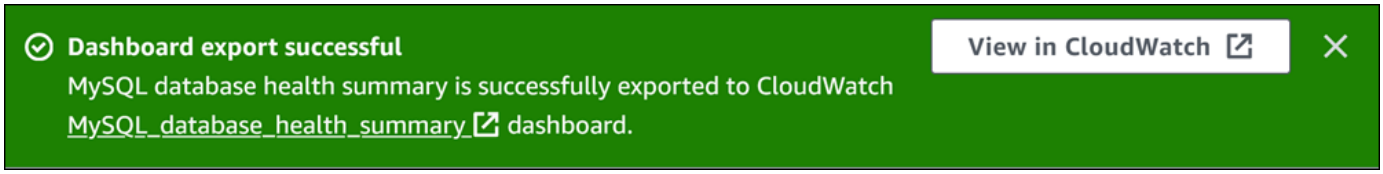


6. Wählen Sie Als neues Dashboard exportieren aus.



7. Geben Sie im Feld Dashboard-Name einen Namen für das neue Dashboard ein und wählen Sie Bestätigen aus.

   In einem Banner wird eine Meldung angezeigt, nachdem der Dashboard-Export erfolgreich war.



8. Wählen Sie im Banner den Link oder In CloudWatch anzeigen, um das Metrik-Dashboard in der CloudWatch-Konsole aufzurufen.

Hinzufügen von Performance-Insights-Metriken zu einem vorhandenen CloudWatch-Dashboard

Fügen Sie einem vorhandenen CloudWatch-Dashboard ein vorkonfiguriertes oder benutzerdefiniertes Metrik-Dashboard hinzu. Sie können dem Metrik-Dashboard eine Bezeichnung hinzufügen, die in einem separaten Abschnitt im CloudWatch-Dashboard angezeigt wird.

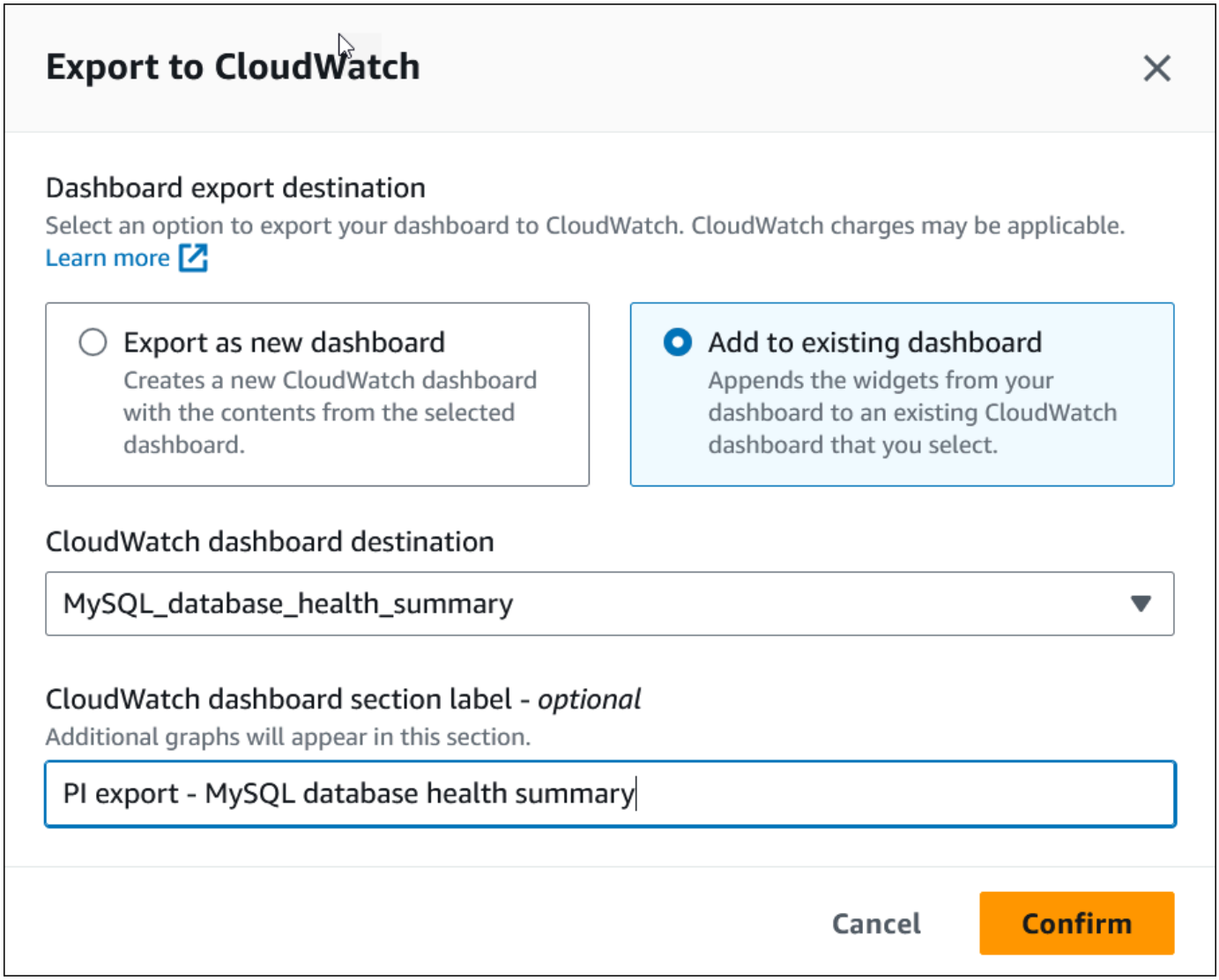
So exportieren Sie die Metriken in ein vorhandenes CloudWatch-Dashboard

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.

   Das Performance-Insights-Dashboard wird für die DB-Instance angezeigt.
4. Scrollen Sie nach unten und wählen Sie Metriken aus.

   Standardmäßig wird das vorkonfigurierte Dashboard mit Performance-Insights-Metriken angezeigt.
5. Wählen Sie das vorkonfigurierte oder benutzerdefinierte Dashboard und dann Export nach CloudWatch aus.

   Das Fenster Export nach CloudWatch wird angezeigt.
6. Wählen Sie Zu vorhandenem Dashboard hinzufügen aus.



7. Geben Sie das Ziel und die Bezeichnung des Dashboards an und wählen Sie dann Bestätigen aus.

   - CloudWatch-Dashboard-Ziel – Wählen Sie ein vorhandenes CloudWatch-Dashboard aus.
   - Bezeichnung des CloudWatch-Dashboard-Abschnitts – optional – Geben Sie einen Namen für die Performance-Insights-Metriken ein, die in diesem Abschnitt im CloudWatch-Dashboard angezeigt werden sollen.

   In einem Banner wird eine Meldung angezeigt, nachdem der Dashboard-Export erfolgreich war.

8. Wählen Sie im Banner den Link oder In CloudWatch anzeigen, um das Metrik-Dashboard in der CloudWatch-Konsole aufzurufen.

Anzeigen eines Metrik-Widgets von Performance Insights in CloudWatch

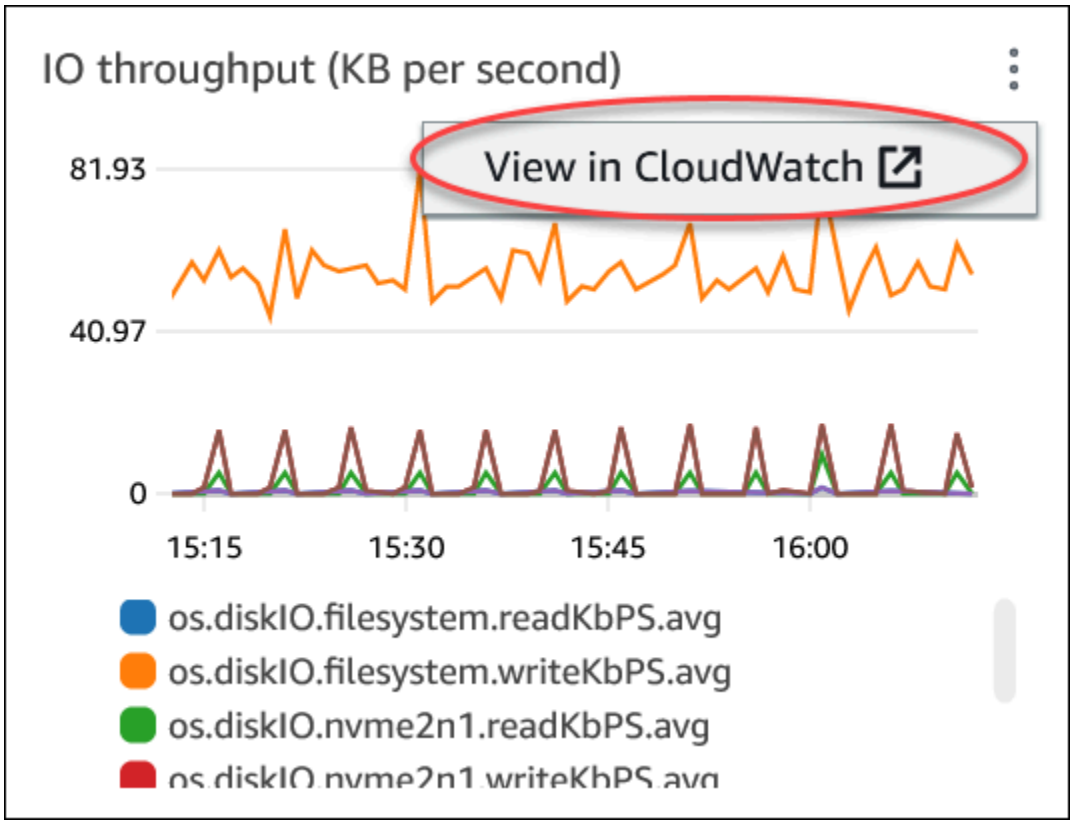
Wählen Sie ein Metrik-Widget von Performance Insights im Dashboard von Amazon RDS Performance Insights aus und zeigen Sie die Metrikdaten in der CloudWatch-Konsole an.

So exportieren Sie ein Metrik-Widget und zeigen die Metrikdaten in der CloudWatch-Konsole an

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.

   Das Performance-Insights-Dashboard wird für die DB-Instance angezeigt.
4. Scrollen Sie nach unten zu Metriken.

   Standardmäßig wird das vorkonfigurierte Dashboard mit Performance-Insights-Metriken angezeigt.
5. Wählen Sie ein Metrik-Widget und dann im Menü die Option In CloudWatch anzeigen aus.



   Die Metrikdaten werden in der CloudWatch-Konsole angezeigt.

Erstellen von CloudWatch-Alarmen zur Überwachung von Amazon Aurora

Sie können einen CloudWatch-Alarm erstellen, der eine Amazon SNS-Nachricht sendet, sobald sich der Status des Alarms ändert. Ein Alarm überwacht eine Metrik über einen bestimmten, von Ihnen definierten Zeitraum. Der Alarm kann auch eine oder mehrere Aktionen durchführen, die vom Wert der Metrik im Vergleich zu einem gegebenen Schwellenwert in einer Reihe von Zeiträumen abhängt. Die Aktion ist eine Benachrichtigung, die an ein Amazon SNS-Thema oder eine Amazon EC2 Auto Scaling-Richtlinie gesendet wird.

Alarme rufen nur Aktionen für nachhaltige Statusänderungen auf. CloudWatch-Alarme rufen keine Aktionen auf, nur weil sie sich in einem bestimmten Status befinden. Der Status muss sich geändert haben und für eine festgelegte Anzahl an Zeiträumen aufrechterhalten worden sein.

> **Note**
>
> Sie sollten in Aurora `WRITER`- oder `READER`-Rollenmetriken zum Einrichten von Alarmen verwenden, statt sich auf Metriken für bestimmte DB-Instances zu verlassen. Aurora DB-Instance-Rollen können ihre Rolle mit der Zeit ändern. Sie finden diese rollenbasierten Metriken in der CloudWatch-Konsole.
> Aurora Auto Scaling legt Alarme automatisch auf der Basis von `READER`-Rollenmetriken fest. Weitere Informationen zu Aurora Auto Scaling finden Sie unter Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten.

Sie können die metrische mathematische Funktion DB_PERF_INSIGHTS in der CloudWatch-Konsole verwenden, um Amazon RDS nach Performance-Insights-Zählermetriken abzufragen. Die Funktion DB_PERF_INSIGHTS umfasst auch die DBLoad-Metrik in Intervallen unter einer Minute. Sie können für diese Metriken CloudWatch-Alarme einstellen.

Weitere Informationen zum Erstellen eines Alarms finden Sie unter Erstellen eines Alarms zu Performance-Insights-Zählermetriken aus einer AWS-Datenbank.

So richten Sie mit der AWS CLI einen Alarm ein

- Rufen Sie die folgende Seite auf `put-metric-alarm`. Weitere Informationen finden Sie in der AWS CLI-Befehlsreferenz.

So richten Sie mit der CloudWatch-API einen Alarm ein:

- Rufen Sie die folgende Seite auf `PutMetricAlarm`. Weitere Informationen finden Sie in der Amazon CloudWatch API-Referenz.

Weitere Informationen zum Festlegen von Amazon-SNS-Themen sowie zur Erstellung von Alarmen finden Sie unter Using Amazon CloudWatch alarms (Verwendung von Amazon-CloudWatch-Alarmen).

Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights

Überwachen Sie die Datenbanklast (DB-Last) für Ihre Flotte an DB-Instances von Amazon Aurora mit Database Insights. Die DB-Last misst den Sitzungsaktivitätsgrad in Ihrer Datenbank. Sie können Database Insights verwenden, um die Leistung Ihrer Amazon-Aurora-Datenbanken in großem Umfang zu überwachen.

Mit Database Insights können Sie die DB-Last Ihrer Datenbankflotte visualisieren und die Last nach Wartezuständen, SQL-Anweisungen, Hosts oder Benutzern filtern.

Standardmäßig aktiviert RDS den Standardmodus von Database Insights für Ihre Amazon-Aurora-Datenbanken. Wenn Sie den erweiterten Modus von Database Insights für einen DB-Cluster aktivieren, aktiviert RDS Database Insights für jede DB-Instance im Cluster.

Informationen zur Verwendung von Database Insights in der Amazon-CloudWatch-Konsole finden Sie unter CloudWatch Database Insights im Amazon-CloudWatch-Benutzerhandbuch.

Preisgestaltung

Informationen zur Preisgestaltung finden Sie unter Amazon CloudWatch – Preise.

Themen

- Unterstützung für DB-Engine-, Regions- und Instance-Klassen von Amazon Aurora für Database Insights
- Einschalten des erweiterten Modus von Database Insights für Amazon Aurora
- Aktivieren des Standardmodus von Database Insights für Amazon Aurora
- Konfigurieren Ihrer Datenbank zur Überwachung langsamer SQL-Abfragen mit Database Insights für Amazon Aurora
- Überlegungen zu Database Insights für Amazon Aurora

Unterstützung für DB-Engine-, Regions- und Instance-Klassen von Amazon Aurora für Database Insights

Die folgende Tabelle enthält DB-Engines von Amazon Aurora, die Performance Insights unterstützen.

DB-Engine von Amazon-Aurora	Unterstützte Engine-Versionen und Regionen	Beschränkungen für Instance-Klasse
Amazon Aurora MySQL-Compatible Edition	Weitere Informationen zur Verfügbarkeit von Versionen und Regionen von Database Insights mit Aurora MySQL finden Sie unter Performance-Insights mit Aurora MySQL.	Bei Database Insights gelten die folgenden Beschränkungen für die Engine-Klasse: • db.t2 – nicht unterstützt • db.t3 – nicht unterstützt • „db.t4g.micro“ und „db.t4g.small“ – nicht unterstützt
Amazon Aurora PostgreSQL-Compatible Edition	Weitere Informationen zur Verfügbarkeit von Versionen und Regionen von Database Insights mit Aurora PostgreSQL finden Sie unter Performance-Insights mit Aurora PostgreSQL.	Nicht zutreffend
Aurora PostgreSQL Limitless Database	Weitere Informationen zur Verwendung von Database Insights mit Aurora PostgreSQ L Limitless Database finden Sie unter. Überwachen von Aurora PostgreSQL Limitless Database mit CloudWatch Database Insights	Nicht zutreffend

Database Insights unterstützt Amazon Aurora Serverless v2.

Unterstützung für DB-Engine-, Regions- und Instance-Klassen von Amazon Aurora für Database Insights-Features

Die folgende Tabelle enthält DB-Engines von Amazon Aurora, die Database Insights-Features unterstützen.

Funktion	Preisstufe	Unterstützte Regionen	Unterstützte DB-Engines	Unterstützte Instance-Klassen
SQL-Statistiken für Performance Insights	Alle	Alle	Alle	Alle
Analysieren der Datenbankleistung für einen bestimmten Zeitraum	Nur kostenpflichtige Stufe	Alle	Alle	Alle außer db.serverless (Aurora Serverless v2)
Anzeigen proaktiver Empfehlungen von Performance Insights	Nur kostenpflichtige Stufe	• USA Ost (Ohio) • USA Ost (Nord-Virginia) • USA West (Nordkalifornien) • USA West (Oregon) • Asia Pacific (Mumbai) • Asia Pacific (Seoul) • Asien-Pazifik (Singapur) • Asien-Pazifik (Sydney) • Asien-Pazifik (Tokio) • Canada (Central)	Alle	Alle außer db.serverless (Aurora Serverless v2)

Funktion	Preisstufe	Unterstützte Regionen	Unterstützte DB-Engines	Unterstützte Instance-Klassen
		• Europe (Frankfurt) • Europa (Irland) • Europa (London) • Europe (Paris) • Europa (Stockholm) • Südamerika (São Paulo)		

Unterstützung für Regionen von Amazon Aurora für Database Insights

Aurora unterstützt Database Insights in den folgenden AWS-Regionen.

- USA Ost (Nord-Virginia)
- USA Ost (Ohio)
- USA West (Nordkalifornien)
- USA West (Oregon)
- Africa (Cape Town)
- Asien-Pazifik (Hongkong)
- Asien-Pazifik (Hyderabad)
- Asien-Pazifik (Jakarta)
- Asien-Pazifik (Malaysia)
- Asien-Pazifik (Melbourne)
- Asia Pacific (Mumbai)
- Asien-Pazifik (Osaka)
- Asien-Pazifik (Seoul)
- Asien-Pazifik (Singapur)

- Asien-Pazifik (Sydney)
- Asien-Pazifik (Tokio)
- Canada (Central)
- Kanada West (Calgary)
- Europe (Frankfurt)
- Europa (Irland)
- Europa (London)
- Europa (Mailand)
- Europe (Paris)
- Europa (Spanien)
- Europa (Stockholm)
- Europa (Zürich)
- Israel (Tel Aviv)
- Naher Osten (Bahrain)
- Naher Osten (VAE)
- Südamerika (São Paulo)
- AWS GovCloud (USA-Ost)
- AWS GovCloud (USA-West)

Einschalten des erweiterten Modus von Database Insights für Amazon Aurora

Gehen Sie wie folgt vor, um den erweiterten Modus von Database Insights für Amazon Aurora zu aktivieren.

Aktivieren des erweiterten Modus von Database Insights beim Erstellen eines DB-Clusters

Aktivieren Sie den erweiterten Modus von Database Insights, wenn Sie eine Datenbank für Amazon Aurora erstellen.

Console

In der Konsole können Sie den erweiterten Modus von Database Insights aktivieren, wenn Sie einen DB-Cluster erstellen. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den erweiterten Modus von Database Insights, wenn Sie eine -DB-Cluster über die Konsole erstellen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie Create database (Datenbank erstellen) aus.
4. Wählen Sie im Abschnitt Database Insights Erweiterter Modus aus. Wählen Sie dann eine der folgenden Optionen aus:
   - Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Der Aufbewahrungszeitraum für den erweiterten Modus von Database Insights muss 15 Monate betragen.
   - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
5. Wählen Sie Create database (Datenbank erstellen) aus.

AWS CLI

Um den erweiterten Modus von Database Insights beim Erstellen eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl create-db-cluster auf und geben Sie die folgenden Werte ein:

- `--database-insights-mode advanced`, um den erweiterten Modus von Database Insights zu aktivieren.
- `--engine` – Die Datenbank-Engine für die DB-Instanceden DB-Cluster.
- `--db-cluster-identifier` – Der Bezeichner für den -DB-Cluster.
- `--enable-performance-insights`, um Performance Insights für Database Insights zu aktivieren.

- `--performance-insights-retention-period` – Der Aufbewahrungszeitraum für Daten für -DB-Cluster. Um Database Insights zu aktivieren, muss der Aufbewahrungszeitraum mindestens 465 Tage betragen.

Im folgenden Beispiel wird der erweiterte Modus von Database Insights aktiviert, wenn eine ein DB-Cluster erstellt wird.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
    --database-insights-mode advanced \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Für Windows:

```
aws rds create-db-cluster ^
    --database-insights-mode advanced ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

Um den erweiterten Modus von Database Insights zu aktivieren, wenn Sie -DB-Cluster erstellen, geben Sie die folgenden Parameter für Ihren Amazon-RDS-API-Vorgang CreateDBCluster an.

- `DatabaseInsightsMode` auf `advanced`
- `EnablePerformanceInsights` auf `True`
- `PerformanceInsightsRetentionPeriod` bis mindestens 465 Tage

Aktivieren des erweiterten Modus von Database Insights beim Ändern DB-Clusters

Aktivieren Sie Database Insights, wenn Sie eine Datenbank für Amazon Aurora ändern. Wenn Sie einen DB-Cluster ändern, um den erweiterten Modus von Database Insights zu aktivieren, verursacht dies keine Ausfallzeiten.

> **Note**
>
> Um Database Insights zu aktivieren, muss jede DB-Instance in einem DB-Cluster dieselben Einstellungen für Performance Insights und Enhanced Monitoring aufweisen.

Console

In der Konsole können Sie den erweiterten Modus von Database Insights aktivieren, wenn Sie einen DB-Cluster ändern. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den erweiterten Modus von Database Insights, wenn Sie eine -DB-Cluster über die Konsole ändern

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie eine DB-Instance, und wählen Sie Modify (Ändern) aus.
4. Wählen Sie im Abschnitt Database Insights Erweiterter Modus aus. Wählen Sie dann eine der folgenden Optionen aus:
   - Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Der Aufbewahrungszeitraum für den erweiterten Modus von Database Insights muss 15 Monate betragen.
   - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
5. Klicken Sie auf Weiter.
6. Wählen Sie für Einplanung von Änderungen die Option Sofort anwenden aus. Wenn Sie Anwendung während des nächsten geplanten Wartungsfensters auswählen, ignoriert Ihre Datenbank diese Einstellung und aktiviert den erweiterten Modus von Database Insights sofort.
7. Wählen Sie Cluster ändern aus.

AWS CLI

Um den erweiterten Modus von Database Insights beim Ändern eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl modify-db-cluster auf und geben Sie die folgenden Werte ein:

- `--database-insights-mode advanced`, um den erweiterten Modus von Database Insights zu aktivieren.
- `--db-cluster-identifier` – Der Bezeichner für den DB-Cluster.
- `--enable-performance-insights`, um Performance Insights für Database Insights zu aktivieren.
- `--performance-insights-retention-period` – Der Aufbewahrungszeitraum für Daten für Ihren DB-Cluster. Um den erweiterten Modus von Database Insights zu aktivieren, muss der Aufbewahrungszeitraum mindestens 465 Tage betragen.

Im folgenden Beispiel wird der erweiterte Modus von Database Insights aktiviert, wenn ein DB-Cluster geändert wird.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Für Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

Um den erweiterten Modus von Database Insights zu aktivieren, wenn Sie eine DB-Cluster ändern, geben Sie die folgenden Parameter für Ihren Amazon-RDS-API-Vorgang ModifyDBCluster an.

- `DatabaseInsightsMode` auf `advanced`
- `EnablePerformanceInsights` auf `True`
- `PerformanceInsightsRetentionPeriod` bis mindestens 465 Tage

Aktivieren des Standardmodus von Database Insights für Amazon Aurora

Um den Standardmodus von Database Insights für Amazon Aurora zu aktivieren, verwenden Sie die folgenden Verfahren.

Aktivieren des Standardmodus von Database Insights beim Erstellen von DB-Clustern

Aktivieren Sie den Standardmodus von Database Insights, wenn Sie eine Datenbank für Amazon Aurora erstellen.

Console

In der Konsole können Sie den Standardmodus von Database Insights aktivieren, wenn Sie einen DB-Cluster erstellen. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den Standardmodus von Database Insights, wenn Sie DB-Cluster über die Konsole erstellen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie Create database (Datenbank erstellen) aus.
4. Wählen Sie im Abschnitt Database Insights Standardmodus aus. Wählen Sie dann eine der folgenden Optionen, um Performance Insights zu aktivieren oder zu deaktivieren:
   - Klicken Sie auf Performance Insights aktivieren, um Performance Insights zu deaktivieren.
   - Klicken Sie auf Performance Insights aktivieren, um Performance Insights zu aktivieren. Um Performance Insights zu konfigurieren, geben Sie die folgenden Optionen an:
     - Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Die Aufbewahrungsfrist muss mindestens 7 Tage betragen.
     - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der

Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.

5. Wählen Sie Create database (Datenbank erstellen) aus.

AWS CLI

Um den Standardmodus von Database Insights beim Erstellen eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl create-db-cluster auf und geben Sie die folgenden Werte ein:

- `--database-insights-mode standard`, um den Standardmodus von Database Insights zu aktivieren.
- `--engine` – Die Datenbank-Engine für die den DB-Cluster.
- `--db-cluster-identifier` – Der Bezeichner für den DB-Cluster.
- `--enable-performance-insights` oder `--no-enable-performance-insights`, um Performance Insights zu aktivieren und zu deaktivieren. Wenn Sie `--enable-performance-insights` festlegen, müssen Sie auch den `--performance-insights-retention-period` angeben – Der Aufbewahrungszeitraum für Daten für Ihren DB-Cluster. Die Aufbewahrungsfrist muss mindestens 7 Tage betragen.

Im folgenden Beispiel werden der Standardmodus von Database Insights aktiviert und Performance Insights deaktiviert, wenn ein DB-Cluster erstellt wird.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

Für Windows:

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
```

```
    --performance-insights-retention-period 7
```

Im folgenden Beispiel werden der Standardmodus von Database Insights aktiviert und Performance Insights deaktiviert, wenn ein DB-Cluster erstellt wird.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
    --database-insights-mode standard \
    --engine aurora-postgresql \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

Für Windows:

```
aws rds create-db-cluster ^
    --database-insights-mode standard ^
    --engine aurora-postgresql ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

Um den Standardmodus von Database Insights zu aktivieren, wenn Sie eine einen DB-Cluster erstellen, geben Sie die folgenden Parameter für Ihren Amazon-RDS-API-Vorgang CreateDBCluster an.

- `DatabaseInsightsMode` auf `standard`
- `EnablePerformanceInsights` auf `True` oder `False` Wenn Sie `EnablePerformanceInsights` auf `True` festlegen, müssen Sie `PerformanceInsightsRetentionPeriod` auf mindestens 7 Tage setzen.

Aktivieren des Standardmodus von Database Insights beim Erstellen von DB-Clustern

Aktivieren Sie den Standardmodus von Database Insights, wenn Sie eine Datenbank für Amazon Aurora ändern. Wenn Sie einen DB-Cluster ändern, um den Standardmodus von Database Insights zu aktivieren, verursacht dies keine Ausfallzeiten.

> **Note**
>
> Um Database Insights zu aktivieren, muss jede DB-Instance in einem DB-Cluster dieselben Einstellungen für Performance Insights und Enhanced Monitoring aufweisen.

Console

In der Konsole können Sie den Standardmodus von Database Insights aktivieren, wenn Sie einen DB-Cluster ändern. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den Standardmodus von Database Insights, wenn Sie DB-Cluster über die Konsole ändern

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie eine DB-Instance, und wählen Sie Modify (Ändern) aus.
4. Wählen Sie im Abschnitt Database Insights Standardmodus aus. Wählen Sie dann aus den folgenden Optionen aus:
   - Klicken Sie auf Performance Insights aktivieren, um Performance Insights zu deaktivieren.
   - Klicken Sie auf Performance Insights aktivieren, um Performance Insights zu aktivieren. Um Performance Insights zu konfigurieren, geben Sie die folgenden Optionen an:
     - Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Die Aufbewahrungsfrist muss mindestens 7 Tage betragen.
     - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
5. Klicken Sie auf Weiter.
6. Wählen Sie für Einplanung von Änderungen die Option Sofort anwenden aus. Wenn Sie Anwendung während des nächsten geplanten Wartungsfensters auswählen, ignoriert Ihre Datenbank diese Einstellung und aktiviert den Standardmodus von Database Insights sofort.
7. Wählen Sie Cluster ändern aus.

AWS CLI

Um den Standardmodus von Database Insights beim Ändern eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl modify-db-cluster auf und geben Sie die folgenden Werte ein:

- `--database-insights-mode standard`, um den Standardmodus von Database Insights zu aktivieren.
- `--db-cluster-identifier` – Der Bezeichner für den DB-Cluster.
- `--enable-performance-insights` oder `--no-enable-performance-insights`, um Performance Insights zu aktivieren und zu deaktivieren. Wenn Sie `--enable-performance-insights` festlegen, müssen Sie auch den `--performance-insights-retention-period` angeben – Der Aufbewahrungszeitraum für Daten für DB-Cluster. Die Aufbewahrungsfrist muss mindestens 7 Tage betragen.

Im folgenden Beispiel werden der Standardmodus von Database Insights sowie Performance Insights aktiviert, wenn ein DB-Cluster geändert wird.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

Für Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

Im folgenden Beispiel werden der Standardmodus von Database Insights aktiviert und Performance Insights deaktiviert, wenn ein DB-Cluster geändert wird.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
```

```
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --no-enable-performance-insights
```

Für Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

Um den Standardmodus von Database Insights zu aktivieren, wenn Sie DB-Cluster ändern, geben Sie die folgenden Parameter für Ihren API-Vorgang ModifyDBCluster von Amazon RDS an.

- `DatabaseInsightsMode` auf `standard`
- `EnablePerformanceInsights` auf `True` oder `False` Wenn Sie `EnablePerformanceInsights` auf `True` festlegen, müssen Sie `PerformanceInsightsRetentionPeriod` auf mindestens 7 Tage setzen.

Konfigurieren Ihrer Datenbank zur Überwachung langsamer SQL-Abfragen mit Database Insights für Amazon Aurora

Um langsame SQL-Abfragen für Ihre Datenbank zu überwachen, können Sie den Abschnitt Langsame SQL-Abfragen im Dashboard von Database Insights verwenden. Bevor Sie Ihre Datenbank für die Überwachung langsamer SQL-Abfragen konfigurieren, ist der Abschnitt Langsame SQL-Abfragen leer.

Weitere Informationen zur Überwachung langsamer SQL-Abfragen im Dashboard von Database Insights finden Sie unter Viewing the Database Instance Dashboard for CloudWatch Database Insights im Amazon-CloudWatch-Benutzerhandbuch.

Führen Sie die folgenden Schritte aus, um Ihre Datenbank für die Überwachung langsamer SQL-Abfragen mit Database Insights zu konfigurieren:

1. Aktivieren Sie Protokollexporte in CloudWatch Logs.
2. Erstellen oder ändern Sie die DB-Cluster-Parametergruppe für Ihren DB-Cluster .

Informationen zur Konfiguration von Protokollexporten finden Sie unter Veröffentlichen von Datenbankprotokollen in Amazon CloudWatch Logs im Amazon-Aurora-Benutzerhandbuch.

Weitere Informationen zum Erstellen oder Ändern Ihrer DB-Cluster-Parametergruppe finden Sie in den folgenden Themen.

- Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora
- Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora

Amazon Aurora MySQL

Um Ihren Amazon Aurora MySQL-DB-Cluster für die Überwachung langsamer SQL-Abfragen zu konfigurieren, können Sie die folgende Parameterkombination als Beispiel verwenden:

- `slow_query_log` – festlegen auf `1`
- `long_query_time` – festlegen auf `1.0`
- `log_output` – festlegen auf `FILE`

Dies ist eine mögliche Konfiguration. Eine umfassende Anleitung zu den Protokollparametern für langsame Abfragen von MySQL und zusätzlichen Konfigurationsoptionen finden Sie in der MySQL-Dokumentation zum Protokoll für langsame Abfragen.

Amazon Aurora PostgreSQL

Um Ihren PostgreSQL-DB-Cluster von Amazon Aurora für die Überwachung langsamer SQL-Abfragen zu konfigurieren, können Sie die folgende Parameterkombination als Beispiel verwenden. Beachten Sie, dass die Einstellung dieser Parameter die Leistung Ihres DB-Clusters beeinträchtigen kann.

- `log_min_duration_statement` – festlegen auf `1000`
- `log_statement` – festlegen auf `none`
- `log_destination` – festlegen auf `stderr`

Dies ist eine mögliche Konfiguration. Eine umfassende Anleitung zu PostgreSQL-Protokollparametern und zusätzlichen Konfigurationsoptionen finden Sie in der PostgreSQL-Dokumentation zur Protokollierungskonfiguration.

> **Note**
>
> Für Aurora MySQL können Sie den Parameter `long_query_time` mit einer Granularität von 1 Mikrosekunde konfigurieren. Sie können diesen Parameter beispielsweise auf `0.000001` festlegen. Je nach Anzahl der Abfragen auf der DB-Instance kann der Wert des Parameters `long_query_time` die Leistung verringern. Beginnen Sie mit dem Wert `1.0` und passen Sie ihn an Ihre Workload an. Wenn Sie diesen Parameter auf `0` festlegen, protokolliert Database Insights alle Abfragen.

Informationen zu Protokollen von Aurora MySQL und Aurora PostgreSQL finden Sie im Folgenden.

- Aurora-MySQL-Datenbank-Protokolldateien
- Datenbank-Protokolldateien von Aurora PostgreSQL

Überlegungen zu Database Insights für Amazon Aurora

Im Folgenden finden Sie Überlegungen zu Database Insights für Amazon Aurora.

- Sie können Database Insights nicht für eine DB-Instance in einem DB-Cluster verwalten.
- Um den erweiterten Modus von Database Insights zu aktivieren, müssen Sie Performance Insights aktivieren und den Aufbewahrungszeitraum für Performance Insights auf mindestens 465 Tage (15 Monate) festlegen. Für die Festlegung des Aufbewahrungszeitraums von Performance Insights auf 15 Monate fallen neben den Kosten für Database Insights keine zusätzlichen Kosten an. Weitere Informationen zur Preisgestaltung von Database Insights finden Sie unter Amazon CloudWatch – Preise.
- Um Database Insights zu aktivieren, muss jede DB-Instance in einem DB-Cluster dieselben Einstellungen für Performance Insights und Enhanced Monitoring aufweisen.
- Wenn Sie eine(n) DB-Cluster ändern, um einen der beiden Modi von Database Insights zu aktivieren, verursacht dies keine Ausfallzeiten.

Überwachung mit Performance Insights auf

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.

Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.

Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.

Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

Performance Insights erweitert vorhandene Amazon-Aurora-Überwachungsfunktionen, um die Leistung Ihres Clusters zu veranschaulichen und ihre Analyse zu unterstützen. Mit dem Performance-Insights-Dashboard können Sie die Datenbanklast Ihres Amazon-Aurora-Clusters visualisieren und die Last nach Wartezeiten, SQL-Anweisungen, Hosts oder Benutzern filtern. Weitere Informationen zur Verwendung von Performance Insights mit Amazon DocumentDB finden Sie im Amazon-DocumentDB-Entwicklerhandbuch.

Themen

- Übersicht über Performance-Insights für Amazon Aurora
- Aktivieren und Deaktivieren von Performance Insights für Aurora
- Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL
- Konfigurieren von Zugriffsrichtlinien für Performance Insights
- Analyse der Metriken mit dem Performance-Insights-Dashboard
- Anzeigen proaktiver Empfehlungen von Performance Insights
- Abrufen von Metriken mit der Performance-Insights-API für Aurora
- Protokollieren von Performance-Insights-An AWS CloudTrail
- Performance-Insights-API und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)

Übersicht über Performance-Insights für Amazon Aurora

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.

Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.

Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.

Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-

Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

Standardmäßig aktiviert RDS Performance Insights im Konsolenerstellungsassistenten für Amazon-RDS-Engines. Wenn Sie Performance Insights auf DB-Cluster-Ebene aktivieren, aktiviert RDS Performance Insights für jede DB-Instance im Cluster. Wenn mehr als eine Datenbank auf einer DB-Instance vorhanden ist, aggregiert Performance Insights Leistungsdaten.

Das folgende Video gibt eine Übersicht über Performance Insights für Amazon Aurora.

Verwenden von Performance Insights, um die Leistung von Amazon Aurora-PostgreSQL zu analysieren

Themen

- Datenbanklast
- Maximale CPU
- DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance Insights
- Preisgestaltung und Datenspeicherung für Performance Insights

Datenbanklast

Die Datenbanklast (DB-Last) misst den Sitzungsaktivitätsgrad in Ihrer Datenbank. `DBLoad` ist die wichtigste Metrik in Performance Insights und Performance Insights erfasst jede Sekunde die Datenbanklast.

Themen

- Aktive Sitzungen
- Durchschnittliche aktive Sitzungen
- Durchschnittliche aktive Ausführungen
- Dimensionen

Aktive Sitzungen

Eine Datenbank-Sitzung repräsentiert den Dialog einer Anwendung mit einer relationalen Datenbank. Eine aktive Sitzung ist eine Verbindung, die Arbeit an die DB-Engine gesendet hat und auf eine Antwort wartet.

Eine Sitzung ist aktiv, wenn sie entweder auf der CPU läuft oder darauf wartet, dass eine Ressource verfügbar wird, damit sie fortfahren kann. Beispielsweise kann eine aktive Sitzung warten, bis eine Seite (oder ein Block) in den Speicher eingelesen wird, und verbraucht dann CPU, während sie Daten von der Seite liest.

Durchschnittliche aktive Sitzungen

Die durchschnittlich aktive Sitzungen (AAS) ist die Einheit für die `DBLoad`-Metrik in Performance Insights. Es wird gemessen, wie viele Sitzungen gleichzeitig in der Datenbank aktiv sind.

Jede Sekunde ruft Performance-Insights eine Stichprobe der Anzahl der Sitzungen ab, die gleichzeitig eine Abfrage ausführen. Für jede aktive Sitzung sammelt Performance Insights die folgenden Daten:

- SQL-Anweisung
- Sitzungsstatus (läuft auf der CPU oder wartet)
- Host
- Benutzer, der SQL ausführt

Performance Insights berechnet die AAS durch Dividieren der Gesamtzahl der Sitzungen durch die Gesamtzahl der Stichproben für einen bestimmten Zeitraum. Die folgende Tabelle zeigt beispielsweise 5 aufeinander folgende Stichproben einer laufenden Abfrage, die in Intervallen von 1 Sekunde abgefragt wurden.

Beispiel	Anzahl der Sitzungen, die eine Abfrage ausführen	AAS	Berechnung
1	2	2	2 Sitzungen insgesamt / 1 Stichprobe
2	0	1	2 Sitzungen insgesamt / 2 Stichproben

Beispiel	Anzahl der Sitzungen, die eine Abfrage ausführen	AAS	Berechnung
3	4	2	6 Sitzungen insgesamt / 3 Stichproben
4	0	1.5	6 Sitzungen insgesamt / 4 Stichproben
5	4	2	10 Sitzungen insgesamt / 5 Stichproben

Im vorherigen Beispiel beträgt die DB-Last für das Zeitintervall 2 AAS. Diese Messung bedeutet, dass im Durchschnitt 2 Sitzungen gleichzeitig während des Zeitraums aktiv waren, in dem die 5 Stichproben erfasst wurden.

Durchschnittliche aktive Ausführungen

Die durchschnittlichen aktiven Ausführungen (AAE) pro Sekunde stehen im Zusammenhang mit AAS. Um die AAE zu berechnen, teilt Performance Insights die Gesamtausführungszeit einer Abfrage durch das Zeitintervall. Die folgende Tabelle zeigt die AAE-Berechnung für dieselbe Abfrage in der vorherigen Tabelle.

Verstrichene Zeit	Gesamtausführungszeit	AAE	Berechnung
60	120	2	120 Durchführungssekunden/60 verstrichene Sekunden
120	120	1	120 Durchführungssekunden/120 verstrichene Sekunden
180	380	2.11	380 Ausführungssekunden/180 verstrichene Sekunden

Verstrichene Zeit	Gesamtausführungszeit	AAE	Berechnung
240	380	1.58	380 Ausführungssekunden/240 verstrichene Sekunden
300	600	2	600 Durchführungssekunden/300 verstrichene Sekunden

In den meisten Fällen sind die AAS und die AAE für eine Abfrage ungefähr gleich. Da es sich bei den Eingaben zu den Berechnungen jedoch um unterschiedliche Datenquellen handelt, variieren die Berechnungen häufig geringfügig.

Dimensionen

Die `db.load` Metrik unterscheidet sich von den anderen Zeitreihenmetriken, da Sie sie in Unterkomponenten aufteilen können, die als Dimensionen bezeichnet werden. Sie können sich Dimensionen als „Aufteilungs“-Kategorien für die verschiedenen Merkmale der `DBLoad`-Metrik vorstellen.

Wenn Sie Leistungsprobleme diagnostizieren, sind die folgenden Dimensionen oft am nützlichsten:

Themen

- Warteereignisse
- Haupt-SQL

Eine vollständige Liste der Dimensionen für die Aurora-Engines finden Sie unter DB-Last aufgeteilt nach Dimensionen.

Warteereignisse

Ein Warteereignis bewirkt, dass eine SQL-Anweisung wartet, bis ein bestimmtes Ereignis eintritt, bevor sie mit der Ausführung fortfahren kann. Warteereignisse sind eine wichtige Dimension oder Kategorie für die DB-Last, da sie angeben, wo die Arbeit behindert wird.

Jede aktive Sitzung läuft entweder auf der CPU oder wartet. Sitzungen verbrauchen beispielsweise CPU, wenn sie Speicher nach einem Puffer suchen, eine Berechnung durchführen oder Prozeduralcode ausführen. Wenn Sitzungen keine CPU verbrauchen, warten sie möglicherweise darauf, dass ein Speicherpuffer frei wird, eine Datendatei gelesen oder ein Protokoll geschrieben wird. Je mehr Zeit eine Sitzung auf Ressourcen wartet, desto weniger Zeit läuft sie auf der CPU.

Wenn Sie eine Datenbank tunen, versuchen Sie oft, die Ressourcen herauszufinden, auf die Sitzungen warten. Beispielsweise könnten zwei oder drei Warteereignisse 90 Prozent der DB-Last ausmachen. Diese Maßnahme bedeutet, dass aktive Sitzungen im Durchschnitt die meiste Zeit damit verbringen, auf eine kleine Anzahl von Ressourcen zu warten. Wenn Sie die Ursache dieser Wartezeiten herausfinden können, können Sie eine Lösung versuchen.

Die Warteereignisse variieren je nach DB-Engine:

- Eine Liste der am häufigsten verwendeten Warteereignisse für Aurora MySQL finden Sie unter Aurora-MySQL-Warteereignisse. Um zu erfahren, wie man mit diesen Warteereignissen tunen kann, finden Sie unter Optimieren von Aurora MySQL.
- Informationen zu allen MySQL-Warteereignissen finden Sie unter Wait Event Summary Tables in der MySQL-Dokumentation.
- Eine Liste der am häufigsten verwendeten Warteereignisse für Aurora PostgreSQL finden Sie unter Amazon-Aurora-PostgreSQL-Warteereignisse. Um zu erfahren, wie man mit diesen Warteereignissen tunen kann, finden Sie unter Optimierung mit Warteereignissen für Aurora PostgreSQL.
- Informationen zu allen PostgreSQL-Warteereignissen finden Sie unter Der Statistikkollektor > Warteereignistabellen in der PostgreSQL-Dokumentation.

Haupt-SQL

Während Warteereignisse Engpässe zeigen, zeigt Top-SQL an, welche Abfragen am meisten zur DB-Last beitragen. Beispielsweise könnten derzeit viele Abfragen gleichzeitig in der Datenbank ausgeführt werden, aber eine einzelne Abfrage könnte 99 % der DB-Last verbrauchen. In diesem Fall könnte die hohe Belastung auf ein Problem mit der Abfrage hinweisen.

Standardmäßig zeigt die Performance-Insights-Konsole die wichtigsten SQL-Abfragen an, die zur Datenbanklast beitragen. Die Konsole zeigt auch relevante Statistiken für jede Anweisung an. Um Leistungsprobleme für eine bestimmte Anweisung zu diagnostizieren, können Sie deren Ausführungsplan untersuchen.

Maximale CPU

Das Diagramm Datenbanklast im Dashboard dient zum Erfassen, Aggregieren und Anzeigen von Sitzungsinformationen. Um zu sehen, ob aktive Sitzungen die maximale CPU überschreiten, sehen Sie sich ihre Beziehung zur Max vCPU-Linie an. Performance Insights bestimmt den Wert Max. vCPU anhand der Anzahl der vCPU (virtuellen CPU)-Cores für Ihre DB-Instance. Bei Aurora Serverless v2 stellt Max vCPU (vCPU-Max.) die geschätzte Anzahl von vCPUs dar.

Es kann jeweils ein Prozess auf einer vCPU ausgeführt werden. Wenn die Anzahl der Prozesse die Anzahl der vCPUs übersteigt, werden die Prozesse in die Warteschlange gestellt. Wenn die Warteschlange zunimmt, nimmt die Datenbankleistung ab. Wenn die DB-Last häufig über der Max vCPU-Linie liegt und der primäre Wartezustand „CPU“ lautet, ist die CPU überlastet. In diesem Fall sollten Sie Verbindungen zur Instance drosseln, SQL-Abfragen mit hoher CPU-Last anpassen oder eine größere Instance-Klasse in Betracht ziehen. Hohe und konsistente Instances von Wartezuständen deuten darauf hin, dass es möglicherweise Engpässe oder Probleme mit Ressourcenkonflikten gibt, die behoben werden müssen. Dies kann auch dann zutreffen, wenn die DB-Last die mit Max vCPU definierte Linie nicht überschreitet.

DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance Insights

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.
Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen

in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.
Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

Die folgende Tabelle enthält DB-Engines von Amazon Aurora, die Performance Insights unterstützen.

DB-Engine von Amazon-Aurora	Unterstützte Engine-Versionen und Regionen	Beschränkungen für Instance-Klasse
Amazon Aurora MySQL-Compatible Edition	Weitere Informationen über die Verfügbarkeit von Versionen und Regionen von Performance-Insights mit Aurora MySQL finden Sie unter Performance-Insights mit Aurora MySQL.	Bei Performance Insights gelten die folgenden Beschränkungen für die Engine-Klasse: • db.t2 – nicht unterstützt • db.t3 – nicht unterstützt • „db.t4g.micro“ und „db.t4g.small“ – nicht unterstützt
Amazon Aurora PostgreSQL-Compatible Edition	Weitere Informationen über die Verfügbarkeit von Versionen und Regionen von Performance-Insights mit Aurora PostgreSQL finden Sie unter Performance-Insights mit Aurora PostgreSQL.	N/A

DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance-Insights-Funktionen

Die folgende Tabelle enthält DB-Engines von Amazon Aurora, die Performance-Insights-Funktionen unterstützen.

Funktion	Preisstufe	Unterstützte Regionen	Unterstützte DB-Engines	Unterstützte Instance-Klassen
SQL-Statistiken für Performance Insights	Alle	Alle	Alle	Alle
Analysieren der Datenbankleistung für einen bestimmten Zeitraum	Nur kostenpflichtige Stufe	Alle	Alle	Alle außer db.serverless (Aurora Serverless v2)
Anzeigen proaktiver Empfehlungen von Performance Insights	Nur kostenpflichtige Stufe	• US East (Ohio) • USA Ost (Nord-Virginia) • USA West (Nordkalifornien) • USA West (Oregon) • Asia Pacific (Mumbai) • Asia Pacific (Seoul) • Asien-Pazifik (Singapur) • Asien-Pazifik (Sydney)	Alle	Alle außer db.serverless (Aurora Serverless v2)

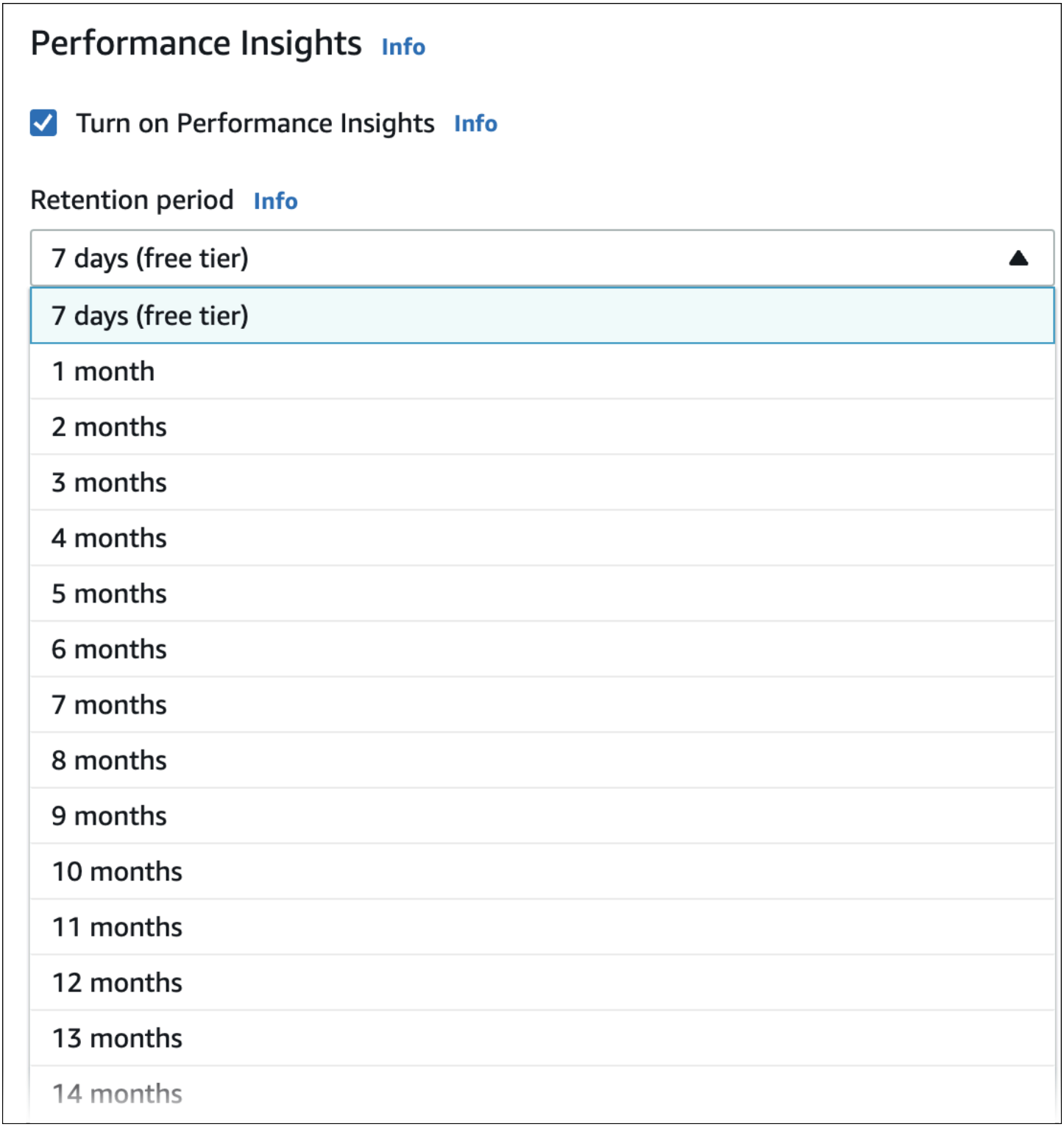
Funktion	Preisstufe	Unterstützte Regionen	Unterstützte DB-Engines	Unterstützte Instance-Klassen
		• Asien-Pazifik (Tokio) • Canada (Central) • Europe (Frankfurt) • Europa (Irland) • Europa (London) • Europe (Paris) • Europa (Stockholm) • Südamerika (São Paulo)		

Preisgestaltung und Datenspeicherung für Performance Insights

Performance Insights bietet standardmäßig 7 Tage Leistungsdatenverlauf und 1 Million API-Anfragen pro Monat. Sie können auch längere Aufbewahrungsfristen erwerben. Umfassende Informationen zur Preisgestaltung finden Sie unter Performance Insights – Preise.

In der RDS-Konsole können Sie einen der folgenden Aufbewahrungszeiträume für Ihre Performance Insights Insights-Daten auswählen:

- Standard (7 Tage)
- ***n***MonateWobei ***n*** eine Zahl von 1–24 ist



Informationen zum Festlegen einer Aufbewahrungsfrist mithilfe der AWS CLI finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora.

> **Note**
>
> Das Stoppen von DB-Clustern mit aktiviertem Performance Insights hat keine Auswirkungen auf die Datenaufbewahrung. Solange DB-Cluster angehalten sind, sammelt Performance Insights keine Daten.

Aktivieren und Deaktivieren von Performance Insights für Aurora

> **Important**
>
> AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
> Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.
> Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.
> Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

Sie können Performance Insights für DB-Cluster beim Erstellen aktivieren. Bei Bedarf können Sie es später deaktivieren, indem Sie Ihre DB-Cluster über die Konsole ändern. Das Aktivieren und Deaktivieren von Performance Insights führt nicht zu Ausfallzeiten, einem Neustart oder einem Failover.

> **Note**
>
> Das Leistungsschema ist ein optionales Leistungstool, das von Aurora MySQL verwendet wird. Nach dem Aktivieren oder Deaktivieren des Leistungsschemas müssen Sie neu starten. Wenn Sie jedoch Performance Insights aktivieren oder deaktivieren, müssen Sie keinen Neustart durchführen. Weitere Informationen finden Sie unter Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL.

Wenn Sie Performance Insights in Verbindung mit globalen Aurora-Datenbanken nutzen, aktivieren Sie Performance Insights individuell für die Datenbanken in jeder AWS-Region. Details hierzu finden Sie unter Überwachung einer Amazon Aurora globalen Datenbank mit Amazon RDS Performance Insights.

Der Performance-Insights-Agent verbraucht eine begrenzte Menge an CPU und Arbeitsspeicher auf dem DB-Host. Wenn die DB-Last hoch ist, begrenzt der Agent die Auswirkungen auf die Leistung, indem Daten seltener erfasst werden.

Console

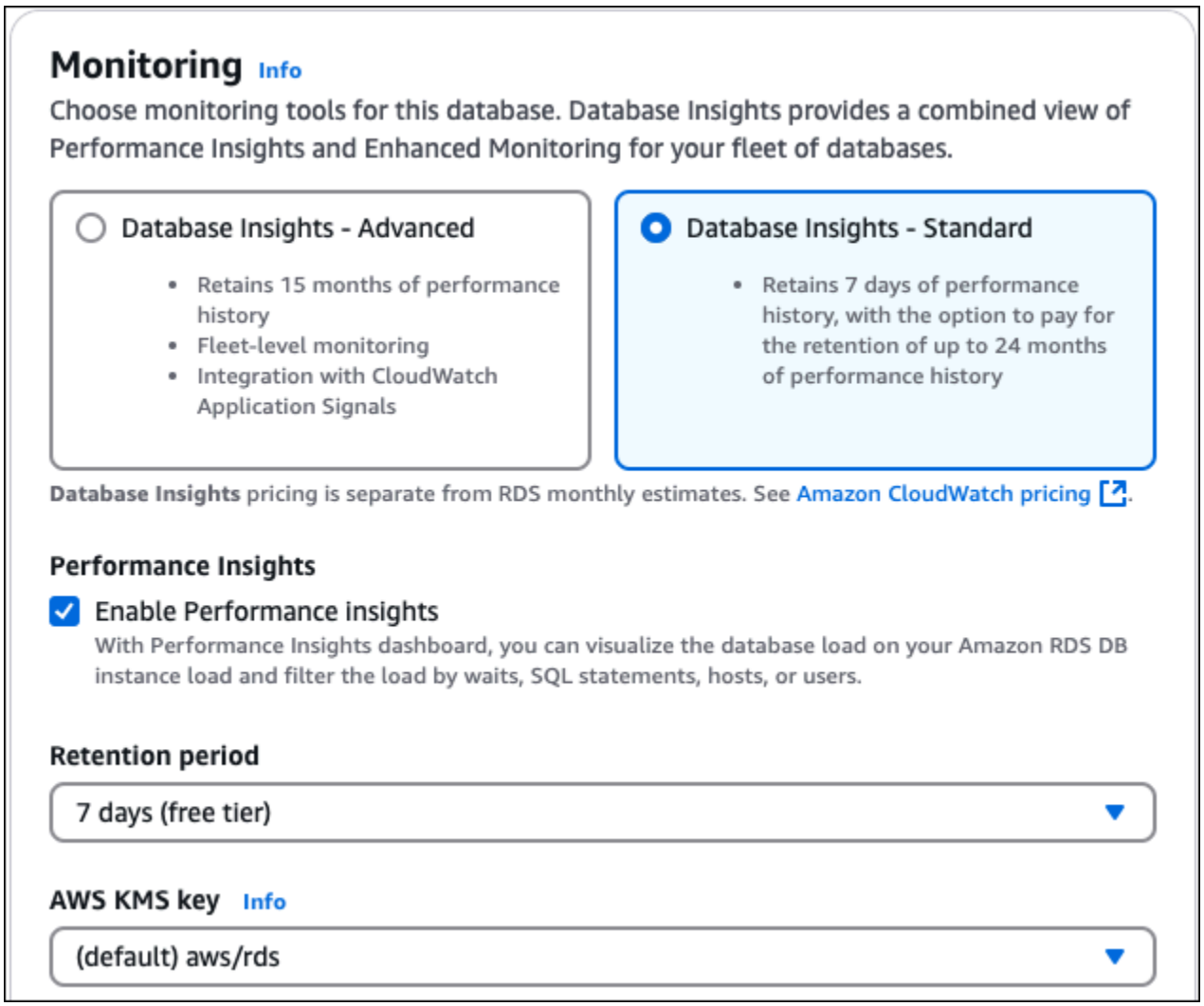
In der Konsole können Sie Performance Insights aktivieren oder deaktivieren, wenn Sie einen DB-Cluster erstellen. Durch die Aktivierung von Performance Insights können Sie Performance-Insights-Einstellungen und -Optionen für Ihren DB-Cluster verwalten. Einstellungen auf Cluster-Ebene gelten für alle DB-Instances im Cluster.

Aktivieren oder Deaktivieren von Performance Insights beim Erstellen von DB-Clustern

Nach der Erstellung neuer DB-Cluster aktiviert Amazon RDS standardmäßig Performance Insights. Um Performance Insights für Ihren DB-Cluster zu deaktivieren, wählen Sie die Option Database Insights – Standard und deaktivieren Sie die Option Performance Insights aktivieren.

Befolgen Sie die Anleitungen für Ihre DB-Engine unter Erstellen eines Amazon Aurora-DB Clusters, um einen DB-Cluster zu erstellen.

Der folgende Screenshot zeigt den Abschnitt Performance-Insights.



Wenn Sie Performance Insights aktivieren wählen, haben Sie die folgenden Optionen:

- Aufbewahrung (nur für den Standardmodus von Database Insights) – Die Dauer, für die Performance-Insights-Daten aufbewahrt werden. Die Aufbewahrungseinstellung ist Standard (7 Tage). Um Ihre Leistungsdaten länger aufzubewahren, geben Sie 1–24 Monate an. Weitere Informationen zum Aufbewahrungszeitraum finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights.
- AWS KMS key— Geben Sie Ihre an AWS KMS key. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Ändern einer AWS KMS -Richtlinie für Performance Insights.

Aktivieren oder Deaktivieren von Performance Insights beim Ändern von DB-Clustern

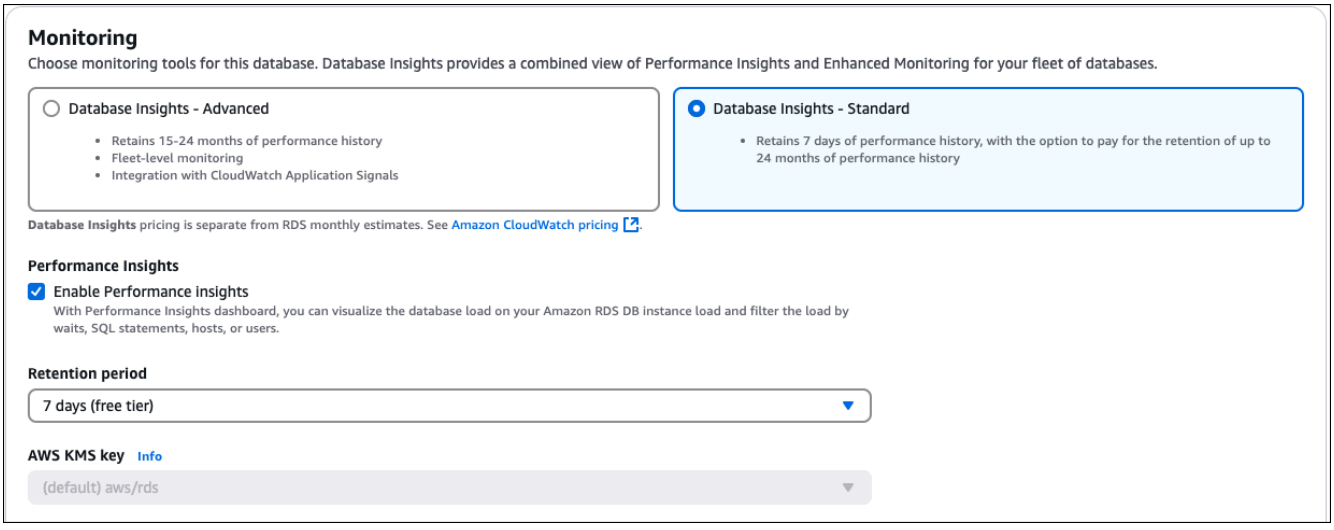
In der Konsole können Sie einen DB-Cluster ändern, um Performance Insights zu verwalten.

So verwalten Sie Performance Insights für DB-Cluster über die Konsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Datenbanken aus.
3. Wählen Sie einen DB-Cluster und dann Ändern aus.
4. Klicken Sie auf Performance Insights aktivieren, um Performance Insights zu aktivieren. Um Performance Insights für Ihren DB-Cluster zu deaktivieren, wählen Sie die Option Database Insights – Standard und deaktivieren Sie die Option Performance Insights aktivieren.

   Wenn Sie Performance Insights aktivieren wählen, haben Sie die folgenden Optionen:

   - Aufbewahrung (nur für den Standardmodus von Database Insights) – Die Dauer, für die Performance-Insights-Daten aufbewahrt werden. Die Aufbewahrungseinstellung ist Standard (7 Tage). Um Ihre Leistungsdaten länger aufzubewahren, geben Sie 1–24 Monate an. Weitere Informationen zum Aufbewahrungszeitraum finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights.
   - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.



5. Klicken Sie auf Weiter.

6. Wählen Sie für Einplanung von Änderungen die Option Sofort anwenden aus. Wenn Sie im nächsten geplanten Wartungsfenster die Option Anwenden wählen, ignoriert Ihre Instance diese Einstellung und aktiviert Performance Insights sofort.
7. Wählen Sie Modify instance (Instance ändern).

AWS CLI

Wenn Sie den create-db-instance AWS CLI Befehl verwenden, aktivieren Sie Performance Insights, indem Sie entweder `advanced` oder angeben `--enable-performance-insights` und `--database-insights-mode` darauf setzen`standard`. Um Performance Insights zu deaktivieren, geben Sie `--no-enable-performance-insights` an und legen Sie `database-insights-mode` auf `standard` fest.

Sie können diese Werte auch mit den folgenden AWS CLI Befehlen angeben:

- create-db-cluster
- modify-db-cluster
- create-db-instance-read-replikat
- modify-db-instance
- restore-db-instance-from-3

Um Performance Insights für einen DB-Cluster mit dem zu verwalten AWS CLI

- Rufen Sie den modify-db-cluster AWS CLI Befehl auf und geben Sie die folgenden Werte ein:
  - `--db-cluster-identifier` – Der Name der DB-Instancein Ihrem DB-Cluster.
  - `--enable-performance-insights` zum Aktivieren oder `--no-enable-performance-insights` zum Deaktivieren
  - `database-insights-mode` – Der Modus von Database Insights für DB-Cluster. Um Performance Insights zu deaktivieren, legen Sie diesen Wert auf `standard` fest.

  Das folgende Beispiel aktiviert Performance Insights für `sample-db-cluster`.

  Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
```

```
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights
```

Für Windows:

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights
```

Wenn Sie Performance Insights in der CLI aktivieren, können Sie mit der `--performance-insights-retention-period`-Option optional die Anzahl der Tage angeben, die Performance-Insights-Daten aufbewahrt werden sollen. Sie können angeben7, *`month`* * 31 (wobei *`month`* es sich um eine Zahl zwischen 1—23 handelt) oder731. Wenn Sie beispielsweise Ihre Leistungsdaten 3 Monate lang aufbewahren möchten, geben Sie 93 an, was 3 * 31 ist. Der Standardwert ist 7 Tage. Weitere Informationen zum Aufbewahrungszeitraum finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights.

Das folgende Beispiel aktiviert Performance Insights für `sample-db-cluster` und legt fest, dass Performance-Insights-Daten für 93 Tage (3 Monate) aufbewahrt werden.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

Für Windows:

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

Wenn Sie eine Aufbewahrungsfrist wie 94 Tage angeben, was kein gültiger Wert ist, gibt RDS einen Fehler aus.

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance
 operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93,
 124, 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

> **Note**
>
> Sie können Performance Insights nur für eine Instance in einem DB-Cluster umschalten, in der Performance Insights nicht auf Cluster-Ebene verwaltet wird.

RDS API

Wenn Sie eine neue DB-Instance in Ihrem DB-Cluster mithilfe des Amazon RDS-API-Vorgangs „DBInstanceVorgang erstellen“ erstellen, aktivieren Sie Performance Insights, indem Sie `EnablePerformanceInsights` auf einstellen`True`. Um Performance Insights zu deaktivieren, legen Sie `EnablePerformanceInsights` auf `False` und `DatabaseInsightsMode` auf `standard` fest.

Sie können den Wert für `EnablePerformanceInsights` auch mittels der folgenden API-Operationen angeben:

- CreateDBCluster
- Modifizieren SieDBCluster
- Modifizieren SieDBInstance
- CreateDBInstanceReadReplica
- Aus S3 wiederherstellen DBInstance

Wenn Sie Performance Insights aktivieren, können Sie optional mit dem Parameter `PerformanceInsightsRetentionPeriod` den Zeitraum in Tagen angeben, wie lange Performance-Insights-Daten gespeichert werden sollen. Sie können angeben7, *`month`* * 31 (wobei *`month`* es sich um eine Zahl zwischen 1—23 handelt) oder. `731` Wenn Sie beispielsweise Ihre Leistungsdaten 3 Monate lang aufbewahren möchten, geben Sie `93` an, was 3 * 31 ist. Der Standardwert ist 7 Tage. Weitere Informationen zum Aufbewahrungszeitraum finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights.

Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL

Das Leistungsschema ist eine optionale Funktion zur Überwachung der Laufzeitleistung von Aurora MySQL auf niedriger Detailebene. Das Leistungsschema ist so konzipiert, dass es minimale Auswirkungen auf die Datenbankleistung hat. Performance Insights ist eine separate Funktion, die Sie mit oder ohne Leistungsschema verwenden können.

Themen

- Überblick über das Leistungsschema-Objekt
- Performance Insights und das Performance-Schema
- Automatische Verwaltung des Leistungsschemas durch Performance Insights
- Auswirkung eines Neustarts auf das Leistungsschema
- Feststellen, ob Performance Insights das Leistungsschema verwaltet
- Aktivieren des Leistungsschemas für Performance Insights in Aurora MySQL

Überblick über das Leistungsschema-Objekt

Das Leistungsschema überwacht Ereignisse in Aurora MySQL-Datenbanken. Ein Ereignis ist eine Datenbankserver-Aktion, die Zeit in Anspruch nimmt und instrumentiert wurde, damit Timing-Informationen erfasst werden können. Nachfolgend finden Sie Beispiele für Ereignisse:

- Funktionsaufrufe
- Wartet auf das Betriebssystem
- Phasen der SQL-Ausführung
- Gruppen von SQL-Anweisungen

Die `PERFORMANCE_SCHEMA`-Speicher-Engine ist ein Mechanismus zur Implementierung der Leistungsschema-Funktion. Diese Engine sammelt Ereignisdaten mithilfe der Instrumentierung im Quellcode der Datenbank. Die Engine speichert Ereignisse in Arbeitsspeichertabellen in der `performance_schema`-Datenbank. Sie können `performance_schema` genauso abfragen, wie Sie andere Tabellen abfragen können. Weitere Informationen finden Sie unter MySQL-Leistungsschema im MySQL-Referenzhandbuch.

Performance Insights und das Performance-Schema

Performance Insights und das Leistungsschema sind separate Funktionen, die jedoch verbunden sind. Das Verhalten von Performance Insights für Aurora MySQL hängt davon ab, ob das Performance-Schema aktiviert ist und wenn ja, ob Performance Insights das Performance-Schema automatisch verwaltet. Die folgende Tabelle beschreibt das Verhalten.

Performance-Schema ist aktiviert	Performance-Insights-Verwaltungsmodus	Performance-Insights-Verhalten
Ja	Automatisch	• Sammelt detaillierte Überwachungsinformationen auf niedriger Ebene • Sammelt jede Sekunde aktive Sitzungsmetriken • Zeigt die DB-Last kategorisiert nach detaillierten Warteereignissen an, mit denen Sie Engpässe identifizieren können
Ja	Manuell	• Erfasst Warteereignisse und Metriken pro SQL • Sammelt jede Sekunde aktive Sitzungsmetriken • Meldet Benutzerzustände wie Einfügen und Senden, mit denen Sie Engpässe nicht identifizieren können
Nein	N/A	• Erfasst keine Warteereignisse, Metriken pro SQL oder andere detaillierte Überwachungsinformationen auf niedriger Ebene • Sammelt aktive Sitzungsmetriken alle fünf Sekunden statt jede Sekunde • Meldet Benutzerzustände wie Einfügen und Senden, mit denen Sie Engpässe nicht identifizieren können

Automatische Verwaltung des Leistungsschemas durch Performance Insights

Wenn Sie eine Aurora-MySQL-DB-Instance erstellen und Performance Insights aktiviert ist, wird das Leistungsschema ebenfalls aktiviert. In diesem Fall verwaltet Performance Insights Ihre Parameter des Leistungsschemas automatisch. Dies ist die empfohlene Konfiguration.

Wenn Performance Insights das Leistungsschema automatisch verwaltet, ist die Quelle von `performance_schema` `System default`.

> **Note**
>
> Die automatische Verwaltung des Leistungsschemas wird für die Instance-Klasse t4g.medium nicht unterstützt.

Sie können das Leistungsschema auch manuell verwalten. Wenn Sie diese Option auswählen, stellen Sie die Parameter entsprechend den Werten in der folgenden Tabelle ein.

Parametername	Parameterwert
`performance_schema`	1 (Die Spalte Source (Quelle) hat den Wert `Modified`)
`performance-schema-consumer-events-waits-current`	`ON`
`performance-schema-instrument`	`wait/%=ON`
`performance_schema_consumer_global_instrumentation`	1
`performance_schema_consumer_thread_instrumentation`	1

Wenn Sie den Parameterwert `performance_schema` manuell ändern und später zur automatischen Verwaltung zurückkehren möchten, finden Sie weitere Informationen unter Aktivieren des Leistungsschemas für Performance Insights in Aurora MySQL.

> **Important**
>
> Beim Aktivieren des Leistungsschemas durch Performance Insights werden die Parametergruppenwerte nicht geändert. Die Werte werden jedoch für die DB-Instances geändert, die ausgeführt werden. Die geänderten Werte können als einzige Möglichkeit über den Befehl `SHOW GLOBAL VARIABLES` eingesehen werden.

Auswirkung eines Neustarts auf das Leistungsschema

Performance Insights und das Leistungsschema unterscheiden sich in ihren Anforderungen für Neustarts der DB-Instance:

Leistungsschema

Sie müssen die DB-Instance neu starten, um diese Funktion ein- oder auszuschalten.

Performance Insights

Sie müssen die DB-Instance nicht neu starten, um diese Funktion ein- oder auszuschalten.

Wenn das Leistungsschema derzeit nicht aktiviert ist und Sie Performance Insights aktivieren, ohne die DB-Instance neu zu starten, wird das Leistungsschema nicht aktiviert.

Feststellen, ob Performance Insights das Leistungsschema verwaltet

Sehen Sie sich die folgende Tabelle an, um herauszufinden, ob Performance Insights derzeit das Leistungsschema für alle unterstützten Engine-Hauptversionen verwaltet.

Einstellung des performance_schema-Parameters	Einstellung der Spalte „Source“ (Quelle)	Performance Insights verwaltet das Leistungsschema?
`0`	`System default`	Ja
`0` oder `1`	`Modified`	Nein

Mit dem folgenden Verfahren stellen Sie fest, ob Performance Insights das Leistungsschema automatisch verwaltet.

So stellen Sie fest, ob Performance Insights das Leistungsschema automatisch verwaltet

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Parameter groups (Parametergruppen).
3. Wählen Sie den Namen der Parametergruppe für Ihre DB-Instance.
4. Geben Sie im Suchfeld **`performance_schema`** ein.
5. Überprüfen Sie, ob Quelle dem Systemstandardwert entspricht und für Wert 0 festgelegt ist. Ist das der Fall, verwaltet Performance Insights das Leistungsschema automatisch.

   Im hier gezeigten Beispiel verwaltet Performance Insights das Leistungsschema nicht automatisch.

Aktivieren des Leistungsschemas für Performance Insights in Aurora MySQL

Angenommen, Performance Insights ist für Ihre DB-Instance aktiviert, verwaltet aber derzeit nicht das Leistungsschema. Wenn Sie zulassen möchten, dass Performance Insights das Leistungsschema automatisch verwaltet, führen Sie die folgenden Schritte aus.

So konfigurieren Sie das Leistungsschema für die automatische Verwaltung

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Parameter groups (Parametergruppen).
3. Wählen Sie den Namen der Parametergruppe für Ihre DB-Instance.
4. Wählen Sie Bearbeiten aus.
5. Geben Sie im Suchfeld **`performance_schema`** ein.
6. Wählen Sie den `performance_schema`-Parameter aus.
7. Wählen Sie Auf Standardwert setzen.
8. Bestätigen Sie, indem Sie Werte auf Standardwert zurücksetzen auswählen.
9. Wählen Sie Save Changes.

10. Starten Sie die DB-Instance neu.

> **Important**
>
> Wenn Sie das Leistungsschema aktivieren oder deaktivieren, müssen Sie die DB-Instance unbedingt neu starten.

Weitere Informationen zum Ändern von Instance-Parametern finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora. Weitere Informationen zu den Seiten im Dashboard finden Sie unter Analyse der Metriken mit dem Performance-Insights-Dashboard. Weitere Informationen zum MySQL-Leistungsschema finden Sie in der MySQL-Dokumentation unter MySQL Performance Schema (für 8.0) und MySQL Performance Schema (für 8.4).

Konfigurieren von Zugriffsrichtlinien für Performance Insights

Um auf Performance Insights zugreifen zu können, muss ein Principal über die entsprechenden Berechtigungen von AWS Identity and Access Management (IAM) verfügen.

> **Note**
>
> Um Performance Insights mit einem vom Kunden verwalteten Schlüssel zu verwenden, gewähren Sie Benutzern die `kms:GenerateDataKey` Berechtigungen `kms:Decrypt` und für Ihren AWS AWS KMS Schlüssel.

Greifen Sie mit diesen Methoden auf Performance Insights zu:

- Hängen Sie die `AmazonRDSPerformanceInsightsReadOnly` verwaltete Richtlinie für den schreibgeschützten Zugriff an
- Hängen Sie die `AmazonRDSPerformanceInsightsFullAccess` verwaltete Richtlinie für den Zugriff auf alle Operationen der Performance Insights Insights-API an
- Erstellen Sie eine benutzerdefinierte IAM-Richtlinie mit bestimmten Berechtigungen
- AWS KMS Berechtigungen für verschlüsselte Performance Insights Insights-Daten konfigurieren
- Richten Sie mithilfe von Berechtigungen auf Ressourcenebene einen differenzierten Zugriff ein
- Verwenden Sie die Tag-basierte Zugriffskontrolle, um Berechtigungen mithilfe von Ressourcen-Tags zu verwalten

Anfügen der Richtlinie AmazonRDSPerformanceInsightsReadOnly an einen IAM-Prinzipal

`AmazonRDSPerformanceInsightsReadOnly`ist eine AWS verwaltete Richtlinie, die Zugriff auf alle schreibgeschützten Operationen der Amazon RDS Performance Insights-API gewährt.

Wenn Sie eine Verbindung `AmazonRDSPerformanceInsightsReadOnly` zu einem Berechtigungssatz oder einer Rolle herstellen, müssen Sie auch die folgenden CloudWatch Berechtigungen anhängen:

- `GetMetricStatistics`
- `ListMetrics`
- `GetMetricData`

Mit diesen Berechtigungen kann der Empfänger Performance Insights mit anderen Konsolenfunktionen verwenden.

Weitere Informationen zu CloudWatch Berechtigungen finden Sie in der CloudWatch Amazon-Berechtigungsreferenz.

Mehr über `AmazonRDSPerformanceInsightsReadOnly` erfahren Sie unter Von AWS verwaltete Richtlinie: AmazonRDSPerformanceInsightsReadOnly.

Anfügen der Richtlinie AmazonRDSPerformanceInsightsFullAccess an einen IAM-Prinzipal

`AmazonRDSPerformanceInsightsFullAccess`ist eine AWS verwaltete Richtlinie, die Zugriff auf alle Operationen der Amazon RDS Performance Insights API gewährt.

Wenn Sie eine Verbindung `AmazonRDSPerformanceInsightsFullAccess` zu einem Berechtigungssatz oder einer Rolle herstellen, müssen Sie auch die folgenden CloudWatch Berechtigungen anhängen:

- `GetMetricStatistics`
- `ListMetrics`
- `GetMetricData`

Mit diesen Berechtigungen kann der Empfänger Performance Insights mit anderen Konsolenfunktionen verwenden.

Weitere Informationen zu CloudWatch Berechtigungen finden Sie in der CloudWatch Amazon-Berechtigungsreferenz.

Mehr über `AmazonRDSPerformanceInsightsFullAccess` erfahren Sie unter AWS-verwaltete Richtlinie: AmazonRDSPerformanceInsightsFullAccess.

Erstellen einer benutzerdefinierten IAM-Richtlinie für Performance Insights

Für Benutzer, die nicht über die Richtlinie `AmazonRDSPerformanceInsightsReadOnly` oder `AmazonRDSPerformanceInsightsFullAccess` verfügen, können Sie den Zugriff auf Performance Insights gewähren, indem Sie eine benutzerverwaltete IAM-Richtlinie erstellen oder ändern. Wenn Sie diese Richtlinie an einen IAM-Berechtigungssatz oder eine Rolle anfügen, kann der Empfänger Performance Insights verwenden.

Erstellen eine benutzerdefinierten Richtlinie

1. Öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich Richtlinien.
3. Wählen Sie Richtlinie erstellen aus.
4. Wählen Sie auf der Seite Richtlinie erstellen die Option JSON aus.
5. Kopieren Sie den angegebenen Text in den Abschnitt JSON-Richtliniendokument im AWS Managed Policy Reference Guide für die Richtlinie AmazonRDSPerformanceInsightsReadOnly oder AmazonRDSPerformanceInsightsFullAccess.
6. Wählen Sie Richtlinie prüfen.
7. Geben Sie einen Namen und optional eine Beschreibung für die Richtlinie an und wählen Sie dann Create policy (Richtlinie erstellen) aus.

Sie können die Richtlinie nun an einen Berechtigungssatz oder eine Rolle anfügen. Das folgende Verfahren setzt voraus, dass Sie für diesen Zweck bereits einen Benutzer zur Verfügung haben.

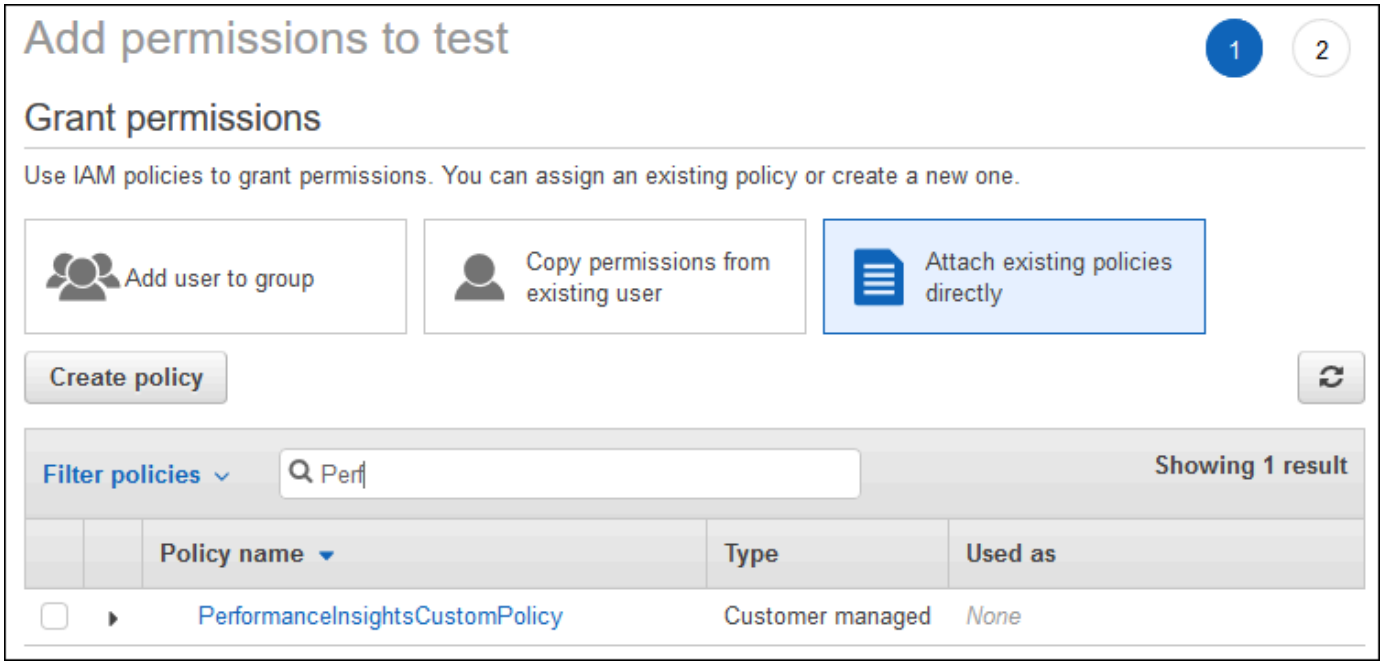
So fügen Sie die Richtlinie an einen Benutzer an

1. Öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Klicken Sie im Navigationsbereich auf Users (Benutzer).

3. Wählen Sie einen vorhandenen Benutzer aus der Liste aus.

> **Important**
>
> Um Performance Insights verwenden zu können, benötigen Sie zusätzlich zur benutzerdefinierten Richtlinie Zugriff auf Amazon RDS. Beispielsweise bietet die vordefinierte Richtlinie `AmazonRDSPerformanceInsightsReadOnly` schreibgeschützten Zugriff auf Amazon RDS. Weitere Informationen finden Sie unter Verwalten des Zugriffs mit Richtlinien.

4. Wählen Sie auf der Seite Übersicht die Option Add permissions (Berechtigungen hinzufügen) aus.
5. Wählen Sie Attach existing policies directly (Vorhandene Richtlinien direkt zuordnen). Geben Sie in Suchen die ersten Zeichen Ihres Richtliniennamens ein, wie in der folgenden Abbildung gezeigt.



6. Wählen Sie Ihre Richtlinie und wählen Sie anschließend Nächster Schritt: Prüfen.
7. Wählen Sie Add permissions (Berechtigungen hinzufügen) aus.

Ändern einer AWS KMS -Richtlinie für Performance Insights

Performance Insights verwendet an AWS KMS key , um sensible Daten zu verschlüsseln. Wenn Sie Performance Insights über die API oder die Konsole aktivieren, haben Sie folgende Möglichkeiten:

- Wählen Sie die Standardeinstellung Von AWS verwalteter Schlüssel.

  Amazon RDS verwendet die Von AWS verwalteter Schlüssel für Ihre neue DB-Instance. Amazon RDS erstellt einen Von AWS verwalteter Schlüssel für Ihr AWS-Konto. Ihr AWS-Konto hat für jeden ein anderes Von AWS verwalteter Schlüssel für Amazon RDS AWS-Region.
- Wählen Sie einen kundenverwalteten Schlüssel.

  Wenn Sie einen vom Kunden verwalteten Schlüssel angeben, benötigen Benutzer in Ihrem Konto, die die Performance Insights API aufrufen, die Berechtigungen `kms:Decrypt` und `kms:GenerateDataKey` für den KMS-Schlüssel. Sie können diese Berechtigungen über IAM-Richtlinien konfigurieren. Wir empfehlen jedoch, dass Sie diese Berechtigungen über Ihre KMS-Schlüsselrichtlinie verwalten. Weitere Informationen finden Sie unter Schlüsselrichtlinien in AWS KMS im Entwicklerhandbuch für AWS Key Management Service .

Example

Das folgende Beispiel zeigt, wie Sie Ihrer KMS-Schlüsselrichtlinie Anweisungen hinzufügen können. Diese Anweisungen erlaubt den Zugriff auf Performance Insights. Je nachdem, wie Sie den KMS-Schlüssel verwenden, möchten Sie möglicherweise einige Einschränkungen ändern. Bevor Sie Ihrer Richtlinie Anweisungen hinzufügen, entfernen Sie alle Kommentare.

JSON

```
{
    "Version":"2012-10-17",
    "Id" : "your-policy",
    "Statement" : [
        {
            "Sid" : "AllowViewingRDSPerformanceInsights",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/Role1"
                ]
                },
             "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
                ],
            "Resource": "*",
```

```
            "Condition" : {
            "StringEquals" : {
                "kms:ViaService" : "rds.us-east-1.amazonaws.com"
                },
            "ForAnyValue:StringEquals": {
                "kms:EncryptionContext:aws:pi:service": "rds",
                "kms:EncryptionContext:service": "pi",
                "kms:EncryptionContext:aws:rds:db-id": "db-
AAAAABBBBBCCCCDDDDDEEEEE"
                }
            }
        }
    ]
}
```

So verwendet Performance Insights vom AWS KMS Kunden verwaltete Schlüssel

Performance Insights verwendet vom Kunden verwaltete Schlüssel, um vertrauliche Daten zu verschlüsseln. Wenn Sie Performance Insights aktivieren, können Sie einen AWS KMS -Schlüssel über die API bereitstellen. Performance Insights erstellt AWS KMS Berechtigungen für diesen Schlüssel. Das Feature verwendet den Schlüssel und führt die erforderlichen Operationen aus, um vertrauliche Daten zu verarbeiten. Zu den vertraulichen Daten gehören Felder wie Benutzer, Datenbank, Anwendung und SQL-Abfragetext. Performance Insights stellt sicher, dass die Daten sowohl im Ruhezustand als auch während der Übertragung verschlüsselt bleiben.

So arbeitet Performance Insights IAM mit AWS KMS

IAM erteilt Berechtigungen für bestimmte. APIs Performance Insights hat die folgenden öffentlichen Bereiche APIs, die Sie mithilfe von IAM-Richtlinien einschränken können:

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`
- `ListAvailableResourceMetrics`

Sie können die folgenden API-Abfragen verwenden, um vertrauliche Daten abzurufen.

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

Wenn Sie die API verwenden, um vertrauliche Daten abzurufen, nutzt Performance Insights die Anmeldeinformationen des Aufrufers. Diese Überprüfung stellt sicher, dass der Zugriff auf vertrauliche Daten auf Benutzer beschränkt ist, die Zugriff auf den KMS-Schlüssel haben.

Wenn Sie diese aufrufen APIs, benötigen Sie Berechtigungen zum Aufrufen der API über die IAM-Richtlinie und Berechtigungen zum Aufrufen der `kms:decrypt` Aktion über die AWS KMS Schlüsselrichtlinie.

Die API `GetResourceMetrics` kann sowohl vertrauliche als auch nicht vertrauliche Daten zurückgeben. Die Anforderungsparameter bestimmen, ob die Antwort vertrauliche Daten enthalten soll. Die API gibt vertrauliche Daten zurück, wenn die Anfrage eine vertrauliche Dimension im Filter- oder Group-by-Parameter enthält.

Weitere Informationen zu den Dimensionen, die Sie mit der `GetResourceMetrics` API verwenden können, finden Sie unter. DimensionGroup

Example Beispiele

Im folgenden Beispiel werden die vertraulichen Daten für die Gruppe `db.user` angefordert:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
  "ServiceType": "RDS",
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
```

```
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

Im folgenden Beispiel werden die nicht vertraulichen Daten für die Metrik `db.load.avg` angefordert:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

Gewähren von differenziertem Zugriff für Performance Insights

Eine differenzierte Zugriffskontrolle bietet zusätzliche Möglichkeiten, den Zugriff auf Performance Insights zu steuern. Diese Zugriffskontrolle kann den Zugriff auf einzelne Dimensionen für die Performance-Insights-Aktionen `GetResourceMetrics`, `DescribeDimensionKeys` und `GetDimensionKeyDetails` zulassen oder verweigern. Um einen differenzierten Zugriff zu verwenden, geben Sie Dimensionen in der IAM-Richtlinie mithilfe von Bedingungsschlüsseln an. Die Auswertung des Zugriffs folgt der Bewertungslogik der IAM-Richtlinie. Weitere Informationen finden Sie unter Logik der Richtlinienauswertung im Handbuch zu -IAM-Benutzer. Wenn in der IAM-Richtlinienanweisung keine Dimension angegeben ist, steuert die Anweisung den Zugriff auf alle Dimensionen für die angegebene Aktion. Eine Liste der verfügbaren Dimensionen finden Sie unter DimensionGroup.

Um herauszufinden, auf welche Dimensionen Ihre Anmeldeinformationen zugreifen dürfen, verwenden Sie den Parameter `AuthorizedActions` in `ListAvailableResourceDimensions` und geben Sie die Aktion an. Die folgenden Werte sind für `AuthorizedActions` zulässig:

- `GetResourceMetrics`
- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`

Wenn Sie `GetResourceMetrics` für den Parameter `AuthorizedActions` angeben, gibt `ListAvailableResourceDimensions` die Liste der Dimensionen zurückgegeben, auf die die Aktion `GetResourceMetrics` zugreifen darf. Wenn Sie im Parameter `AuthorizedActions` mehrere Aktionen angeben, gibt `ListAvailableResourceDimensions` einen Schnittpunkt von Dimensionen zurück, auf die diese Aktionen zugreifen dürfen.

Example

Das folgende Beispiel bietet Zugriff auf die angegebenen Dimensionen für die Aktionen `GetResourceMetrics` und `DescribeDimensionKeys`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
        {
            "Sid": "SingleAllow",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.id",
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }


    ]
}
```

Im Folgenden finden Sie die Antwort für die angeforderte Dimension:

```
 // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
```

```
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                  //  { "Identifier": "db.sql_tokenized.db_id" }, // not included
 because not allows in the IAM Policy
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            }

        ] }
    ]
}
```

Im folgenden Beispiel werden eine Zulassung und zwei Ablehnungen für den Zugriff auf die Dimensionen festgelegt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
          {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
```

```
          },

          {
            "Sid": "O01AllowAllWithoutSpecifyingDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },

        {
            "Sid": "O01DenyAppDimensionForAll",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.application.name"
                    ]
                }
            }
        },

        {
            "Sid": "O01DenySQLForGetResourceMetrics",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
```

```
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }
    ]
}
```

Im Folgenden finden Sie die Antwort für die angeforderten Dimensionen:

```
   // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["GetResourceMetrics"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
```

```
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
}
```

```
// ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // allowed for DescribeDimensionKeys because our IAM Policy
                  // denies it only for GetResourceMetrics
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
```

```
            },
            ...
        ] }
    ]
}
```

Tag-basierte Zugriffskontrolle für Performance Insights verwenden

Sie können den Zugriff auf Performance Insights Insights-Metriken mithilfe von Tags steuern, die von der übergeordneten DB-Instance geerbt wurden. Verwenden Sie IAM-Richtlinien, um den Zugriff auf Performance Insights Insights-Operationen zu steuern. Diese Richtlinien können die Tags auf Ihrer DB-Instance überprüfen, um die Berechtigungen zu ermitteln.

So funktionieren Tags mit Performance Insights

Performance Insights wendet automatisch Ihre DB-Instance-Tags an, um Performance Insights Insights-Metriken zu autorisieren. Wenn Sie Ihrer DB-Instance Tags hinzufügen, können Sie diese Tags sofort verwenden, um den Zugriff auf Performance Insights Insights-Daten zu steuern.

- Um Tags für Performance Insights Insights-Metriken hinzuzufügen oder zu aktualisieren, ändern Sie die Tags auf Ihrer DB-Instance.
- Um Tags für Performance Insights-Metriken anzuzeigen, rufen Sie die Metrik-Ressource Performance Insights `ListTagsForResource` auf. Sie gibt die Tags aus der DB-Instance zurück, die der Metrik zugeordnet ist.

> **Note**
>
> Die `UntagResource` Operationen `TagResource` und geben einen Fehler zurück, wenn Sie versuchen, sie direkt für Performance Insights Insights-Metriken zu verwenden.

Tag-basierte IAM-Richtlinien erstellen

Um den Zugriff auf Performance Insights Insights-Operationen zu steuern, verwenden Sie den `aws:ResourceTag` Bedingungsschlüssel in Ihren IAM-Richtlinien. Diese Richtlinien überprüfen die Tags auf Ihrer DB-Instance.

Example

Diese Richtlinie verhindert den Zugriff auf Performance Insights Insights-Metriken für Produktionsdatenbanken. Die Richtlinie verweigert den `pi:GetResourceMetrics` Vorgang in Performance Insights für alle mit `env:prod` markierten Datenbankressourcen.

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "pi:GetResourceMetrics",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/env": "prod"
                }
            }
        }
    ]
}
```

Analyse der Metriken mit dem Performance-Insights-Dashboard

Important

AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. Juni 2026. Nach diesem Datum unterstützt Amazon RDS die Performance-Insights-Konsole, die flexiblen Aufbewahrungsfristen (1–24 Monate) und die damit verbundenen Preise nicht mehr. Die Performance-Insights-API besteht jedoch weiterhin ohne Preisänderungen. Die Kosten für die Performance Insights Insights-API werden zusammen mit den Kosten für CloudWatch Database Insights in Ihrer AWS Rechnung ausgewiesen.
Wir empfehlen, dass Sie alle , die die kostenpflichtige Stufe von Performance Insights verwenden, vor dem 30. Juni 2026 auf den erweiterten Modus von Database Insights aktualisieren. Informationen zum Upgrade auf den erweiterten Modus von Database Insights finden Sie unter Einschalten des erweiterten Modus von Database Insights für Amazon Aurora.
Wenn Sie nichts unternehmen, nutzen DB-Cluster, die Performance Insights verwenden, standardmäßig den Standardmodus von Database Insights. Im Standardmodus von

Database Insights verlieren Sie möglicherweise den Zugriff auf den Leistungsdatenverlauf, der älter als 7 Tage ist, und können Ausführungspläne und On-Demand-Analysefunktionen in der Amazon-RDS-Konsole möglicherweise nicht verwenden. Nach dem 30. Juni 2026 unterstützt nur der Advanced-Modus von Database Insights Ausführungspläne und On-Demand-Analysen.
Mit CloudWatch Database Insights können Sie die Datenbanklast für Ihre Datenbankflotte überwachen und die Leistung in großem Umfang analysieren und Fehler beheben. Weitere Informationen über Database Insights finden Sie unter Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights. Preisinformationen finden Sie unter CloudWatch Amazon-Preise.

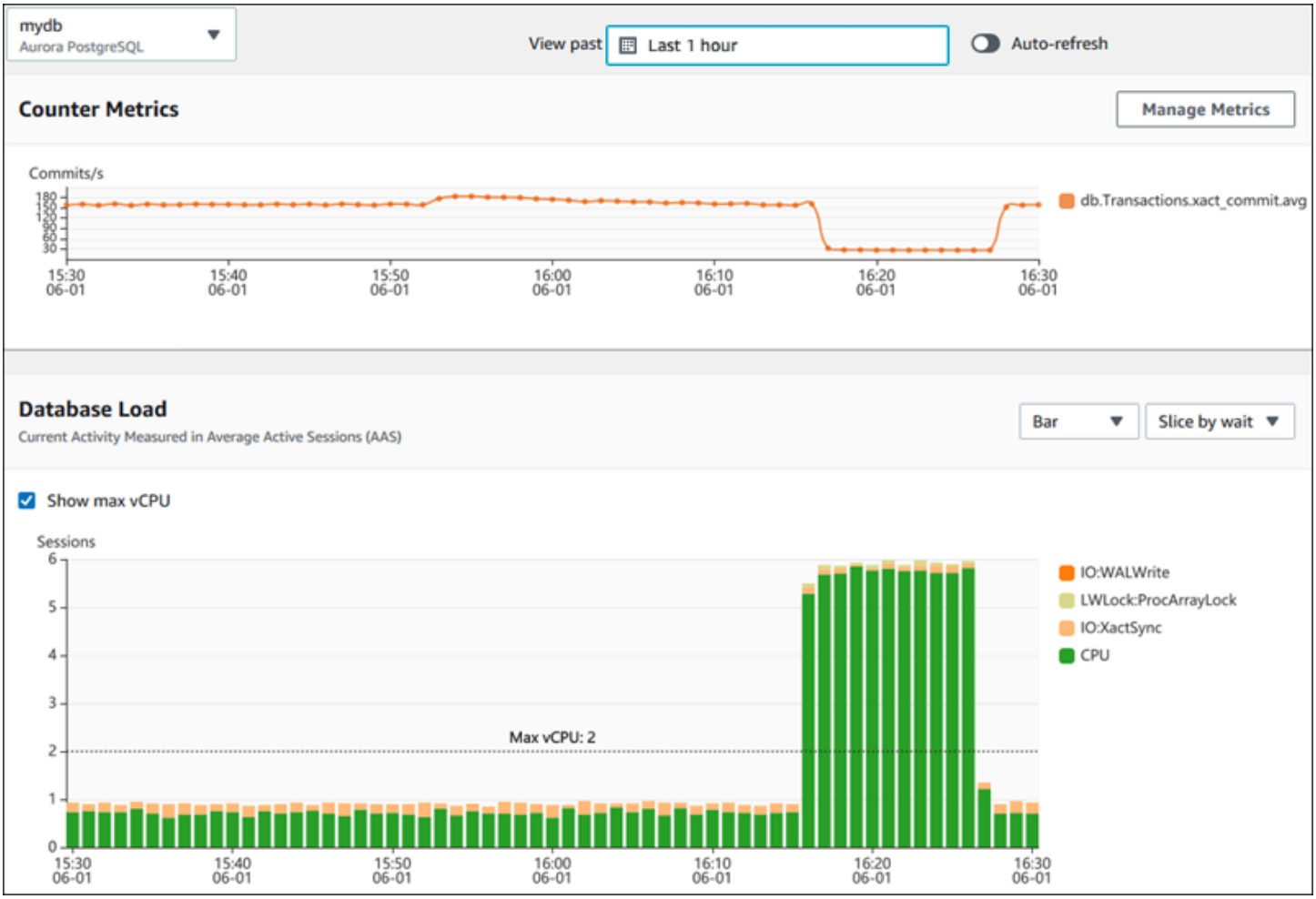
Das Dashboard von Performance Insights enthält Informationen zur Datenbank-Performance, die Sie bei der Analyse und Behebung von Performance-Problemen unterstützen. Auf der Dashboard-Hauptseite finden Sie Informationen zur Datenbanklast. Sie können DB-Lasten nach Dimensionen wie Warteereignissen oder SQL „aufteilen“.

Verwenden der Performance-Insights-Dashboard-Komponenten

- Überblick über Performance Insights
- Zugriff auf das Performance-Insights-Dashboard
- Analysieren der DB-Last nach Warteereignissen
- Analysieren der Datenbankleistung für einen bestimmten Zeitraum
- Analysieren von Abfragen mit der Registerkarte „Top SQL“ in Performance Insights

Überblick über Performance Insights

Das Dashboard ist die einfachste Möglichkeit, mit Performance Insights zu interagieren. Das folgende Beispiel zeigt das Dashboard für eine PostgreSQL-DB-Instance.



Themen

- Zeitraum-Filter
- Zählermetriken-Diagramm
- Datenbank-Ladediagramm
- Dimensionen pro Tabelle

Zeitraum-Filter

Standardmäßig zeigt das Dashboard von Performance Insights die DB-Last der letzten Stunde an. Sie können diesen Bereich so einstellen, dass er 5 Minuten oder bis zu 2 Jahre lang ist. Sie können auch einen benutzerdefinierten relativen Bereich auswählen.

Sie können einen absoluten Bereich mit einem Anfangs- und Enddatum und einer Uhrzeit auswählen. Das folgende Beispiel zeigt den Zeitraum, der am 25.09.2024 um Mitternacht beginnt und am 28.09.2024 um 23:59 Uhr Mitternacht endet.

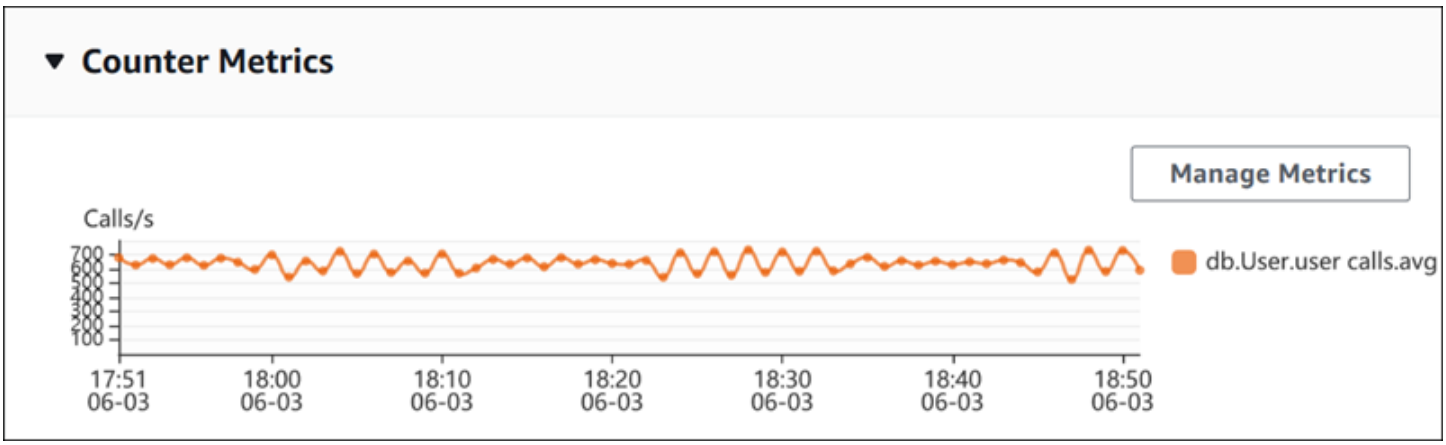
Standardmäßig ist die Zeitzone für das Performance-Insights-Dashboard auf die koordinierte Weltzeit (UTC) eingestellt. Sie können auch zur lokalen Zeitzone wechseln.

Zählermetriken-Diagramm

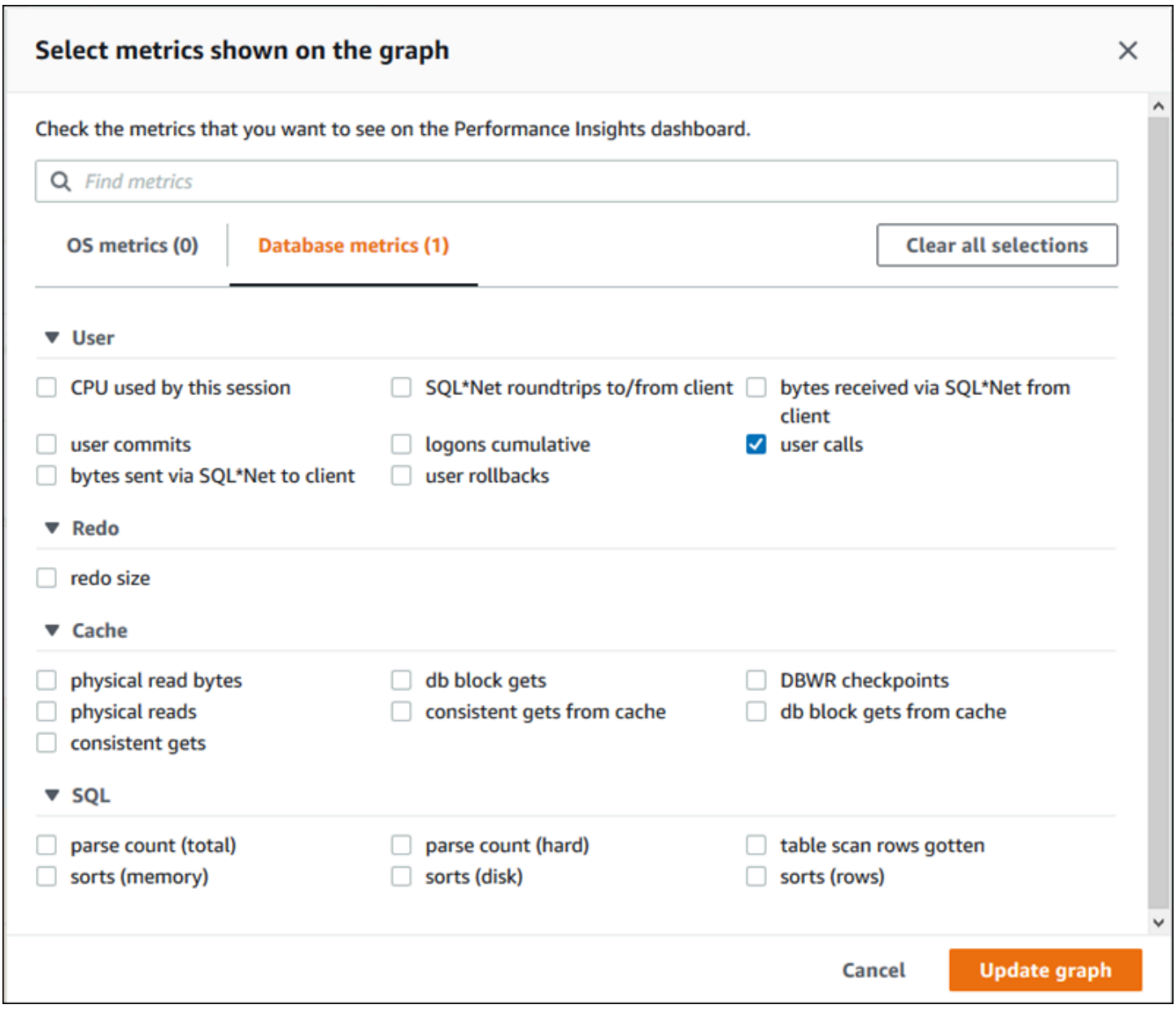
Mithilfe von Zählermetriken können Sie das Performance-Insights-Dashboard anpassen und bis zu 10 weitere Diagramme aufnehmen. Diese Diagramme enthalten eine Auswahl von Dutzenden von Betriebssystem- und Datenbank-Performance-Metriken. Diese Informationen können mit der Datenbanklast korreliert werden, um Performance-Probleme zu identifizieren und zu analysieren.

Das Counter Metrics (Zählermetriken)-Diagramm enthält Daten zu Leistungsindikatoren. Die Standardmetriken hängen von der DB-Engine ab.

- Aurora MySQL– `db.SQL.Innodb_rows_read.avg`
- Aurora PostgreSQL – `db.Transactions.xact_commit.avg`



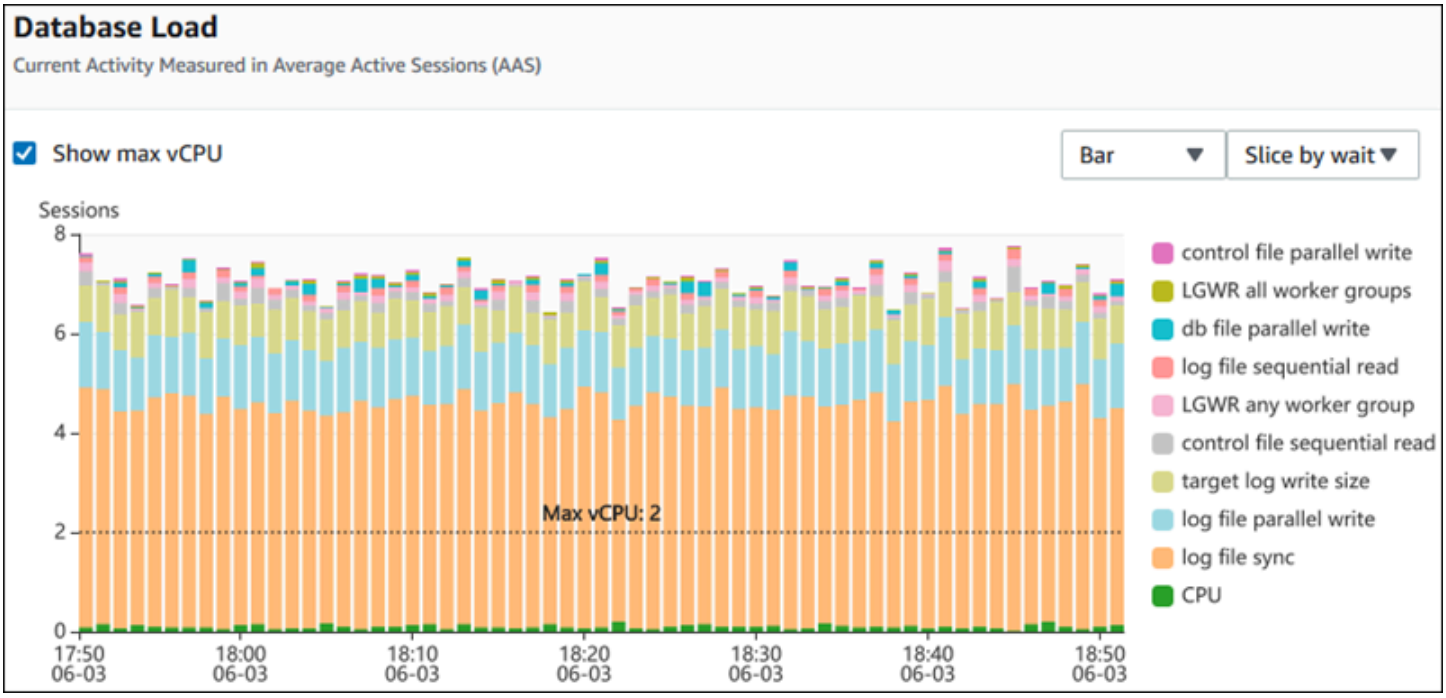
Ändern Sie die Leistungsindikatoren, indem Sie Metriken verwalten wählen. Sie können mehrere Betriebssystem-Metriken oder Datenbank-Metriken, auswählen, wie im folgenden Screenshot veranschaulicht. Um Details für jede Metrik anzuzeigen, bewegen Sie den Mauszeiger über den Metriknamen.



Beschreibungen der Zählermetriken, die Sie für jede DB-Engine hinzufügen können, finden Sie unter Performance-Insights-Zählermetriken.

Datenbank-Ladediagramm

Das Diagramm Database Load (Datenbank-Last) zeigt die Datenbanklast im Vergleich zur Kapazität der DB-Instance, die durch die Max vCPU-Linie dargestellt wird. Standardmäßig stellt das gestapelte Liniendiagramm die DB-Last als durchschnittliche aktive Sitzungen pro Zeiteinheit dar. Die DB-Last wird nach Wartestatus aufgeteilt (gruppiert).

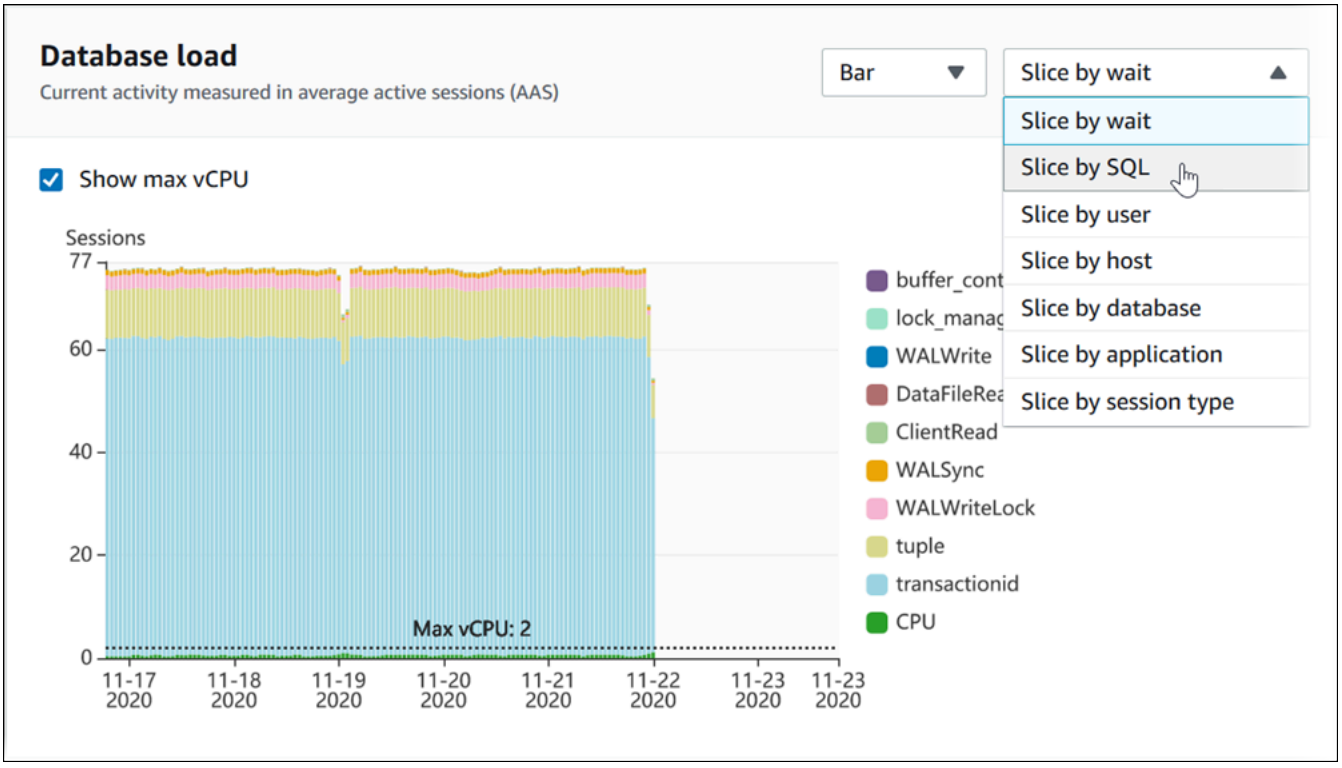


DB-Last aufgeteilt nach Dimensionen

Sie können die Last als aktive Sitzungen anzeigen, die nach unterstützten Dimensionen gruppiert sind. Die folgende Tabelle zeigt, welche Dimensionen für die verschiedenen Engines unterstützt werden.

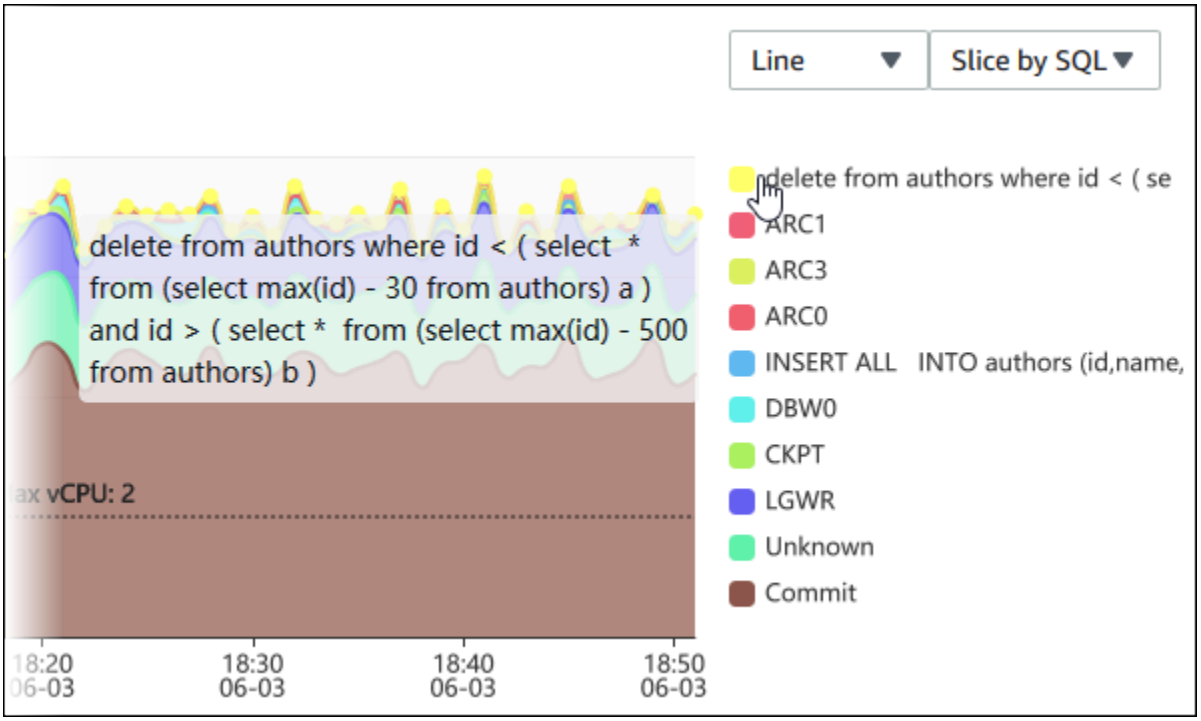
Dimension	Aurora PostgreSQL	Aurora MySQL
Host	Ja	Ja
SQL	Ja	Ja
Benutzer	Ja	Ja
Waits (Warteereignis)	Ja	Ja
Anwendung	Ja	Nein
Datenbank	Ja	Ja
Session type (Sitzungstyp)	Ja	Nein

Der folgende Screenshot zeigt die Dimensionen für eine PostgreSQL-DB-Instance.

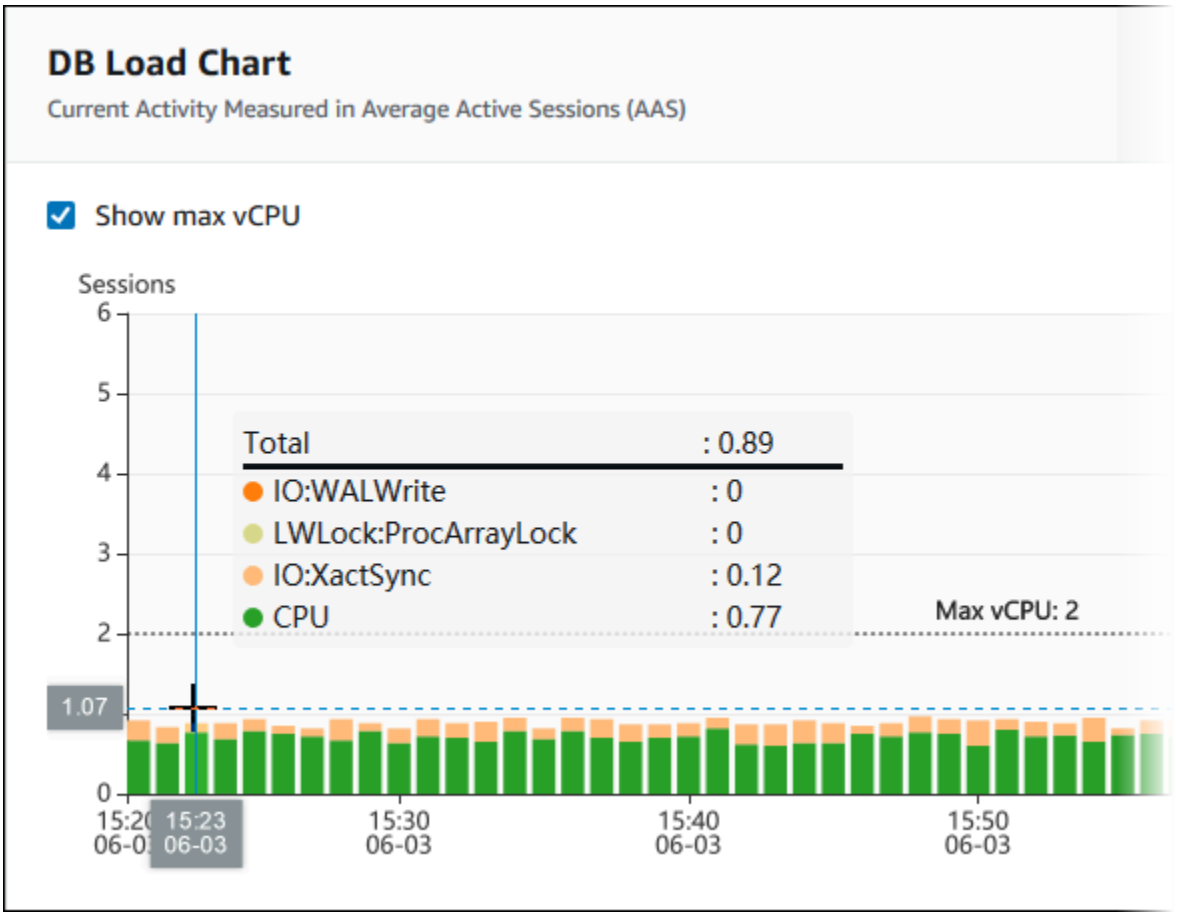


DB-Ladedetails für ein Dimensionselement

Um Details zu einem DB-Lastelement innerhalb einer Dimension anzuzeigen, bewegen Sie den Mauszeiger über den Elementnamen. Die folgende Abbildung zeigt Details zu einer SQL-Anweisung.

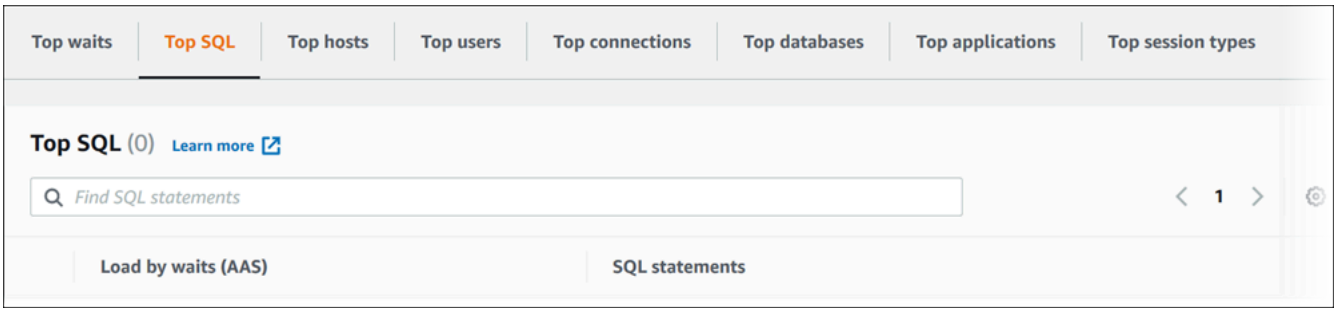


Um Details zu einem Element für den ausgewählten Zeitraum in der Legende anzuzeigen, bewegen Sie den Mauszeiger über dieses Element.



Dimensionen pro Tabelle

Die Tabelle mit den oberen Abmessungen schneidet die DB-Ladung um verschiedene Dimensionen auf. Eine Dimension ist eine Kategorie oder „Aufteilung“ für verschiedene Merkmale der DB-Last. Wenn die Dimension SQL ist, zeigt Haupt-SQL die SQL-Anweisungen an, die am meisten zur DB-Last beitragen.



Wählen Sie eine der folgenden Dimensionsregisterkarten.

Tab	Beschreibung	Unterstützte Engines
Haupt-SQL	Die SQL-Anweisungen, die derzeit ausgeführt werden	Alle
Top waits (Top-Warteereignis)	Das Ereignis, auf das das Datenbank-Backend wartet	Alle
Top hosts (Top-Hosts)	Der Hostname des verbunden en Clients	Alle
Top users (Top-Benutzer)	Der bei der Datenbank angemeldete Benutzer	Alle
Top applications (Top-Anwe ndungen)	Der Name der Anwendung , die mit der Datenbank verbunden ist	Nur Aurora PostgreSQL
Top session types (Top-Sitz ungstypen)	Der Typ der aktuellen Sitzung	Nur Aurora PostgreSQL

So lernen Sie, wie Sie Abfragen analysieren können, indem Sie die Registerkarte Haupt-SQL nutzen, siehe Überblick über die Registerkarte „Top SQL“.

Zugriff auf das Performance-Insights-Dashboard

Amazon RDS bietet im Performance-Insights-Dashboard eine konsolidierte Ansicht der Performance-Insights- und CloudWatch-Metriken.

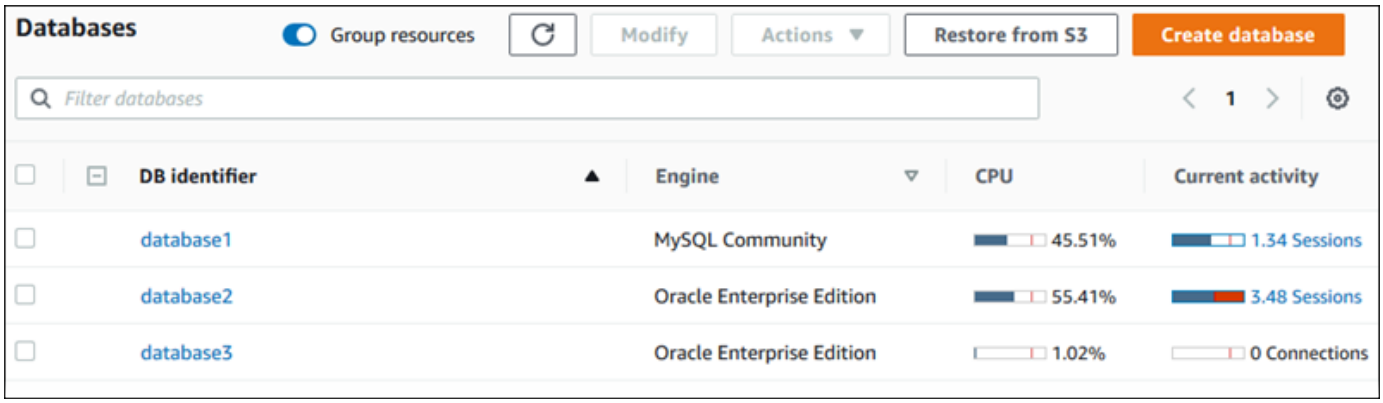
Gehen Sie wie folgt vor, um auf das Performance-Insights-Dashboard zuzugreifen.

So zeigen Sie das Dashboard von Performance Insights in der AWS-Managementkonsole an

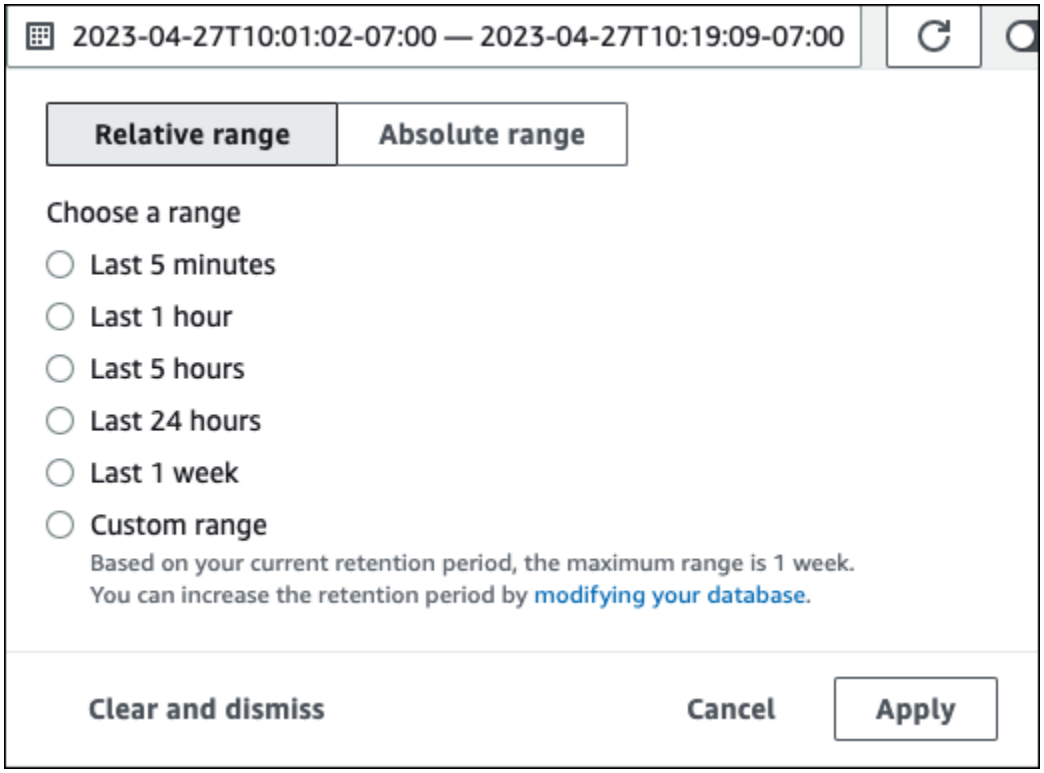
1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.

   Für DB-Instances, bei denen Performance Insights aktiviert sind, können Sie auf das Dashboard auch über das Element Sitzungen in der DB-Instance-Liste zugreifen. Unter Aktuelle Aktivität

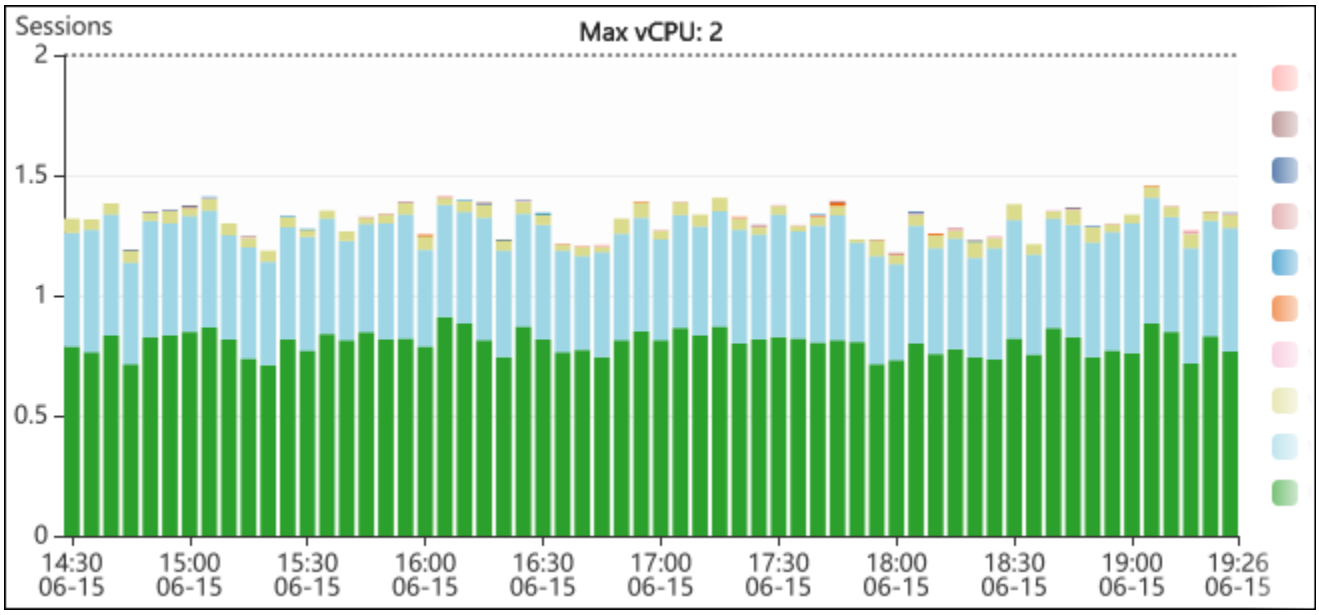
zeigt das Element Sitzungen die Datenbanklast von durchschnittlichen, aktiven Sitzungen der letzten fünf Minuten an. Der Balken zeigt die Last grafisch an. Ist der Balken leer, wird die DB-Instance nicht verwendet. Wenn die Last ansteigt, wird der Balken blau ausgefüllt. Wenn die Last die Anzahl von virtuellen CPUs (vCPUs) auf der DB-Instance-Klasse überschreitet, wird der Balken rot und zeigt so einen potenziellen Engpass an.



4. (Optional) Wählen Sie das Datum oder den Zeitraum oben rechts aus und geben Sie ein anderes relatives oder absolutes Zeitintervall an. Sie können jetzt einen Zeitraum angeben und einen Bericht zur Datenbankleistungsanalyse erstellen. Der Bericht enthält die identifizierten Einblicke und Empfehlungen. Weitere Informationen finden Sie unter Erstellen eines Leistungsanalyseberichts in Performance Insights.

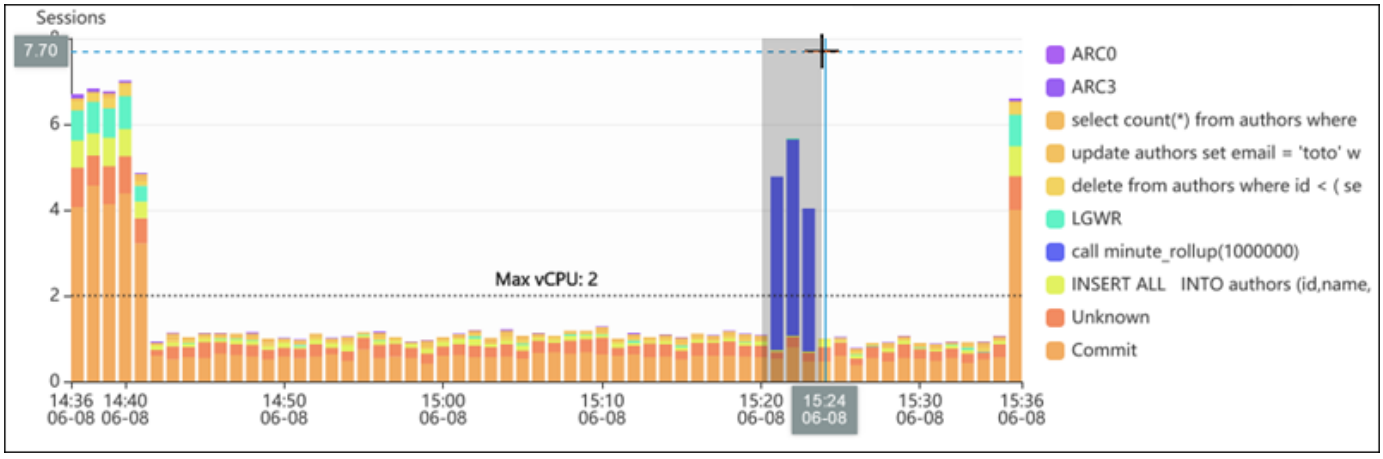


Im folgenden Screenshot beträgt das DB-Last-Intervall 5 Stunden.

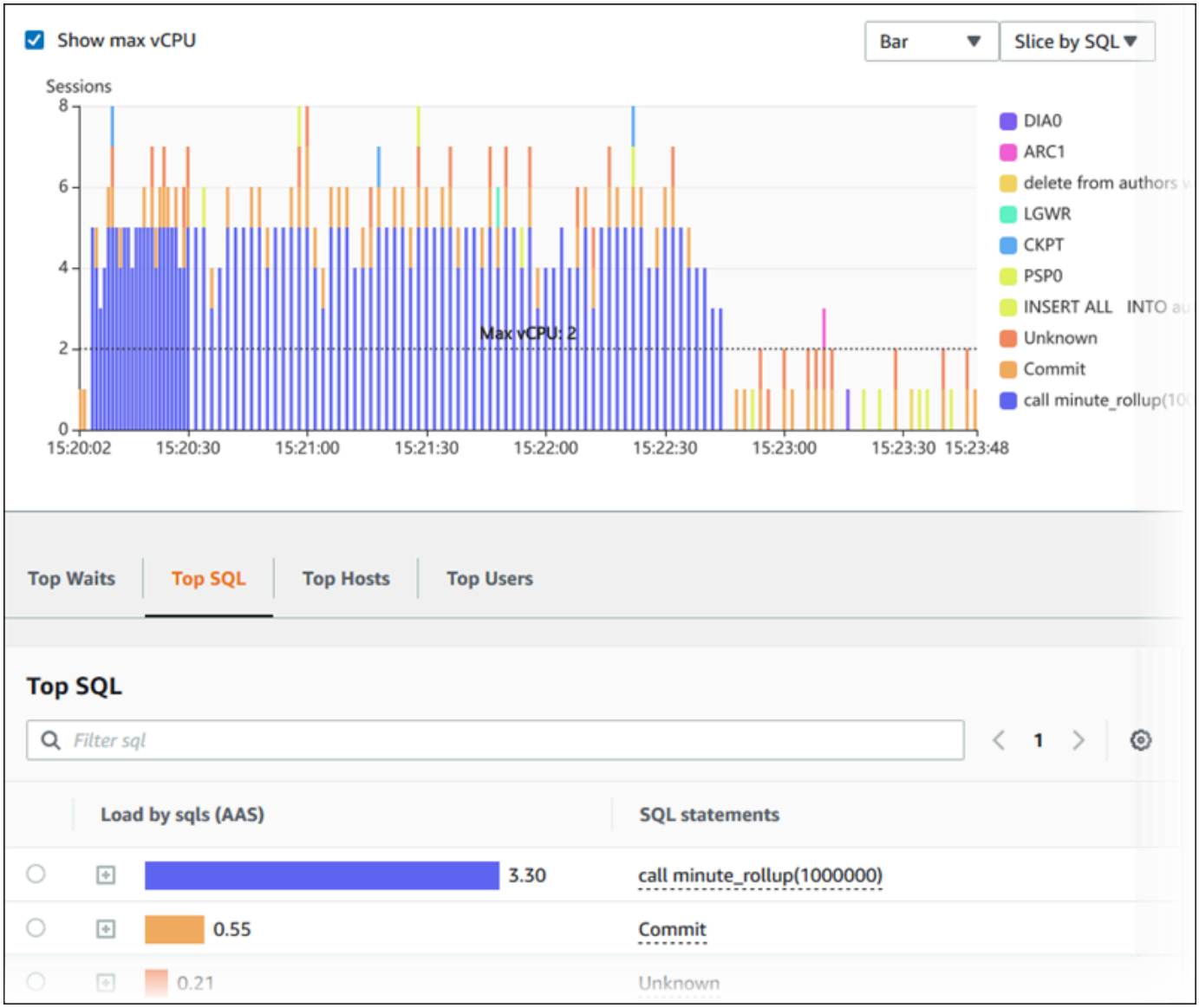


5. (Optional) Um einen Teil des DB-Lastdiagramms zu vergrößern, wählen Sie die Startzeit und ziehen sie mit der Maus an das Ende des gewünschten Zeitraums.

   Der ausgewählte Bereich wird im DB-Lastdiagramm hervorgehoben.



   Wenn Sie die Maustaste loslassen, vergrößert sich das DB-Lastdiagramm auf die ausgewählte AWS-Region und die Tabelle Top dimensions (Hauptdimensionen) wird neu berechnet.



6. (Optional) Um Ihre Daten automatisch zu aktualisieren, aktivieren Sie Automatische Aktualisierung.

Das Performance-Insights-Dashboard wird automatisch mit neuen Daten aktualisiert. Die Aktualisierungsrate hängt von der Menge der angezeigten Daten ab:

- 5 Minuten wird alle 10 Sekunden aktualisiert.
- 1 Stunde wird alle 5 Minuten aktualisiert.
- 5 Stunden wird alle 5 Minuten aktualisiert.
- 24 Stunden wird alle 30 Minuten aktualisiert.

- 1 Woche wird jeden Tag aktualisiert.
- 1 Monat wird jeden Tag aktualisiert.

Analysieren der DB-Last nach Warteereignissen

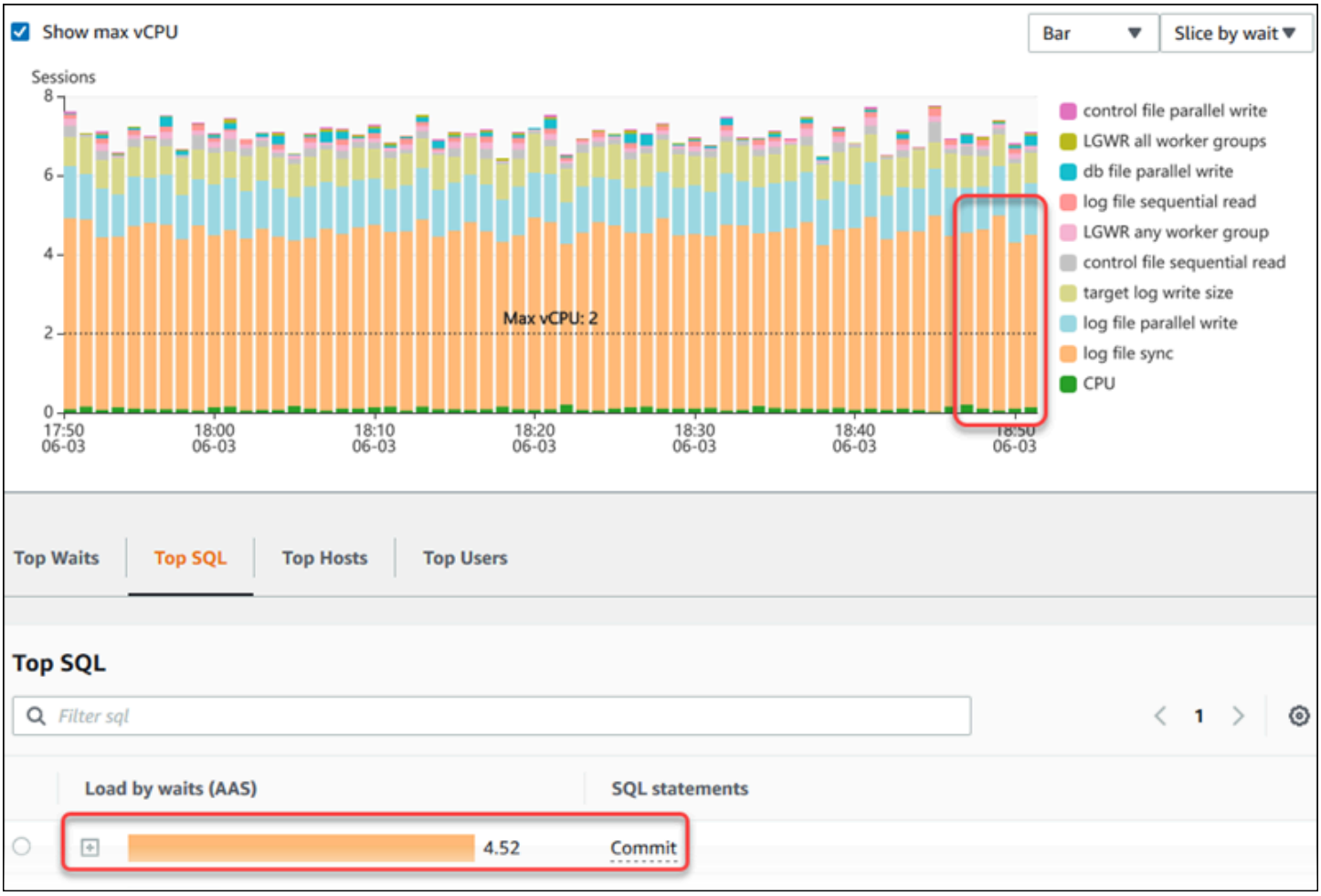
Wenn das Diagramm der durchschnittlich aktiven Sitzungen einen Engpass anzeigt, können Sie herausfinden, woher die Last kommt. Betrachten Sie dazu die Tabelle mit den Hauptlastelementen unterhalb des Datenbanklast-Diagramms Wählen Sie ein bestimmtes Element, wie z. B. eine SQL-Abfrage oder einen Benutzer, um es aufzuschlüsseln und Details zu diesem Element anzuzeigen.

Die DB-Last, gruppiert nach Wartezeiten und Top-SQL-Abfragen, ist die standardmäßige Ansicht im Performance-Insights-Dashboard. Diese Kombination bietet typischerweise den besten Einblick in Performance-Probleme. DB-Last gruppiert nach Wartezeiten zeigt an, ob Ressourcen- oder Parallelitätsengpässe in der Datenbank vorhanden sind. In diesem Fall zeigt die SQL-Registerkarte der Tabelle der Hauptlastelemente, welche Abfragen diese Last verursachen.

Ihr typischer Workflow für die Diagnose von Performance-Problemen ist folgendermaßen:

1. Überprüfen Sie das Diagramm der durchschnittlich aktiven Sitzungen auf irgendwelche Ereignisse, in denen die Datenbanklast die Max CPU-Linie übersteigt.
2. Wenn ja, schauen Sie sich das Diagramm der durchschnittlich aktiven Sitzungen an und identifizieren Sie, welcher Wartezustand oder welche Zustände primär dafür verantwortlich sind.
3. Identifizieren Sie die zusammengefassten Abfragen, welche die Last verursachen, indem Sie nachsehen, welche Abfragen in der SQL-Registerkarte der Tabelle der Hauptlastelemente hauptsächlich zu diesen Wartezuständen beitragen. Sie finden sie in der Spalte DB Load by Wait (DB-Last nach Wartezuständen).
4. Wählen Sie eine dieser zusammengefassten Abfragen in der Registerkarte SQL aus, um sie zu expandieren und untergeordnete Abfragen anzuzeigen, aus denen sie besteht.

Beispielsweise wird im folgenden Dashboard die Protokolldateisynchronisierung für den größten Teil der DB-Last berücksichtigt. Die Wartezeit für Alle Worker-Gruppen in LGWR ist ebenfalls hoch. Das Diagramm Haupt-SQL zeigt auf, wodurch die Wartezustände der Protokolldatei-Synchronisierung verursacht werden: häufige `COMMIT`-Anweisungen. In diesem Fall wird durch eine weniger häufige Übergabe mit Commit die DB-Last reduziert.



Analysieren der Datenbankleistung für einen bestimmten Zeitraum

Analysieren Sie die Datenbankleistung mit On-Demand-Analysen, indem Sie einen Leistungsanalysebericht für einen bestimmten Zeitraum erstellen. Zeigen Sie Leistungsanalyseberichte an, um Leistungsprobleme wie Ressourcenengpässe oder Änderungen an einer Abfrage in Ihrer DB-Instance zu ermitteln. Im Performance-Insights-Dashboard können Sie einen Zeitraum auswählen und einen Leistungsanalysebericht erstellen. Sie können dem Bericht auch ein oder mehrere Tags hinzufügen.

Um diese Funktion nutzen zu können, müssen Sie die Aufbewahrungsfrist für die kostenpflichtige Stufe verwenden. Weitere Informationen finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights

Der Bericht kann auf der Registerkarte Leistungsanalyseberichte – neu ausgewählt und angezeigt werden. Der Bericht enthält die Erkenntnisse, zugehörigen Metriken und Empfehlungen zur

Lösung des Leistungsproblems. Der Bericht kann für die Dauer des Aufbewahrungszeitraums von Performance Insights eingesehen werden.

Der Bericht wird gelöscht, wenn die Startzeit des Berichtsanalysezeitraums außerhalb des Aufbewahrungszeitraums liegt. Sie können den Bericht auch vor Ablauf der Aufbewahrungsfrist löschen.

Sie müssen Performance Insights aktivieren, um die Leistungsprobleme zu erkennen und den Analysebericht für Ihre DB-Instance zu erstellen. Weitere Informationen zum Aktivieren von Performance Insights finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora.

Informationen zur Unterstützung dieser Funktion nach Region, DB-Engine und Instance-Klasse finden Sie unter DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance-Insights-Funktionen.

In den folgenden Abschnitten können Sie einen Leistungsanalysebericht erstellen und anzeigen, ihm Tags hinzufügen und ihn löschen.

Themen

- Erstellen eines Leistungsanalyseberichts in Performance Insights
- Anzeigen eines Leistungsanalyseberichts in Performance Insights
- Hinzufügen von Tags zu einem Leistungsanalysebericht in Performance Insights
- Löschen eines Leistungsanalyseberichts in Performance Insights

Erstellen eines Leistungsanalyseberichts in Performance Insights

Im Performance-Insights-Dashboard können Sie einen Leistungsanalysebericht für einen bestimmten Zeitraum erstellen. Sie können einen Zeitraum auswählen und dem Analysebericht ein oder mehrere Tags hinzufügen.

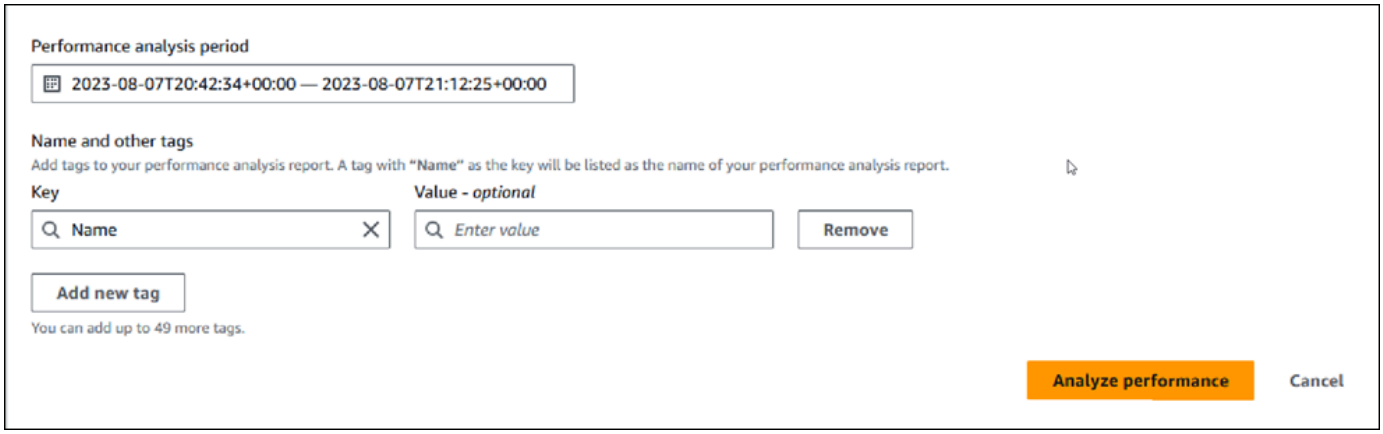
Der Analysezeitraum kann zwischen 5 Minuten und 6 Tagen liegen. Vor dem Start der Analyse müssen mindestens 24 Stunden an Leistungsdaten vorliegen.

Informationen zur Unterstützung dieser Funktion nach Region, DB-Engine und Instance-Klasse finden Sie unter DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance-Insights-Funktionen.

So erstellen Sie einen Leistungsanalysebericht für einen bestimmten Zeitraum

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.
4. Wählen Sie Leistung analysieren im Abschnitt Datenbanklast im Performance-Insights-Dashboard aus.

   Die Felder zum Festlegen des Zeitraums und zum Hinzufügen eines oder mehrerer Tags zum Leistungsanalysebericht werden angezeigt.

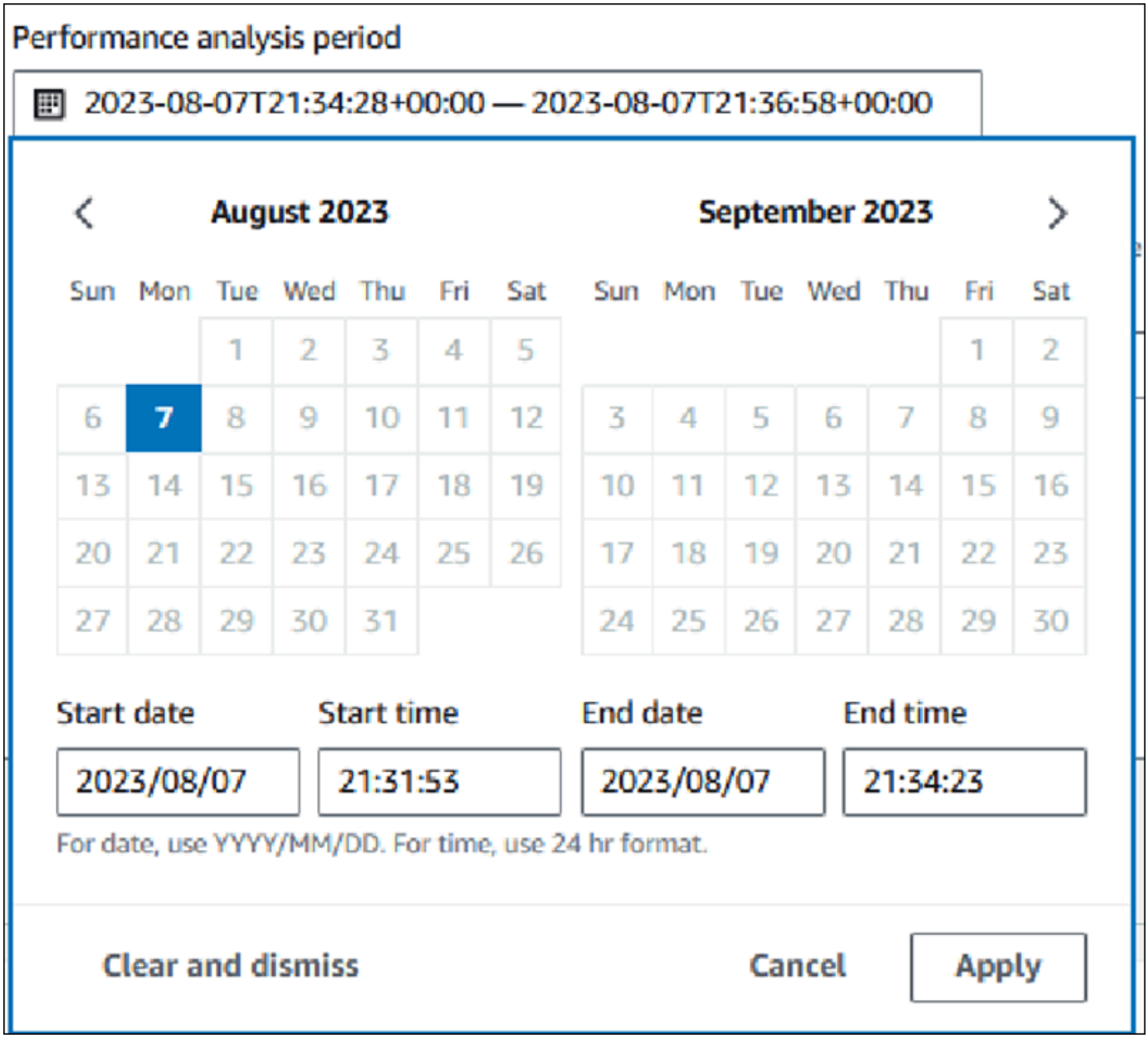


5. Wählen Sie den Zeitraum aus. Wenn Sie oben rechts einen Zeitraum im Abschnitt Relativer Bereich oder Absoluter Bereich festlegen, können Sie nur Datum und Uhrzeit des Analyseberichts innerhalb dieses Zeitraums eingeben oder auswählen. Wenn Sie den Analysezeitraum außerhalb dieses Zeitraums auswählen, wird eine Fehlermeldung angezeigt.

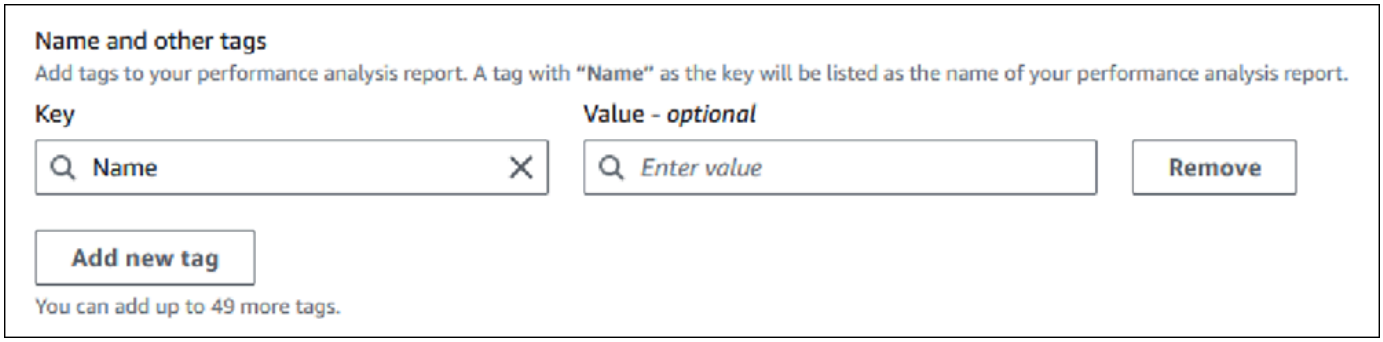
   Führen Sie einen der folgenden Schritte aus, um den Zeitraum festzulegen:

   - Drücken und ziehen Sie einen der Schieberegler im DB-Lastdiagramm.

     Das Feld Zeitraum der Leistungsanalyse zeigt den ausgewählten Zeitraum an und das DB-Lastdiagramm hebt den ausgewählten Zeitraum hervor.
   - Wählen Sie Startdatum, Startzeit, Enddatum und Endzeit im Feld Zeitraum der Leistungsanalyse aus.



6. (Optional) Geben Sie Schlüssel und Wertoptional ein, um ein Tag für den Bericht hinzuzufügen.



7. Wählen Sie Leistung analysieren aus.

   In einem Banner wird eine Meldung angezeigt, die angibt, ob die Berichtserstellung erfolgreich war oder fehlgeschlagen ist. Die Nachricht enthält auch den Link zum Anzeigen des Berichts.

   Das folgende Beispiel zeigt das Banner mit der Meldung, dass der Bericht erfolgreich erstellt wurde.

   Der Bericht kann auf der Registerkarte Leistungsanalyseberichte – neu angezeigt werden.

Mit der AWS CLI können Sie einen Leistungsanalysebericht erstellen. Ein Beispiel zur Erstellung eines Berichts mithilfe der AWS CLI finden Sie unterErstellen eines Leistungsanalyseberichts für einen bestimmten Zeitraum.

Anzeigen eines Leistungsanalyseberichts in Performance Insights

Die Registerkarte Leistungsanalysebericht – neu listet alle Berichte auf, die für die DB-Instance erstellt wurden. Für jeden Bericht wird Folgendes angezeigt:

- ID: eindeutige Kennung des Berichts.
- Name: der Tag-Schlüssel, der dem Bericht hinzugefügt wurde.
- Erstellungszeit des Berichts: Uhrzeit, zu der Sie den Bericht erstellt haben.
- Startzeit der Analyse: Startzeit der Analyse im Bericht.
- Endzeit der Analyse: Endzeit der Analyse im Bericht.

So zeigen Sie einen Leistungsanalysebericht an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus, für die Sie den Analysebericht anzeigen möchten.
4. Scrollen Sie nach unten und wählen Sie im Performance-Insights-Dashboard die Registerkarte Leistungsanalyseberichte – neu aus.

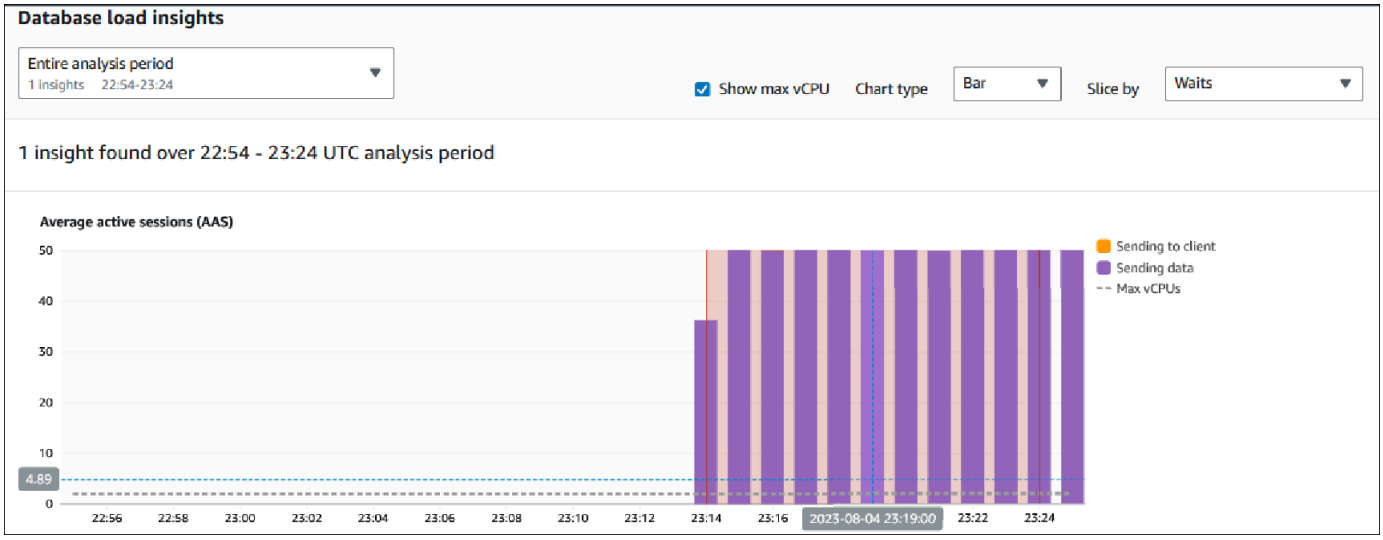
   Es werden alle Analyseberichte für die verschiedenen Zeiträume angezeigt.

5. Wählen Sie die ID des Berichts aus, den Sie ansehen möchten.

   Das DB-Lastdiagramm zeigt standardmäßig den gesamten Analysezeitraum an, wenn mehr als ein Einblick identifiziert wurde. Wenn der Bericht einen Einblick identifiziert hat, zeigt das DB-Lastdiagramm den Einblick standardmäßig an.

   Das Dashboard listet außerdem die Tags für den Bericht im Abschnitt Tags auf.

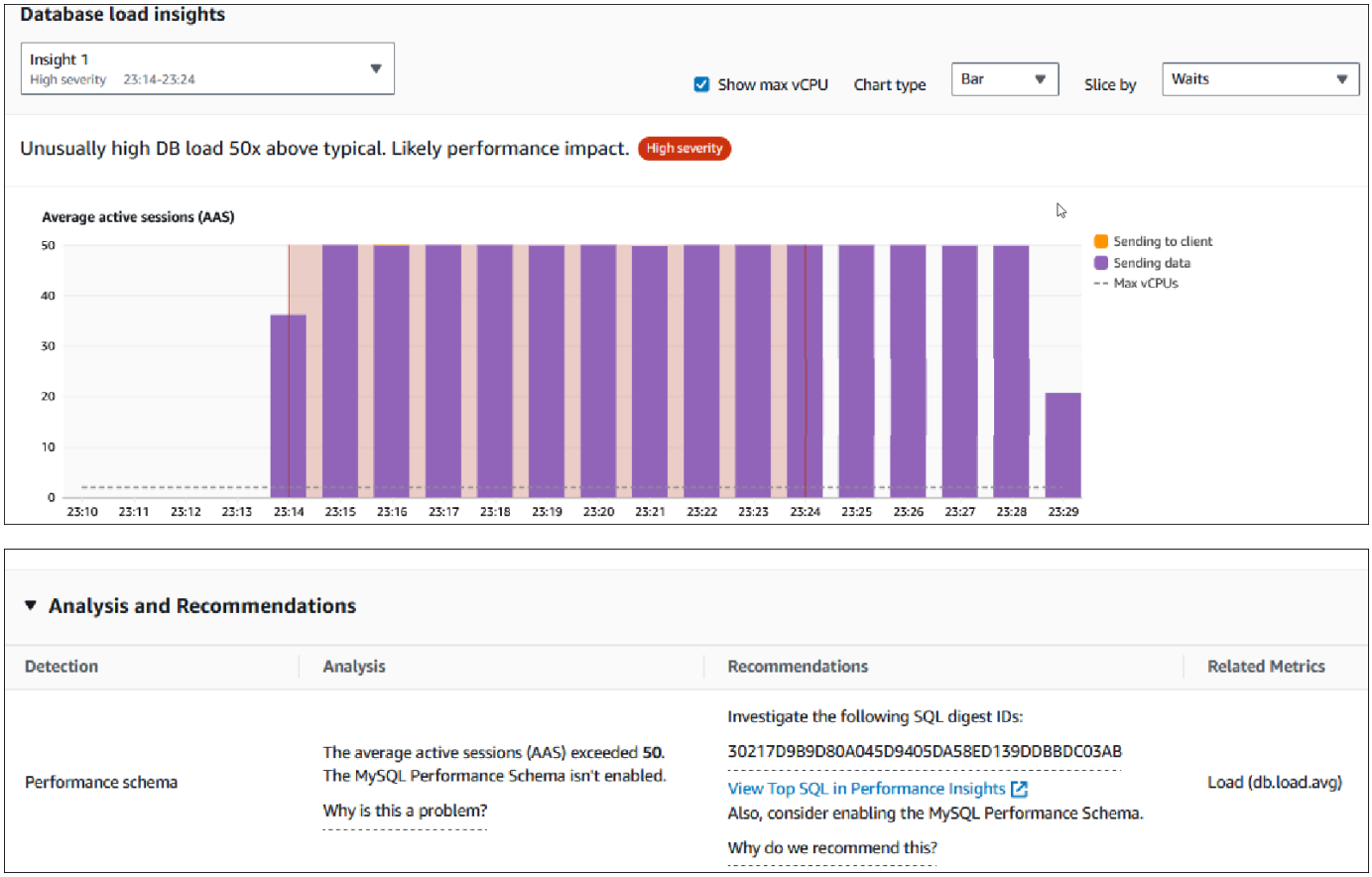
   Das folgende Beispiel zeigt den gesamten Analysezeitraum für den Bericht.



6. Wählen Sie die Einsicht in der Liste Einblicke in die Datenbanklast aus, die Sie anzeigen möchten, wenn im Bericht mehr als ein Einblick identifiziert wird.

   Das Dashboard zeigt die Einblickmeldung, wobei im DB-Lastdiagramm der Zeitraum des Einblicks, die Analyse und Empfehlungen hervorgehoben werden sowie die Liste der Berichts-Tags enthalten ist.

   Das folgende Beispiel zeigt den DB-Lasteinblick im Bericht.



Hinzufügen von Tags zu einem Leistungsanalysebericht in Performance Insights

Sie können ein Tag hinzufügen, wenn Sie einen Bericht erstellen oder ansehen. Sie können bis zu 50 Tags für einen Bericht hinzufügen.

Sie benötigen Berechtigungen, um Tags hinzuzufügen. Weitere Informationen zu Zugriffsrichtlinien für Performance Insights finden Sie unter Konfigurieren von Zugriffsrichtlinien für Performance Insights.

Informationen zum Hinzufügen eines oder mehrerer Tags bei der Erstellung eines Berichts finden Sie in Schritt 6 des Verfahrens Erstellen eines Leistungsanalyseberichts in Performance Insights.

So fügen Sie beim Anzeigen eines Berichts ein oder mehrere Tags hinzu

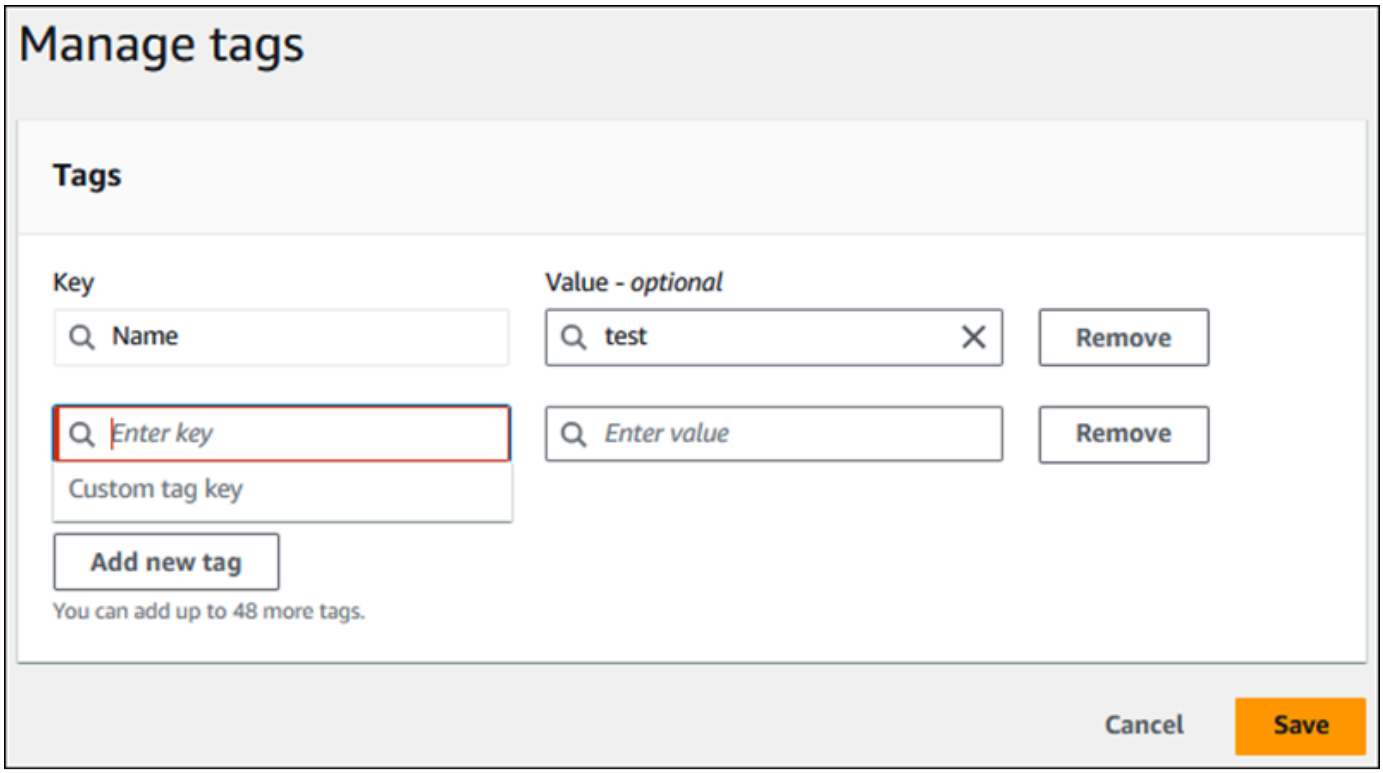
1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.

Das Performance-Insights-Dashboard wird für die DB-Instance angezeigt.

4. Scrollen Sie nach unten und wählen Sie die Registerkarte Leistungsanalyseberichte – neu aus.
5. Wählen Sie den Bericht aus, für den Sie die Tags hinzufügen möchten.

   Das Dashboard zeigt den Bericht an.
6. Scrollen Sie nach unten zu Tags und wählen Sie Tags verwalten aus.
7. Wählen Sie Neues Tag hinzufügen aus.
8. Geben Sie Schlüssel und Wert – optional ein und wählen Sie Neues Tag hinzufügen aus.

   Das folgende Beispiel bietet die Option, ein neues Tag für den ausgewählten Bericht hinzuzufügen.



   Ein neues Tag wird für den Bericht erstellt.

   Im Dashboard wird die Liste der Tags für den Bericht im Abschnitt Tags aufgelistet. Wenn Sie ein Tag aus dem Bericht entfernen möchten, wählen Sie Entfernen neben dem Tag aus.

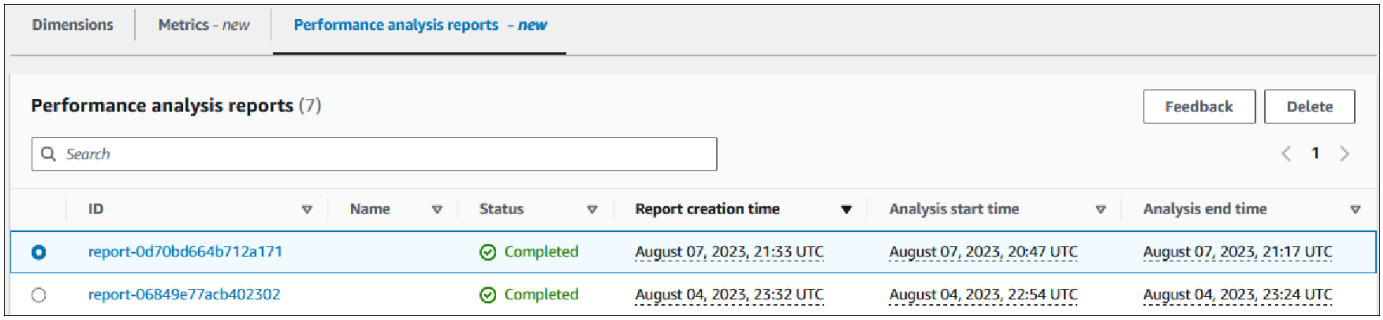
Löschen eines Leistungsanalyseberichts in Performance Insights

Sie können einen Bericht aus der Liste der Berichte, die auf der Registerkarte Leistungsanalyseberichte angezeigt werden, oder beim Anzeigen eines Berichts löschen.

So löschen Sie einen Bericht

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie eine DB-Instance aus.

   Das Performance-Insights-Dashboard wird für die DB-Instance angezeigt.
4. Scrollen Sie nach unten und wählen Sie die Registerkarte Leistungsanalyseberichte – neu aus.
5. Wählen Sie den Bericht aus, den Sie löschen möchten, und klicken Sie oben rechts auf Löschen.



   Ein Bestätigungsfenster wird angezeigt. Der Bericht wird gelöscht, nachdem Sie „Bestätigen“ ausgewählt haben.
6. (Optional) Wählen Sie die ID des Berichts, den Sie löschen möchten.

   Wählen Sie oben rechts auf der Seite Löschen aus.

   Ein Bestätigungsfenster wird angezeigt. Der Bericht wird gelöscht, nachdem Sie „Bestätigen“ ausgewählt haben.

Analysieren von Abfragen mit der Registerkarte „Top SQL“ in Performance Insights

Im Amazon-RDS-Performance-Insights-Dashboard finden Sie Informationen zu laufenden Abfragen auf dem Tab Top SQL (Haupt-SQL) der Tabelle Top dimensions (Hauptdimensionen). Sie können diese Informationen verwenden, um Ihre Abfragen zu optimieren.

Themen

- Überblick über die Registerkarte „Top SQL“
- Zugriff auf mehr SQL-Text im Performance-Insights-Dashboard
- Anzeigen von SQL-Statistiken im Performance-Insights-Dashboard

Überblick über die Registerkarte „Top SQL“

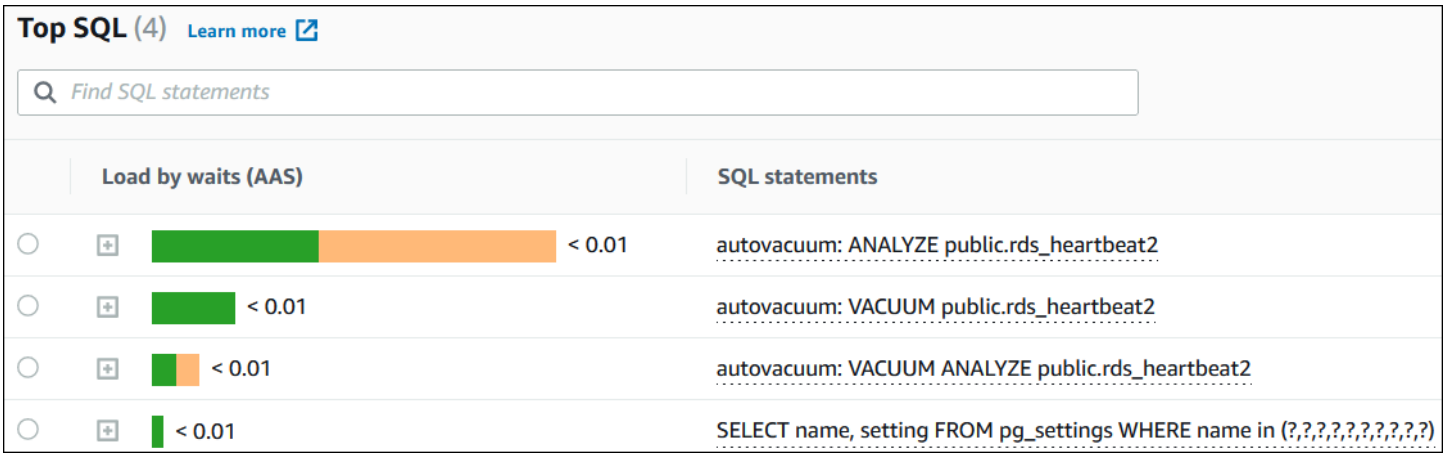
Standardmäßig werden auf der Registerkarte Top SQL (Top-SQL) die 25 Abfragen angezeigt, die hauptsächlich zur Datenbanklast beitragen. Wenn Sie Ihre Abfragen optimieren möchten, können Sie Informationen wie den Abfragetext und SQL-Statistiken analysieren. Sie können auch die Statistiken auswählen, die in der Haupt-SQL Tabulatortaste angezeigt werden.

Themen

- SQL-Text
- SQL-Statistiken
- Nach Waits laden (AAS)
- Anzeigen der SQL-Informationen
- Einstellungen für Statistiken

SQL-Text

Standardmäßig zeigt jede Zeile in der Tabelle Top SQL (Top-SQL) für jede Anweisung 500 Byte Text an.

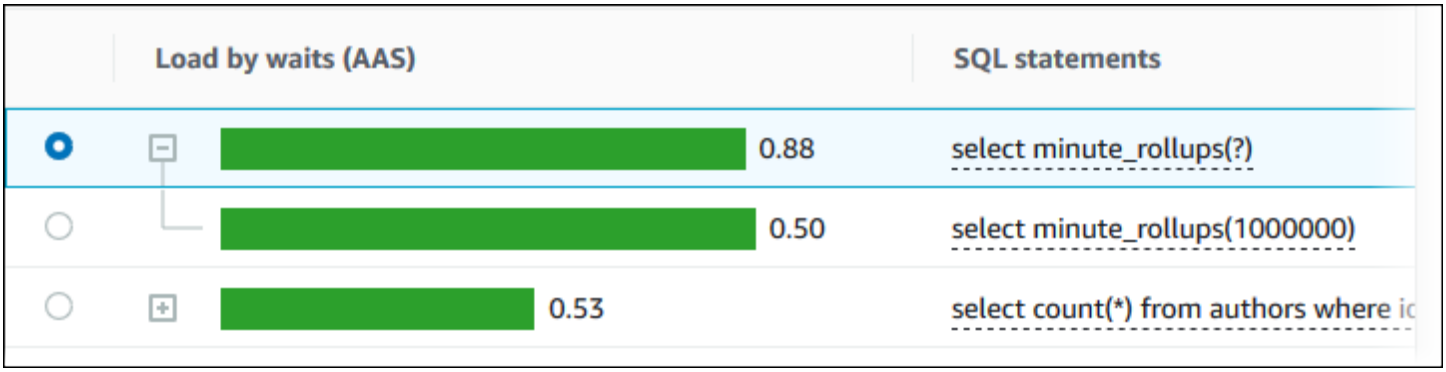


Wie Sie mehr als die standardmäßigen 500 Byte SQL-Text sehen können, erfahren Sie unter Zugriff auf mehr SQL-Text im Performance-Insights-Dashboard.

Ein SQL-Digest ist eine Zusammenstellung mehrerer tatsächlicher Abfragen, die strukturell ähnlich sind, aber möglicherweise unterschiedliche Literalwerte aufweisen. Der Digest ersetzt fest codierte Werte durch ein Fragezeichen. Zum Beispiel könnte `SELECT * FROM emp WHERE lname= ?` ein Digest sein. Dieser Digest kann die folgenden untergeordneten Abfragen enthalten:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Um die literalen SQL-Anweisungen in einem Digest anzuzeigen, wählen Sie die Abfrage aus und dann das Plussymbol (+) aus und dann das Plussymbol (+). Im folgenden Beispiel ist die ausgewählte Abfrage ein Digest.

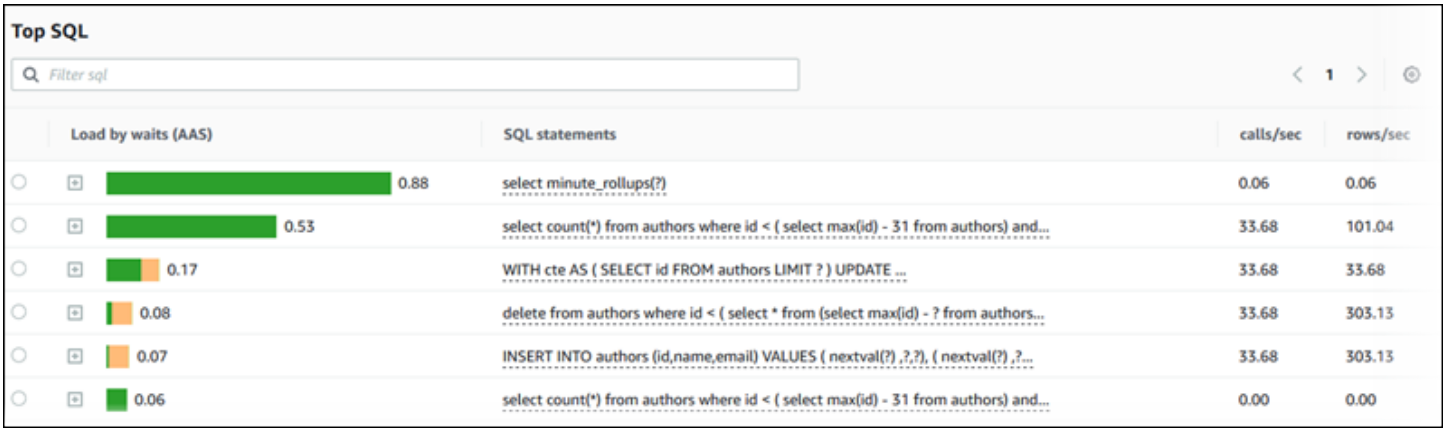


> **Note**
>
> Ein SQL-Digest gruppiert ähnliche SQL-Anweisungen, redigiert jedoch keine sensiblen Daten.

SQL-Statistiken

SQL-Statistiken sind leistungsbezogene Metriken zu SQL-Abfragen. Performance Insights könnte beispielsweise Ausführungen pro Sekunde oder pro Sekunde verarbeitete Zeilen anzeigen. Performance Insights erfasst Statistiken nur für die häufigsten Abfragen. In der Regel entsprechen diese den Top-Abfragen nach Last, die im Performance-Insights-Dashboard angezeigt werden.

Jede Zeile in der Haupt-SQL-Tabelle zeigt relevante Statistiken für die SQL-Anweisung oder -Digest, wie im folgenden Beispiel beschrieben.



Performance Insights kann `0.00` und `-` (unbekannt) für SQL-Statistiken melden. Diese Situation tritt unter den folgenden Bedingungen auf:

- Es ist nur eine Stichprobe vorhanden. Performance Insights berechnet beispielsweise Veränderungsraten für Aurora-PostgreSQL-Abfragen basierend auf mehreren Stichproben der Ansicht `pg_stat_statements`. Wenn eine Workload für kurze Zeit ausgeführt wird, erfasst Performance Insights möglicherweise nur eine Stichprobe, was bedeutet, dass keine Änderungsrate berechnet werden kann. Der unbekannte Wert wird durch einen Bindestrich (`-`) dargestellt.
- Zwei Stichproben haben die gleichen Werte. Performance Insights kann keine Änderungsrate berechnen, da keine Änderung stattgefunden hat, weshalb die Rate als `0.00` gemeldet wird.
- Einer Aurora-PostgreSQL-Anweisung fehlt ein gültiger Bezeichner. PostgreSQL erstellt erst nach dem Parsen und Analysieren einen Bezeichner für eine Anweisung. Somit kann eine Anweisung in den internen In-Memory-Strukturen von PostgreSQL ohne Bezeichner vorhanden sein. Da Performance Insights einmal pro Sekunde interne In-Memory-Strukturen erfasst, werden Abfragen mit niedriger Latenz möglicherweise nur für eine einzige Stichprobe angezeigt. Wenn die Abfrage-ID für dieses Beispiel nicht verfügbar ist, kann Performance Insights diese Anweisung nicht mit den entsprechenden Statistiken verknüpfen. Der unbekannte Wert wird durch einen Bindestrich (`-`) dargestellt.

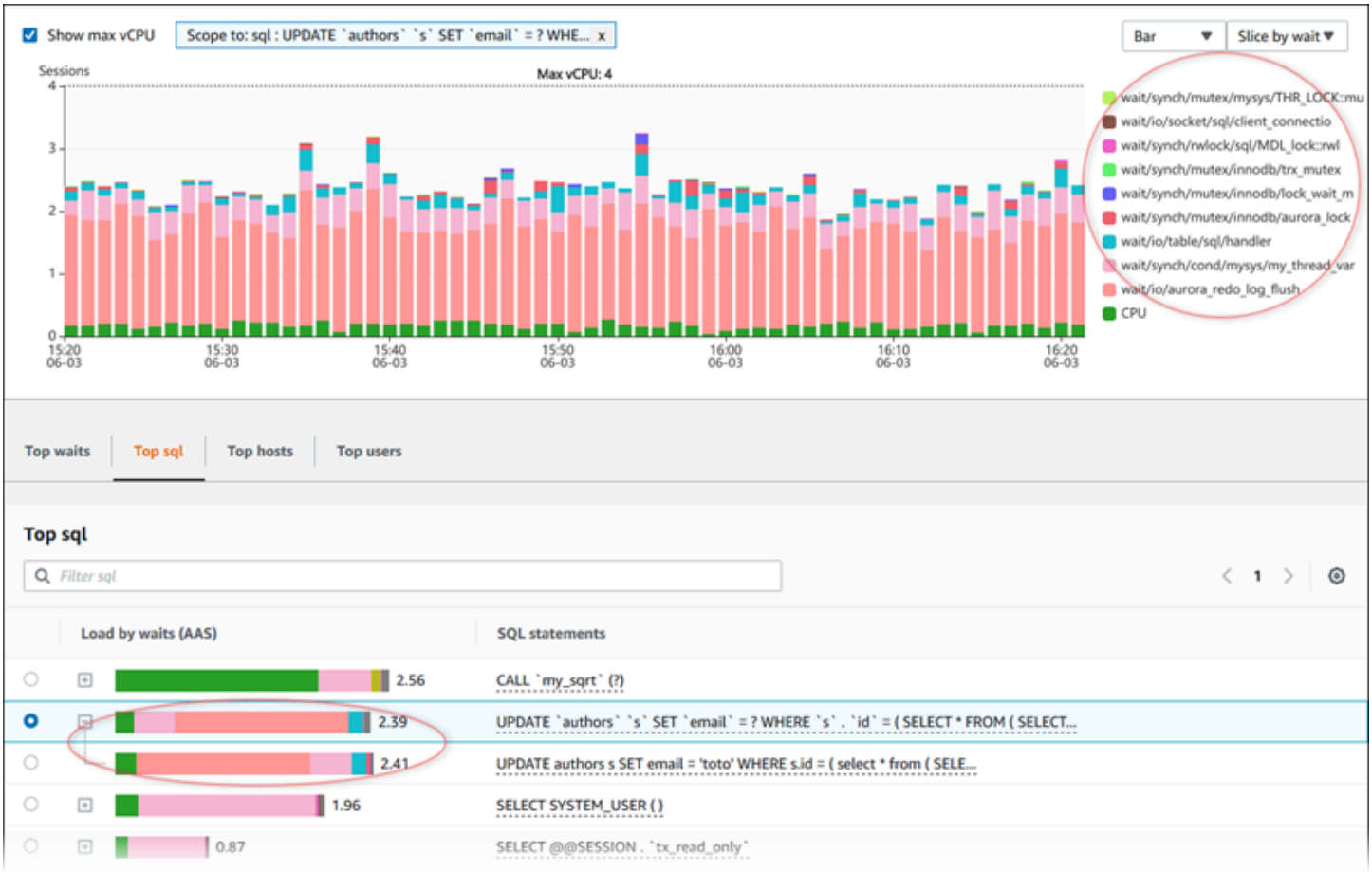
Eine Beschreibung der SQL-Statistiken für die Aurora-Engines finden Sie unter SQL-Statistiken für Performance Insights.

Nach Waits laden (AAS)

In Haupt-SQL veranschaulicht die Spalte Last nachWartezuständen (AAS) den Prozentsatz der Datenbanklast, die jedem Hauptlastelement zugeordnet ist. In dieser Spalte wird die Last für dieses

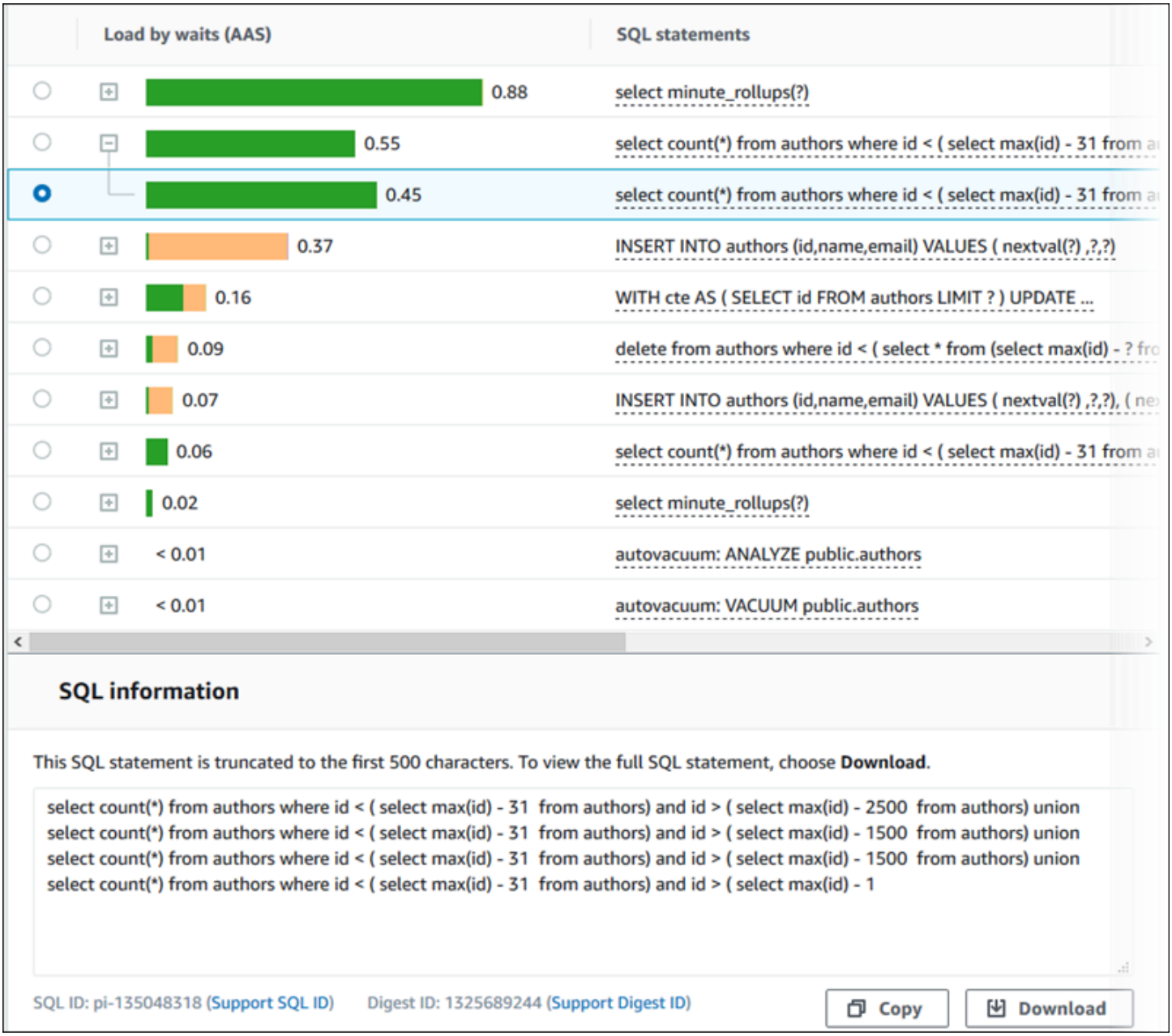
Element nach der aktuell im DB-Last-Diagramm ausgewählten Gruppierung wiedergegeben. Weitere Informationen zu durchschnittlichen aktiven Sitzungen (AAS) finden Sie unter Durchschnittliche aktive Sitzungen.

Beispielsweise können Sie das DB-Last-Diagramm nach Wartezuständen gruppieren. Sie untersuchen SQL-Abfragen in der Tabelle der Hauptlastelemente. In diesem Fall ist der Balken DB Load by Waits (DB-Last nach Wartezuständen) so groß, segmentiert und farbcodiert, dass angezeigt wird, zu wieviel Prozent diese Abfrage zum betreffenden Wartezustand beiträgt. Es zeigt zudem auf, welche Wartezustände sich auf die ausgewählte Abfrage auswirken.



Anzeigen der SQL-Informationen

In der Tabelle Haupt-SQL können Sie eine Anweisung öffnen, um ihre Informationen anzuzeigen. Die Informationen werden im unteren Bereich angezeigt.

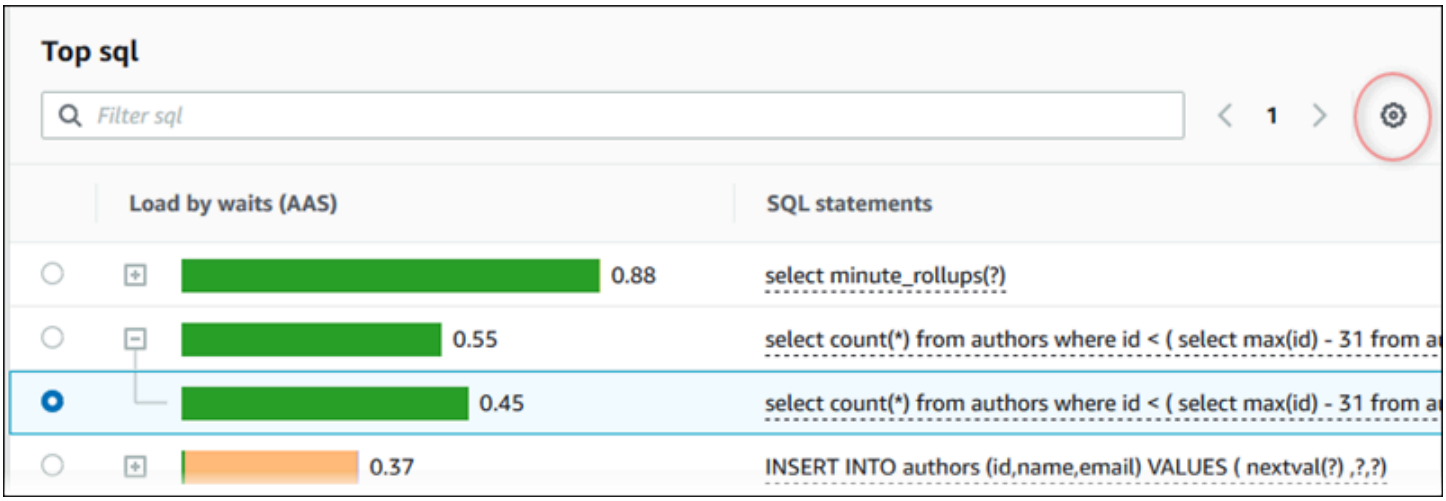


Auf der Registerkarte Haupt-SQL sehen Sie die folgenden ID-Typen, die SQL-Anweisungen zugeordnet sind:

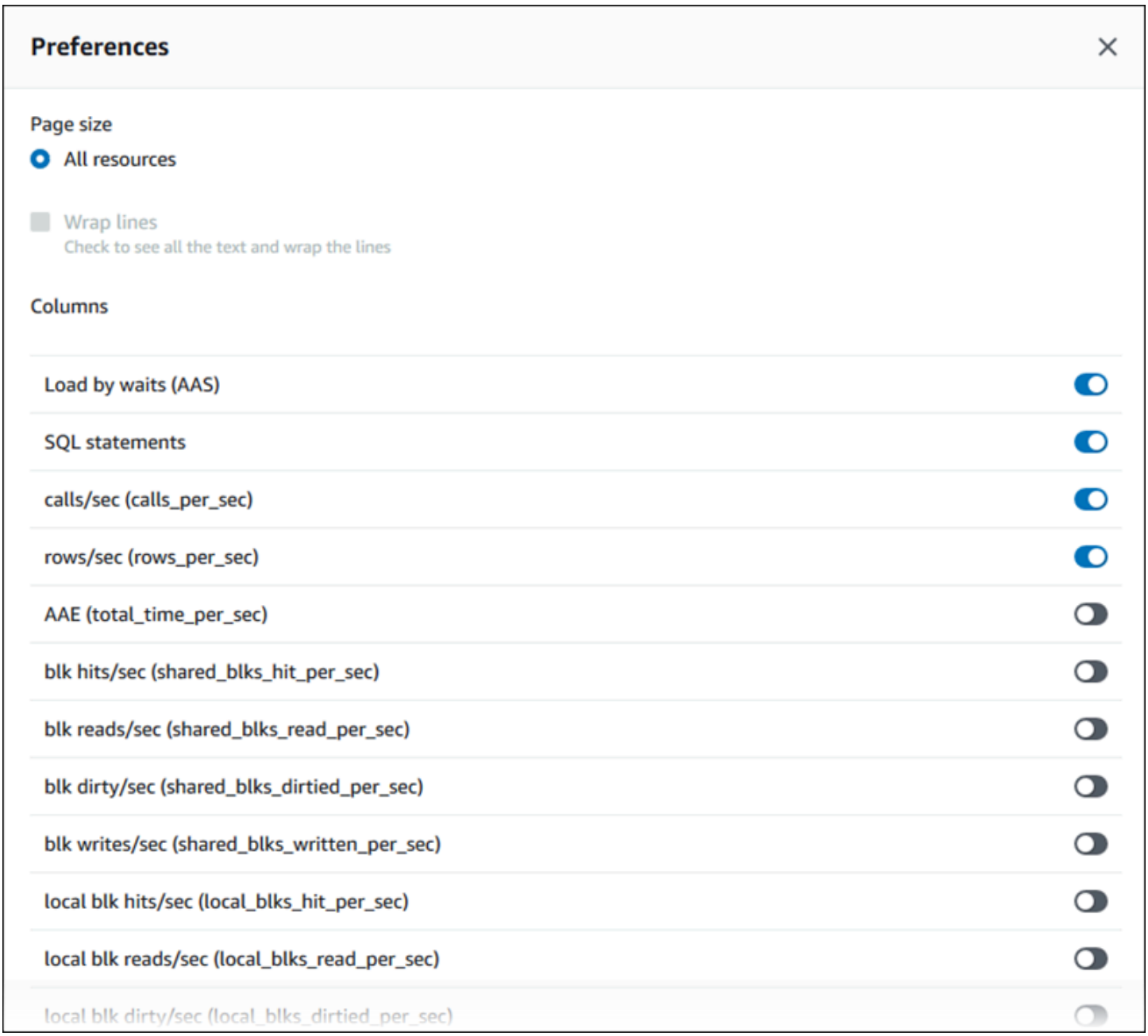
- Support-SQL-ID: ein Hash-Wert der SQL-ID Sie können eine SQL-ID über diesen Wert nur referenzieren, wenn Sie mit AWS Support arbeiten. AWS Support kann nicht auf Ihre SQL-IDs und Ihren SQL-Text zugreifen.
- Support-Digest-ID: ein Hash-Wert der Digest-ID Sie können eine Digest-ID über diesen Wert nur referenzieren, wenn Sie mit AWS Support arbeiten. AWS Support kann nicht auf Ihre Digest-IDs und Ihren SQL-Text zugreifen.

Einstellungen für Statistiken

Sie können die Statistiken steuern, die auf der Registerkarte Haupt-SQL angezeigt werden, indem Sie das Symbol Voreinstellungen auswählen.



Durch das Auswählen des Symbols Präferenzen wird das Fenster Präferenzen geöffnet. Der folgende Screenshot ist ein Beispiel für das Fenster Preferences (Präferenzen).



Aktivieren Sie die Statistiken, die auf der Registerkarte Haupt-SQL angezeigt werden sollen, führen Sie einen Bildlauf mit der Maus zum unteren Rand des Fensters durch und wählen Sie dann Weiter.

Weitere Informationen zu Statistiken pro Sekunde oder pro Aufruf für die Aurora-Engines finden Sie im Abschnitt der Engine-spezifischen SQL-Statistiken unter SQL-Statistiken für Performance Insights.

Zugriff auf mehr SQL-Text im Performance-Insights-Dashboard

Standardmäßig zeigt jede Zeile in der Tabelle Haupt-SQL für jede SQL-Anwendung 500 Byte SQL-Text an.

Wenn eine SQL-Anweisung 500 Byte überschreitet, können Sie mehr Text im SQL-Text-Abschnitt unterhalb der Haupt-SQL-Tabelle sehen. In diesem Fall beträgt die maximale Länge für den in SQL-Text angezeigten Text 4 KB. Dieses Limit wird von der Konsole eingeführt und unterliegt den von der Datenbank-Engine festgelegten Grenzwerten. Zum Speichern des in SQL-Text gezeigten Texts wählen Sie Herunterladen.

Themen

- Beschränkungen der Textgröße für Aurora MySQL
- Festlegen des SQL-Textlimits für Aurora PostgreSQL-DB-Instances
- Anzeigen und Herunterladen von SQL-Text im Performance-Insights-Dashboard

Beschränkungen der Textgröße für Aurora MySQL

Beim Herunterladen von SQL-Text bestimmt die Datenbank-Engine dessen maximale Länge. Sie können SQL-Text bis zu den folgenden Grenzwerten pro Engine herunterladen.

DB-Engine	Maximale Länge des heruntergeladenen Textes
Aurora MySQL	Die Länge ist auf 4.096 Bytes festgelegt.

Der SQL-Text-Abschnitt der Performance-Insights-Konsole zeigt den maximalen Wert an, den die Engine zurückgibt. Wenn Aurora MySQL beispielsweise höchstens 1 KB an Performance Insights zurückgibt, kann es nur 1 KB sammeln und anzeigen, auch wenn die ursprüngliche Abfrage größer ist. Wenn Sie also die Abfrage in SQL-Text anzeigen oder herunterladen, gibt Performance Insights die gleiche Anzahl von Bytes zurück.

Wenn Sie die AWS CLI oder API nutzen, hat Performance Insights nicht das von der Konsole erzwungene Limit von 4 KB. `DescribeDimensionKeys` und `GetResourceMetrics` geben höchstens 500 Bytes zurück.

> **Note**
>
> `GetDimensionKeyDetails` gibt die vollständige Abfrage zurück, aber die Größe unterliegt dem Engine-Limit.

Festlegen des SQL-Textlimits für Aurora PostgreSQL-DB-Instances

Aurora PostgreSQL behandelt Text anders. Sie können die Textgrößenbeschränkung mit dem DB-Instance-Parameter `track_activity_query_size` festlegen. Dieser Parameter hat folgende Merkmale:

Standardtextgröße

In Aurora PostgreSQL Version 9.6 ist die Standardeinstellung für den `track_activity_query_size`-Parameter 1.024 Byte. In Aurora PostgreSQL Version 10 oder höher ist die Standardeinstellung 4.096 Byte.

Maximale Textgröße

Das Limit für `track_activity_query_size` ist 102.400 Bytes für Aurora PostgreSQL Version 12 und niedriger. Das Maximum beträgt 1 MB für Version 13 und höher.

Wenn die Engine 1 MB an Performance Insights zurückgibt, zeigt die Konsole nur die ersten 4 KB an. Wenn Sie die Abfrage herunterladen, erhalten Sie die gesamten 1 MB. In diesem Fall geben das Anzeigen und Herunterladen eine unterschiedliche Anzahl von Bytes zurück. Weitere Informationen über den DB-Instance Parameter `track_activity_query_size` finden Sie unter Laufzeitstatistik in der PostgreSQL-Dokumentation.

Um die SQL-Textgröße zu erhöhen, erhöhen Sie das `track_activity_query_size`-Limit. Um den Parameter zu ändern, ändern Sie die Parametereinstellung in der Parametergruppe, die der Aurora PostgreSQL-DB-Instance zugeordnet ist.

Ändern Sie die Einstellung wie folgt, wenn die Instance die Standardparametergruppe verwendet:

1. Erstellen Sie eine neue DB-Instance-Parametergruppe für die entsprechende DB-Engine und DB-Engine-Version.
2. Stellen Sie den Parameter in der neuen Parametergruppe ein.
3. Ordnen Sie die neue Parametergruppe der DB-Instance zu.

Informationen über das Einstellen eines DB-Instance-Parameters finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.

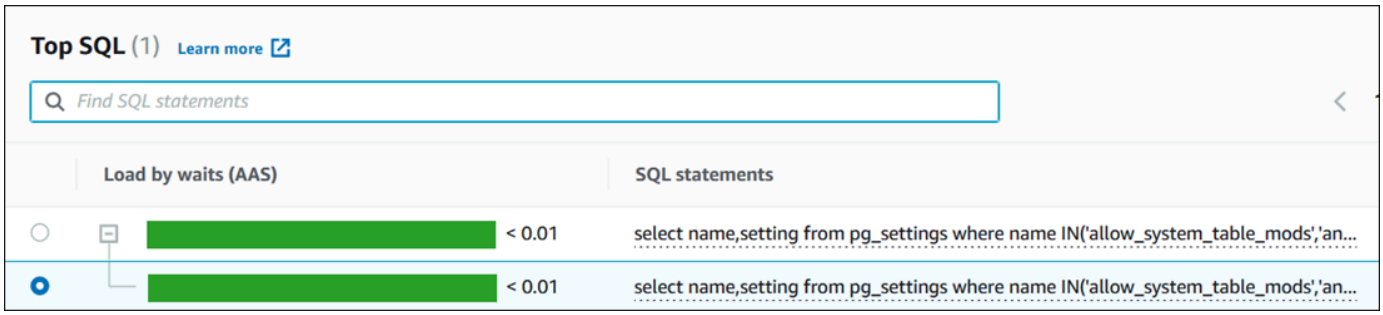
Anzeigen und Herunterladen von SQL-Text im Performance-Insights-Dashboard

Im Performance-Insights-Dashboard können Sie SQL-Text anzeigen oder herunterladen.

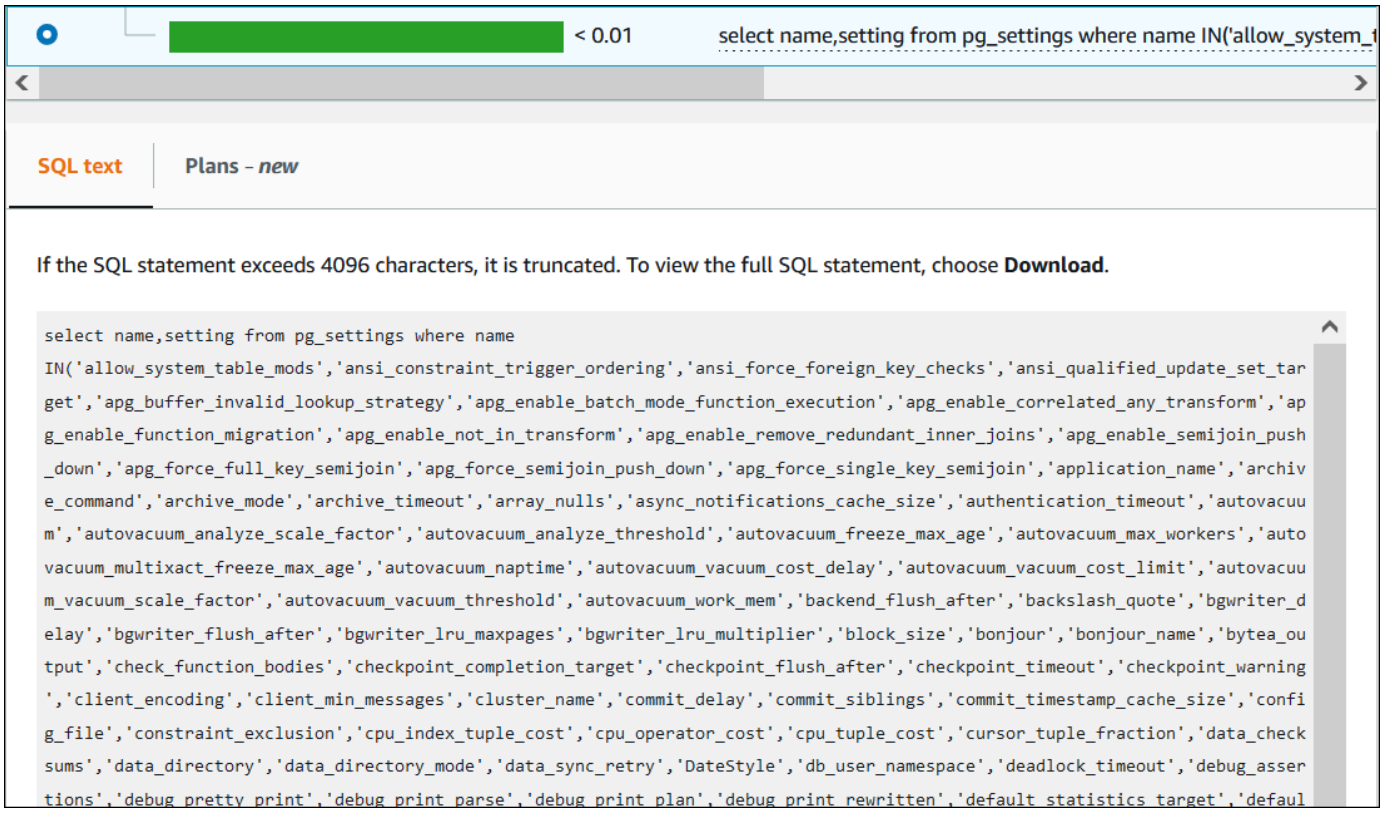
So zeigen Sie mehr SQL-Text im Performance-Insights-Dashboard an

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus.
4. Scrollen Sie im Performance-Insights-Dashboard nach unten zur Registerkarte Top SQL.
5. Wählen Sie das Pluszeichen, um einen SQL-Digest zu erweitern, und eine der untergeordneten Abfragen des Digests aus.

   SQL-Anweisungen mit Text größer als 500 Byte sehen in etwa wie folgt aus.



6. Scrollen Sie nach unten zur Registerkarte SQL text (SQL-Text).



Das Performance-Insights-Dashboard kann bis zu 4.096 Byte für jede SQL-Anweisung anzeigen.

7. (Optional) Wählen Sie Kopieren, um die angezeigte SQL-Anweisung zu kopieren, oder wählen Sie Herunterladen, um die SQL-Anweisung herunterzuladen, um den SQL-Text bis zum Limit der DB-Engine anzuzeigen.

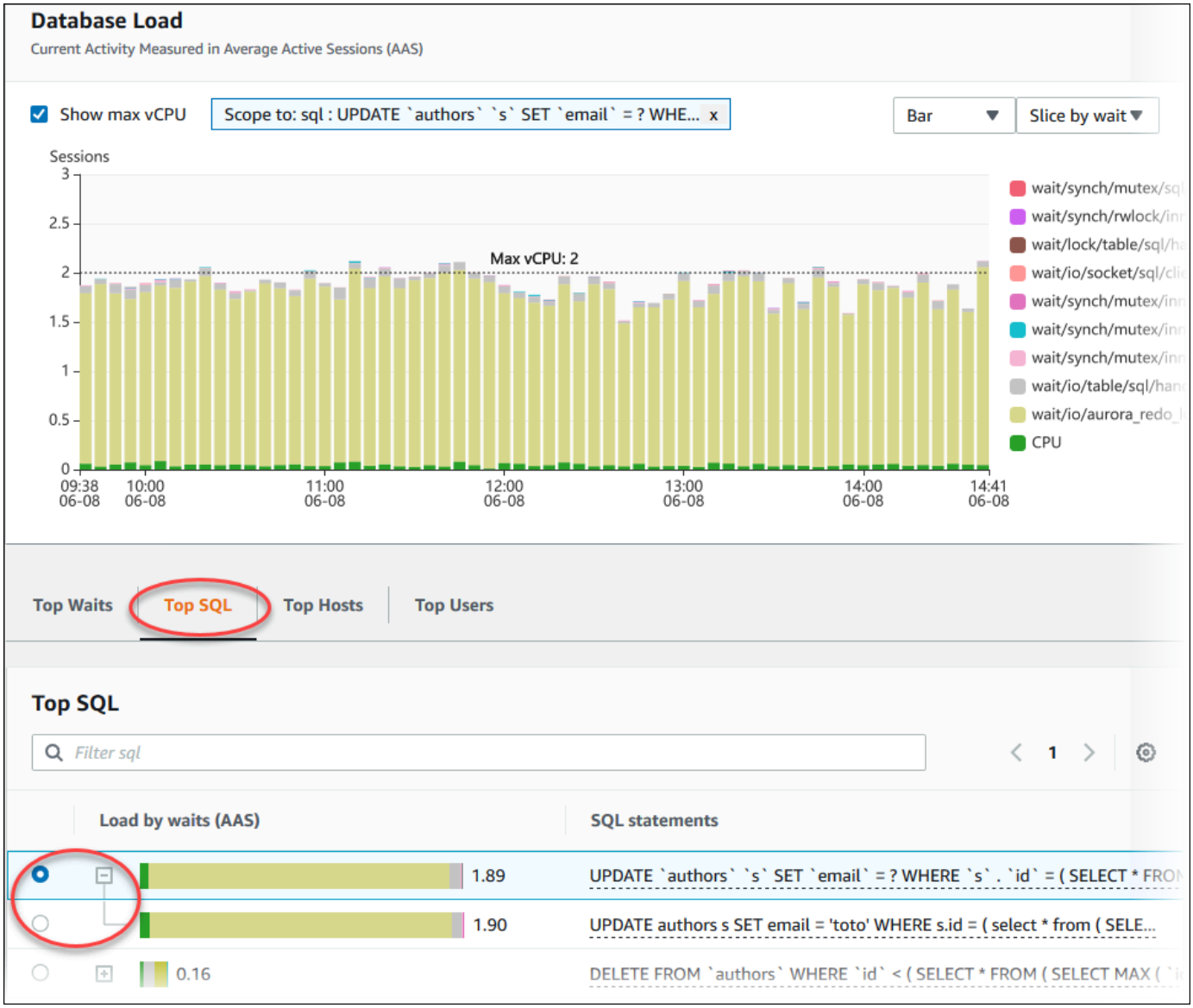
> **Note**
>
> Um die SQL-Anweisung zu kopieren oder herunterzuladen, deaktivieren Sie Popup-Blocker.

Anzeigen von SQL-Statistiken im Performance-Insights-Dashboard

Im Performance-Insights-Dashboard stehen SQL-Statistiken auf dem Tab Top SQL (Haupt-SQL) des Diagramms Database load (Datenbanklast) zur Verfügung.

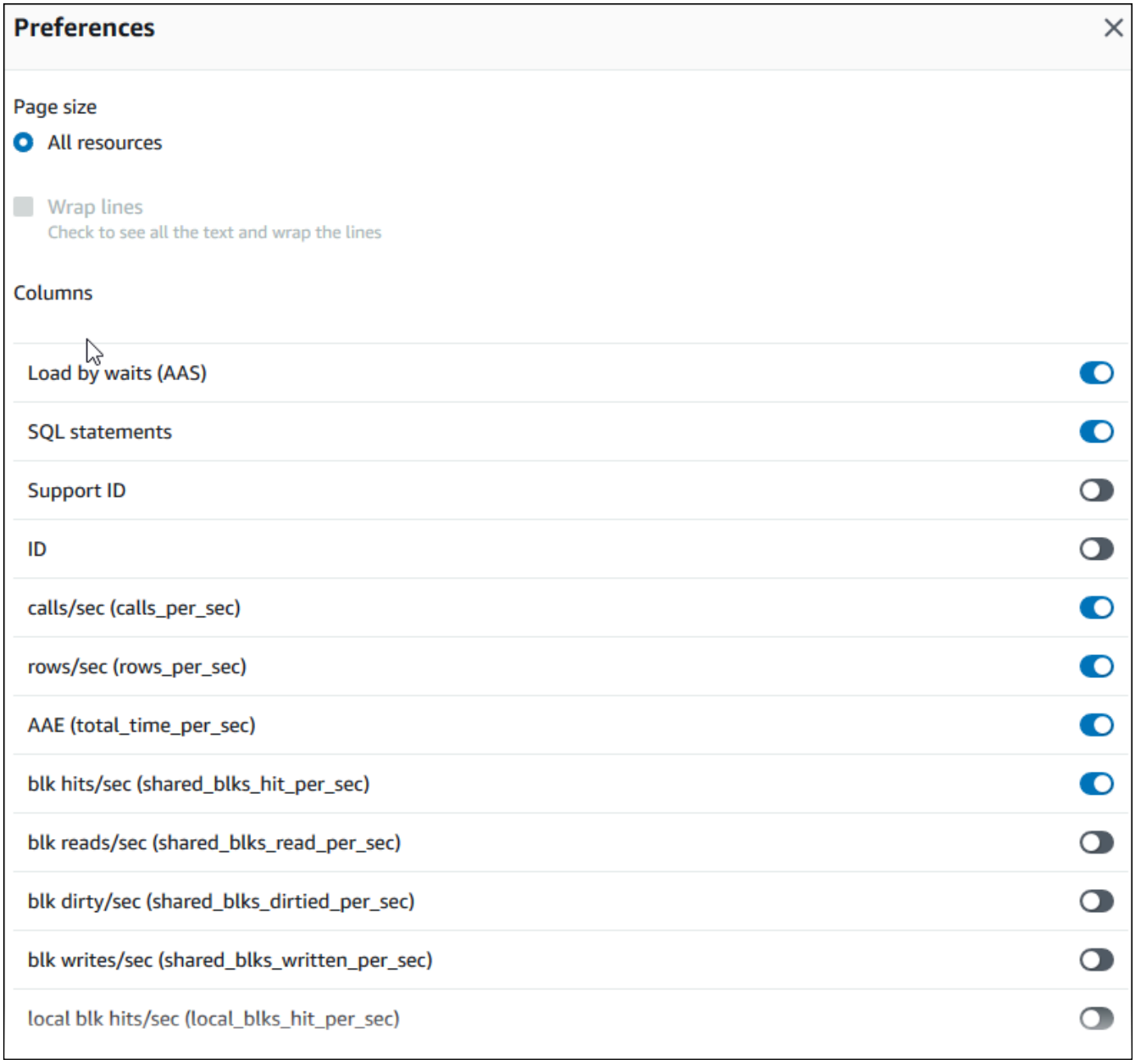
Sehen Sie sich SQL-Statistiken wie folgt an

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Performance Insights aus.
3. Wählen Sie oben auf der Seite die Datenbank aus, deren SQL-Statistiken Sie anzeigen lassen möchten.
4. Scrollen Sie an das Seitenende und klicken Sie auf den Tab Top SQL (Haupt-SQL).
5. Wählen Sie eine individuelle Anweisung (nur Aurora MySQL) oder Digest-Abfrage aus.

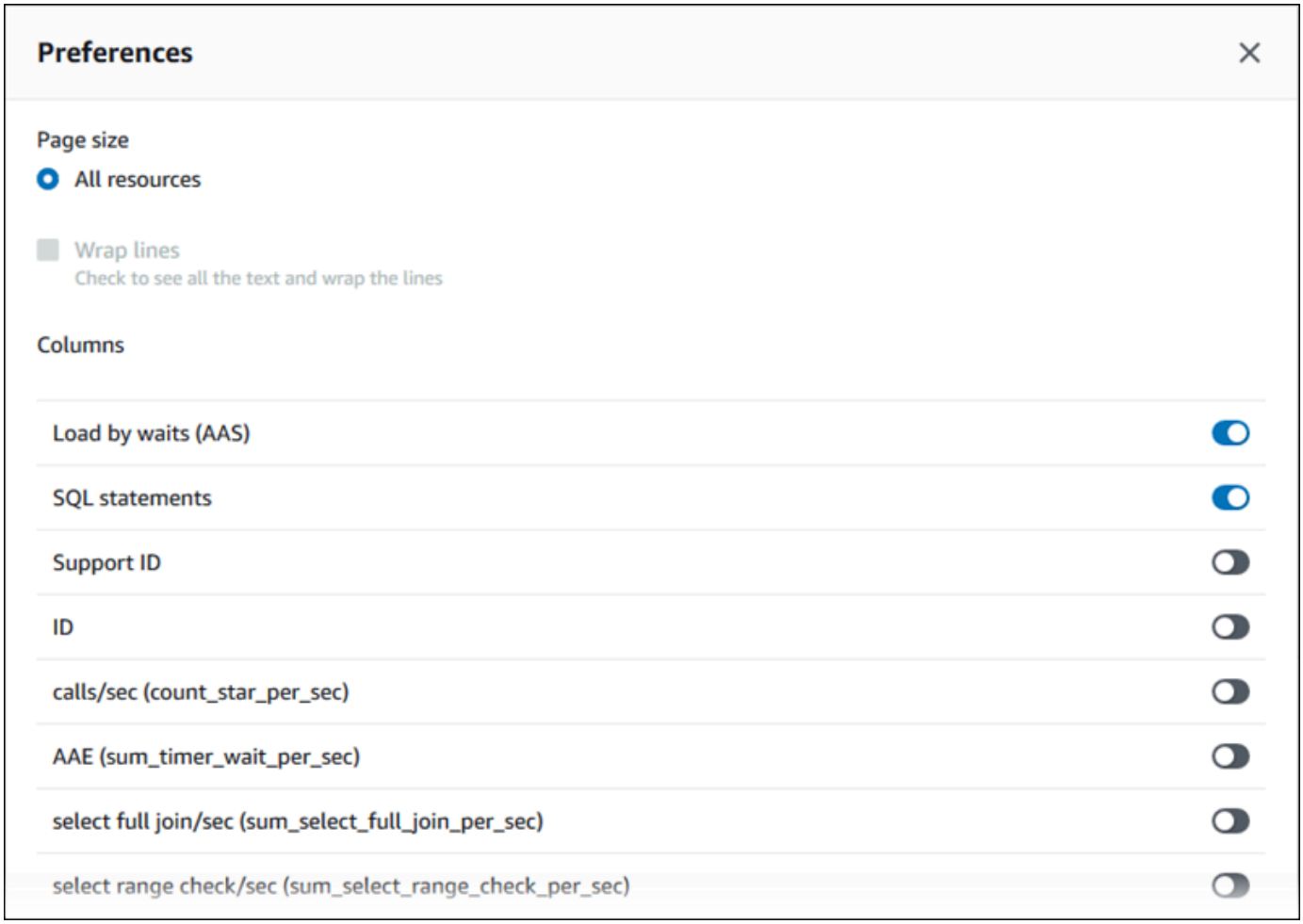


6. Sie können die Statistiken auswählen, die angezeigt werden sollen, indem Sie oben rechts im Diagramm das Zahnradsymbol auswählen. Beschreibungen der SQL-Statistiken für die Amazon RDSAurora-Engines finden Sie unter SQL-Statistiken für Performance Insights.

   Das folgende Beispiel zeigt die Einstellungen für Aurora PostgreSQL.



Das folgende Beispiel zeigt die Einstellungen für Aurora MySQL-DB-Instances.



7. Wählen Sie „Save“ (Speichern) aus, um Ihre Einstellungen zu speichern.

   Die Tabelle Top SQL (Haupt-SQL) wird aktualisiert.

Anzeigen proaktiver Empfehlungen von Performance Insights

Erkenntnisse zur Amazon-RDS-Leistung überwacht bestimmte Metriken und erstellt automatisch Schwellenwerte, indem analysiert wird, welche Werte für eine bestimmte Ressource potenziell problematisch sein könnten. Wenn die neuen Metrikwerte über einen bestimmten Zeitraum einen vordefinierten Schwellenwert überschreiten, generiert Performance Insights eine proaktive Empfehlung. Diese Empfehlung trägt dazu bei, zukünftige Auswirkungen auf die Datenbankleistung zu verhindern. Um diese proaktiven Empfehlungen zu erhalten, müssen Sie Performance Insights mit einer Aufbewahrungsfrist für ein kostenpflichtiges Kontingent aktivieren.

Weitere Informationen zum Aktivieren von Performance Insights finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora. Weitere Informationen zur Preisgestaltung und

Datenaufbewahrung für Performance Insights finden Sie unter Preisgestaltung und Datenspeicherung für Performance Insights.

Informationen zu den Regionen, DB-Engines und Instance-Klassen, die für die proaktiven Empfehlungen unterstützt werden, finden Sie unter DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance-Insights-Funktionen.

Die detaillierte Analyse und die empfohlenen Untersuchungen der proaktiven Empfehlungen finden Sie auf der Seite mit den Empfehlungsdetails.

Weitere Informationen zu Empfehlungen finden Sie unter Empfehlungen von Amazon Aurora.

So zeigen Sie die detaillierte Analyse einer proaktiven Empfehlung an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Führen Sie im Navigationsbereich eine der folgenden Aktionen aus:
   - Wählen Sie Empfehlungen aus.

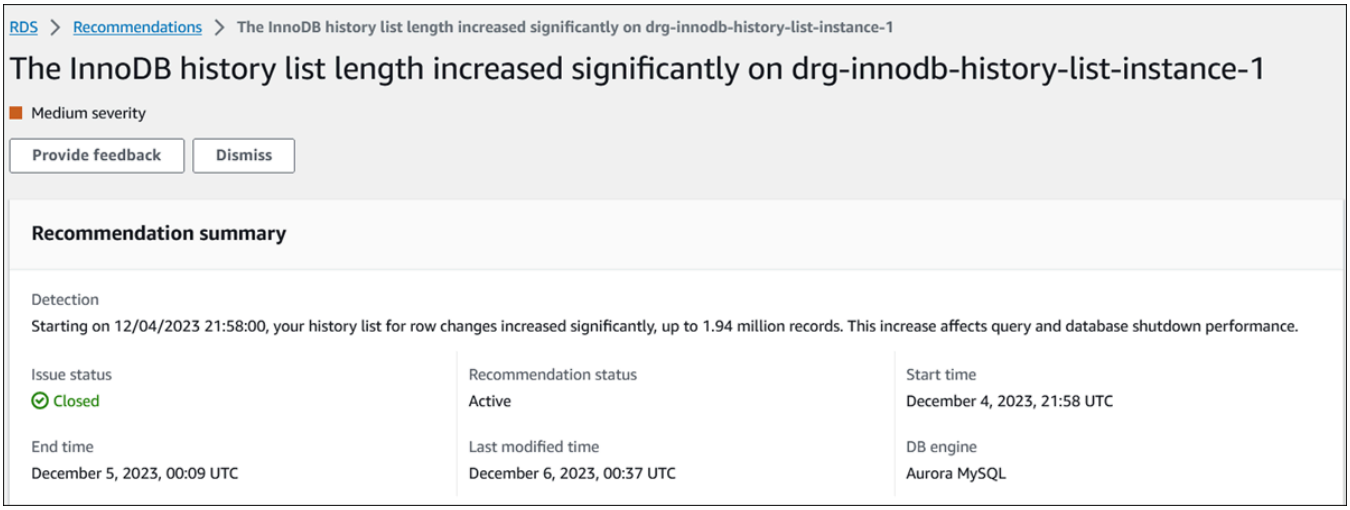
     Auf der Seite Empfehlungen wird eine nach Schweregrad sortierte Liste mit Empfehlungen für alle Ressourcen in Ihrem Konto angezeigt.
   - Wählen Sie Datenbanken und dann Empfehlungen für eine Ressource auf der Datenbankseite aus.

     Die Registerkarte Empfehlungen zeigt die Empfehlungen und entsprechende Details für die ausgewählte Ressource an.
3. Suchen Sie nach einer proaktiven Empfehlung und wählen Sie Details anzeigen aus.

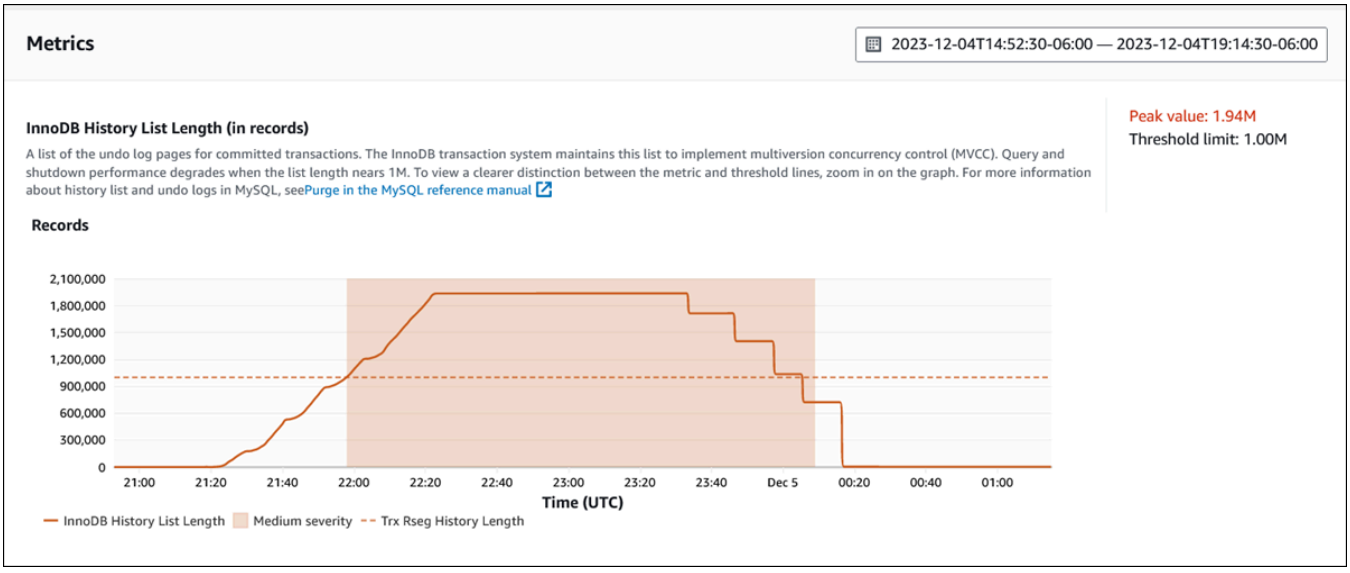
   Die Detailseite zu den Empfehlungen wird angezeigt. Der Titel enthält den Namen der betroffenen Ressource mit dem erkannten Problem und dem Schweregrad.

   Auf der Seite mit den Empfehlungsdetails finden Sie die folgenden Komponenten:

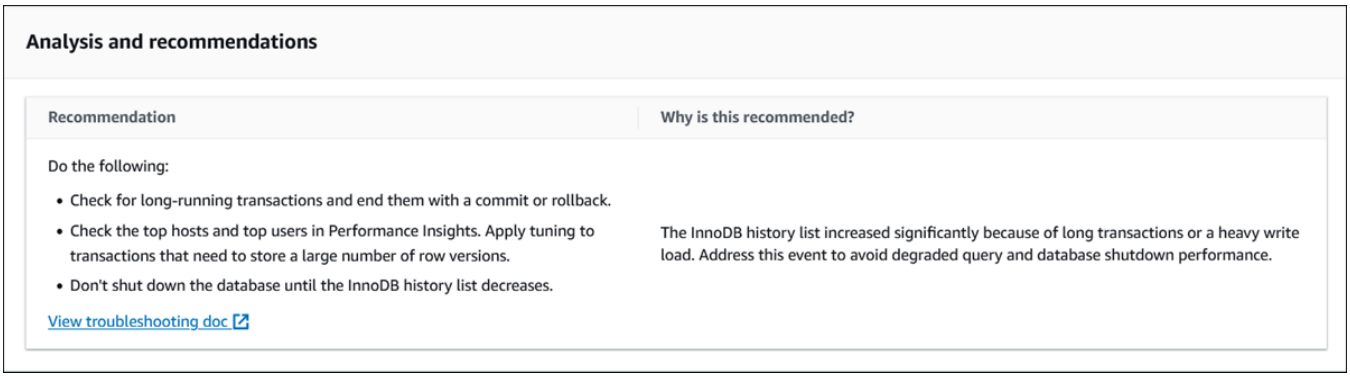
   - Zusammenfassung der Empfehlungen — Das erkannte Problem, Empfehlung und Problemstatus, Start- und Endzeit des Problems, Änderungszeit der Empfehlung und Engine-Typ.



- Metriken – Die Diagramme des erkannten Problems. Jedes Diagramm zeigt einen Schwellenwert, der durch das Basisverhalten der Ressource und die Daten der Metrik bestimmt wird, die ab der Startzeit des Problems gemeldet wurden.



- Analyse und Empfehlungen – Die Empfehlung und der Grund für die vorgeschlagene Empfehlung.



Sie können die Ursache des Problems überprüfen und dann die empfohlenen Maßnahmen zur Behebung des Problems ergreifen oder oben rechts auf Ablehnen klicken, um die Empfehlung abzulehnen.

Abrufen von Metriken mit der Performance-Insights-API für Aurora

Wenn Performance Insights aktiviert ist, bietet die API Einblicke in die Instance-Leistung. Amazon CloudWatch Logs stellt die maßgebliche Quelle für Verkaufsüberwachungsmetriken für AWS-Services bereit.

Performance Insights bietet eine domänenspezifische Ansicht der Datenbanklast, gemessen als durchschnittliche aktive Sitzungen (AAS). Diese Metrik erscheint API-Verbrauchern als zweidimensionaler Zeitreihendatensatz. Die Zeitdimension der Daten stellt die Datenbanklastdaten für jeden Zeitpunkt im abgefragten Zeitraum bereit. Für jeden Zeitpunkt wird die Gesamtlast bezogen auf die angeforderten Dimensionen zerlegt, z. B. `SQL`, `Wait-event`, `User` oder `Host`, gemessen zum betreffenden Zeitpunkt.

Amazon RDS Performance Insights überwacht Ihren , damit Sie die Datenbankleistung analysieren und beheben können. Eine Möglichkeit zum Anzeigen von Performance-Insights-Daten bietet die AWS-Managementkonsole. Performance Insights stellt außerdem eine öffentliche API bereit, sodass Sie Ihre eigenen Daten abfragen können. Sie können die API für Folgendes verwenden:

- Auslagern von Daten in eine Datenbank
- Hinzufügen von Performance-Insights-Daten zu bestehenden Überwachungs-Dashboards
- Entwickeln von Überwachungstools

Zum Verwenden der Performance-Insights-API aktivieren Sie Performance Insights auf einer Ihrer Amazon-RDS-DB-Instances. Weitere Informationen zum Aktivieren von Performance Insights finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora. Weitere Informationen zur Performance-Insights-API finden Sie in der Referenz zur Amazon RDS Performance-Insights-API.

Die Performance-Insights-API bietet die folgenden Operationen.

Performance-Insights-Aktion	AWS CLI command	Beschreibung
CreatePerformanceAnalysisReport	aws pi create-performance-analysis-report	Erstellt einen Leistungsanalysebericht für die DB-Instance für einen bestimmten Zeitraum. Das Ergebnis lautet `AnalysisReportId`. Dies ist der eindeutige Bezeichner des Berichts.
DeletePerformanceAnalysisReport	aws pi delete-performance-analysis-report	Löscht einen Leistungsanalysebericht.
DescribeDimensionKeys	aws pi describe-dimension-keys	Ruft die Schlüssel der Top N-Dimension für eine Metrik für einen bestimmten Zeitraum ab.
GetDimensionKeyDetails	aws pi get-dimension-key-details	Ruft die Attribute der angegebenen Dimensionsgruppe für eine DB-Instance oder Datenquelle ab. Wenn Sie beispielsweise eine SQL-ID angeben und die Dimensionsdetails sind verfügbar, ruft `GetDimensionKeyDetails` den Volltext der Dimension `db.sql.statement` ab, die mit dieser ID verknüpft ist. Dieser Vorgang ist nützlich, da `GetResourceMetrics` und `DescribeDimensionKeys` das Abrufen von umfangreichen SQL-Anweisungen nicht unterstützt.

Performance-Insights-Aktion	AWS CLI command	Beschreibung
GetPerformanceAnalysisReport	aws pi get-performance-analysis-report	Ruft den Bericht einschließlich der Erkenntnisse für den Bericht ab. Das Ergebnis umfasst den Berichtsstatus, die Berichts-ID, Details zum Berichtszeitpunkt, Erkenntnisse und Empfehlungen.
GetResourceMetadata	aws pi get-resource-metadata	Rufen Sie die Metadaten für verschiedene Funktionen ab. Die Metadaten könnten beispielsweise darauf hindeuten, dass eine Funktion für eine bestimmte DB-Instance ein- oder ausgeschaltet ist.
GetResourceMetrics	aws pi get-resource-metrics	Ruft Performance-Insights-Metriken für eine Reihe von Datenquellen über einen Zeitraum ab. Sie können spezifische Dimensionsgruppen und Dimensionen bereitstellen und Aggregation und Filterkriterien für jede Gruppe bereitstellen.
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	Rufen Sie die Dimensionen ab, die für jeden angegebenen Metriktyp für eine bestimmte Instance abgefragt werden können.

Performance-Insights-Aktion	AWS CLI command	Beschreibung
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	Rufen Sie alle verfügbaren Metriken der angegebenen Metriktypen ab, die für eine bestimmte DB-Instance abgefragt werden können.
ListPerformanceAnalysisReports	aws pi list-performance-analysis-reports	Ruft alle Analyseberichte ab, die für die DB-Instance verfügbar sind. Die Berichte werden auf der Grundlage der Startzeit jedes Berichts aufgelistet.
ListTagsForResource	aws pi list-tags-for-resource	Listet alle Metadaten-Tags auf, die der Ressource hinzugefügt wurden. Die Liste enthält den Namen und den Wert des Tags.
TagResource	aws pi tag-resource	Fügt einer Amazon-RDS-Ressource Metadaten-Tags hinzu. Das Tag enthält einen Namen und einen Wert.
UntagResource	aws pi untag-resource	Entfernt die Metadaten-Tags von der Ressource.

Weitere Informationen zum Abrufen von Zeitreihenmetriken und AWS CLI-Beispiele für Performance Insights finden Sie in den folgenden Themen.

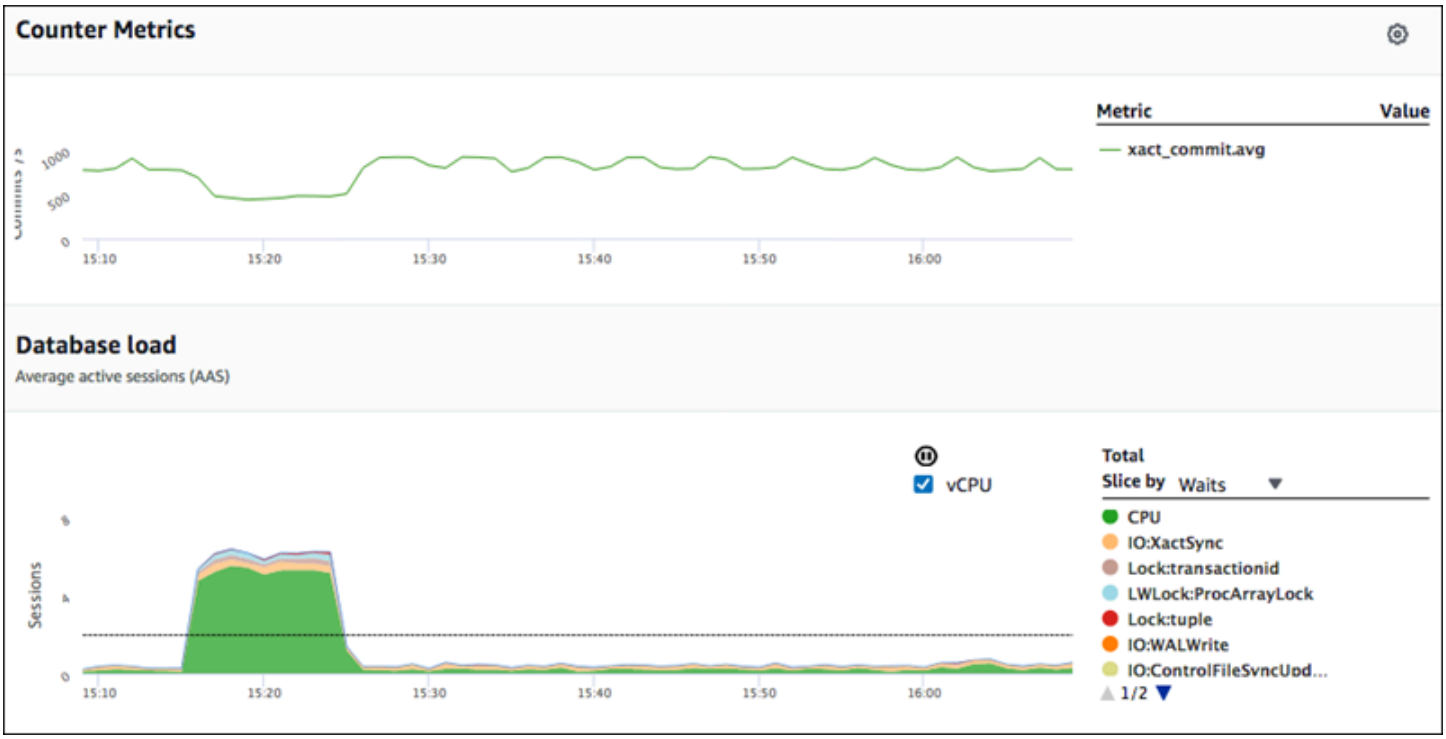
Themen

- Abrufen von Zeitreihenmetriken für Performance Insights
- AWS CLI-Beispiele für Performance-Insights

Abrufen von Zeitreihenmetriken für Performance Insights

Mit der `GetResourceMetrics`-Operation werden ein oder mehrere Zeitreihenmetriken aus den Performance-Insights-Daten abgerufen. Für `GetResourceMetrics` ist eine Metrik und ein Zeitraum erforderlich, damit eine Antwort mit einer Liste von Datenpunkten zurückgegeben wird.

Zum Beispiel dieAWS-Managementkonsole Verwendung`GetResourceMetrics` zum Ausfüllen des Diagramms „Zählermetriken“ und des Datenbank-Load-Diagramms, wie in der folgenden Abbildung dargestellt.



Alle von zurückgegebenen Metriken`GetResourceMetrics` sind Standard-Zeitreihenmetriken, mit Ausnahme von`db.load`. Diese Metrik wird im Diagramm Database Load (Datenbanklast) angezeigt. Die`db.load` Metrik unterscheidet sich von den anderen Zeitreihenmetriken, da Sie sie in Unterkomponenten aufteilen können, die als Dimensionen bezeichnet werden. In der vorherigen Abbildung wird `db.load` unterteilt und nach Wartezuständen gruppiert, aus denen `db.load` besteht.

> **Note**
>
> `GetResourceMetrics` kann auch die `db.sampleload`-Metrik zurückgeben, aber die `db.load`-Metrik ist in den meisten Fällen angemessen.

Informationen zu den Zählermetriken, die von `GetResourceMetrics` zurückgegeben werden, finden Sie unter Performance-Insights-Zählermetriken.

Die folgenden Berechnungen werden für die Metriken unterstützt:

- Durchschnitt – Der durchschnittliche Wert für die Metrik über einen bestimmten Zeitraum. Fügen Sie dem Metriknamen `.avg` an.
- Minimum – Der minimale Wert für die Metrik über einen bestimmten Zeitraum. Fügen Sie dem Metriknamen `.min` an.
- Maximum – Der maximale Wert für die Metrik über einen bestimmten Zeitraum. Fügen Sie dem Metriknamen `.max` an.
- Summe – Die Summe der Metrikwerte über einen bestimmten Zeitraum. Fügen Sie dem Metriknamen `.sum` an.
- Beispielanzahl – Die Anzahl, wie oft die Metrik über einen bestimmten Zeitraum erfasst wurde. Fügen Sie dem Metriknamen `.sample_count` an.

Nehmen wir an, dass eine Metrik beispielsweise 300 Sekunden (5 Minuten) lang erfasst wird und dass die Metrik einmal pro Minute erfasst wird. Die Werte für jede Minute sind 1, 2, 3, 4 und 5. In diesem Fall werden die folgenden Berechnungen zurückgegeben:

- Durchschnitt – 3
- Minimum – 1
- Maximum – 5
- Summe – 15
- Beispielanzahl – 5

Weitere Informationen zur Verwendung des AWS CLI-Befehls `get-resource-metrics` finden Sie unter `get-resource-metrics`.

Geben Sie für die `--metric-queries`-Option eine oder mehrere Abfragen an, um die entsprechenden Ergebnisse zu erhalten. Jede Abfrage besteht aus einem obligatorischen `Metric`- sowie optionalen `GroupBy`- und `Filter`-Parametern. Es folgt ein Beispiel für eine Spezifikation der `--metric-queries`-Option.

```
{
   "Metric": "string",
```

```
  "GroupBy": {
    "Group": "string",
    "Dimensions": ["string", ...],
    "Limit": integer
  },
  "Filter": {"string": "string"
    ...}
```

AWS CLI-Beispiele für Performance-Insights

In den folgenden Abschnitten erfahren Sie mehr über die AWS Command Line Interface (AWS CLI) für Performance Insights und AWS CLI-Anwendungsbeispiele.

Themen

- Integrierte Hilfe für die AWS CLI für Performance Insights
- Abrufen von Zählermetriken
- Abrufen des DB-Lastdurchschnitts für Top-Warteereignisse
- Abrufen des DB-Lastdurchschnitts für Top-SQL-Anweisungen
- Abrufen des nach SQL gefilterten DB-Lastdurchschnitts
- Abrufen des Volltextes einer SQL-Anweisung
- Erstellen eines Leistungsanalyseberichts für einen bestimmten Zeitraum
- Abrufen eines Leistungsanalyseberichts
- Auflisten aller Leistungsanalyseberichte für die DB-Instance
- Löschen eines Leistungsanalyseberichts
- Hinzufügen eines Tags zu einem Leistungsanalysebericht
- Auflisten aller Tags für einen Leistungsanalysebericht
- Löschen der Tags eines Leistungsanalyseberichts

Integrierte Hilfe für die AWS CLI für Performance Insights

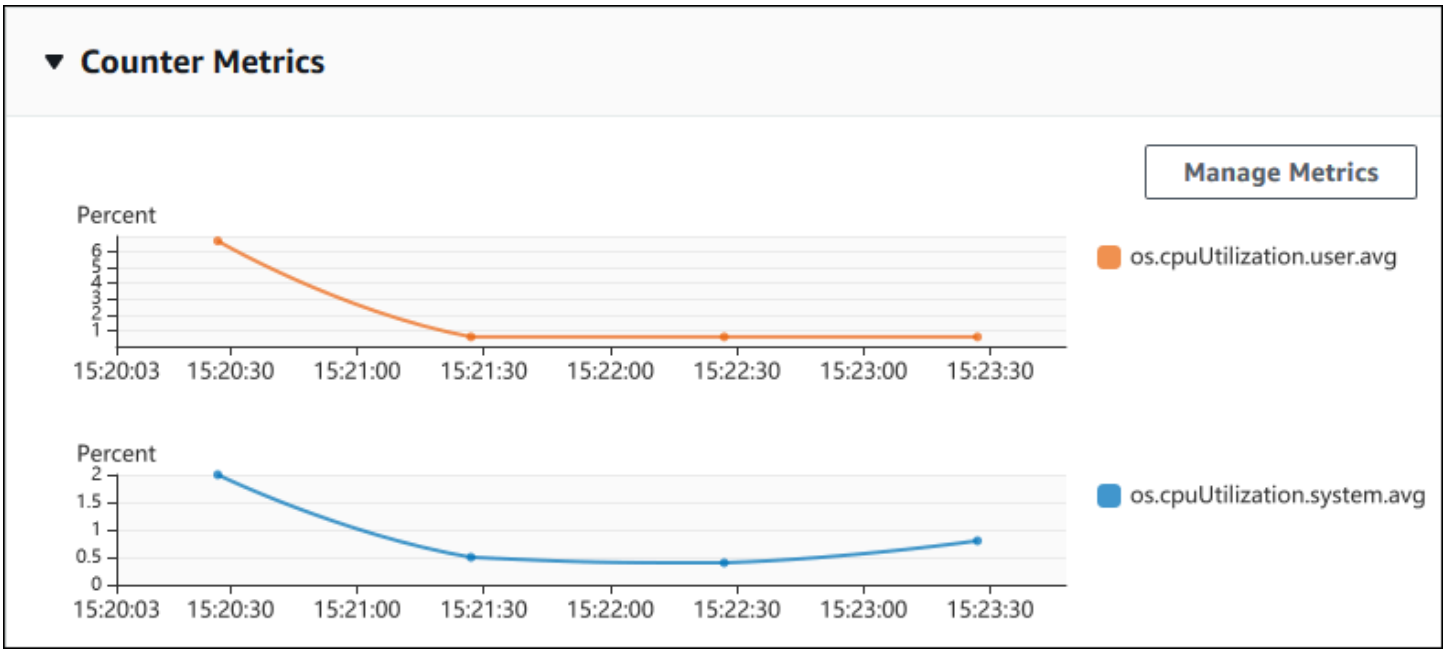
Sie können Performance-Insights-Daten über die anzeige AWS CLI. Hilfe zu den AWS CLI-Befehlen für Performance Insights erhalten Sie durch Eingabe der folgenden Befehle an der Befehlszeile.

```
aws pi help
```

Wenn die AWS CLI nicht installiert ist, finden Sie Informationen zur Installation unter Installing the AWS CLI im AWS CLI-Benutzerhandbuch.

Abrufen von Zählermetriken

Der folgende Screenshot zeigt zwei Zählermetriken-Diagramme in der AWS-Managementkonsole.



Das folgende Beispiel veranschaulicht, wie die Daten, die die AWS-Managementkonsole zum Erstellen der beiden Zählermetriken-Diagramme verwendet, gesammelt werden.

Für Linux, macOS oder Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Für Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
```

```
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Sie können einen Befehl besser lesbar gestalten, indem Sie eine Datei für die Option `--metrics-query` angeben. Im folgenden Beispiel wird eine Datei namens query.json für die Option verwendet. Die Datei enthält Folgendes.

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

Führen Sie den folgenden Befehl aus, um die Datei zu verwenden.

Für Linux, macOS oder Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Für Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Das vorige Beispiel gibt die folgenden Werte für die Optionen an:

- `--service-type` – `RDS` für Amazon RDS
- `--identifier` – Die Ressource-ID für die DB-Instance
- `--start-time` und `--end-time` – Die ISO 8601-Werte `DateTime` für den abzufragenden Zeitraum mit mehreren unterstützten Formaten

Der Abfragezeitraum beträgt eine Stunde:

- `--period-in-seconds` – `60` für eine Abfrage pro Minute
- `--metric-queries` – Ein Array mit zwei Abfragen, jeweils für nur eine Metrik.

  Der Metrikname verwendet Punkte, um die Metrik in eine sinnvolle Kategorie einzustufen, wobei das letzte Element eine Funktion ist. Im Beispiel lautet die Funktion `avg` für jede Abfrage. Wie bei Amazon CloudWatch werden die Funktionen `min`, `max`, `total` und `avg` unterstützt.

Die Antwort sieht in etwa so aus:

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
```

```
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

Die Antwort enthält Werte für `Identifier`, `AlignedStartTime` und `AlignedEndTime`. Bei einem `--period-in-seconds`-Wert von `60` wurden Start- und Endzeiten auf die Minute ausgerichtet. Wenn der `--period-in-seconds`-Wert `3600` lautet, werden Start- und Endzeiten auf die Stunde ausgerichtet.

Die `MetricList` in der Antwort enthält eine Reihe von Einträgen, und zwar jeweils mit einem `Key`- und einem `DataPoints`-Eintrag. Jeder `DataPoint` verfügt über einen `Timestamp` und einen `Value`. Jede `Datapoints`-Liste enthält 60 Datenpunkte, da die Abfragen eine Stunde lang jede Minute Daten abfragen, und zwar mit den Werten `Timestamp1/Minute1`, `Timestamp2/Minute2` usw. bis `Timestamp60/Minute60`.

Da sich die Abfrage auf zwei verschiedene Zählermetriken bezieht, enthält die -Antwort zwei Element `MetricList`.

Abrufen des DB-Lastdurchschnitts für Top-Warteereignisse

Das folgende Beispiel entspricht der Abfrage, die die AWS-Managementkonsole zum Erstellen eines Stapelflächendiagramms verwendet. Mit diesem Beispiel wird der `db.load.avg`-Wert für die letzte Stunde abgerufen, wobei die Last auf die sieben Top-Warteereignisse aufgeteilt ist. Der Befehl ist

mit dem Befehl unter identisc Abrufen von Zählermetriken. Die Datei query.json enthält hingegen Folgendes.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]
```

Führen Sie den folgenden Befehl aus.

Für Linux, macOS oder Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Für Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Das Beispiel gibt die Metrik `db.load.avg` und eine `GroupBy`-Sortierung der sieben Top-Warteereignisse an. Details zu gültigen Werten für dieses Beispiel finden Sie unter DimensionGroup in der Performance-Insights-API-Referenz.

Die Antwort sieht in etwa so aus:

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
```

```
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
```

```
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

In dieser Antwort gibt es acht Einträge in der `MetricList`. Davon bezieht sich ein Eintrag auf den `db.load.avg`-Gesamtwert und sieben Einträge jeweils auf den `db.load.avg`-Wert, der auf eines der sieben Top-Warteereignisse aufgeteilt ist. Im Gegensatz zum ersten Beispiel, bei dem eine Gruppierungsdimension vorlag, muss für jede Gruppierung der Metrik ein Schlüssel vorliegen. Für jede Metrik kann nicht nur ein Schlüssel vorhanden sein, wie im Anwendungsfall der Basiszählermetrik.

Abrufen des DB-Lastdurchschnitts für Top-SQL-Anweisungen

Im folgenden Beispiel werden `db.wait_events` entsprechend der 10 Top-SQL-Anweisungen gruppiert. Es gibt zwei verschiedene Gruppen für SQL-Anweisungen.

- `db.sql` – Die vollständige SQL-Anweisung, wie `select * from customers where customer_id = 123`
- `db.sql_tokenized` – Die SQL-Anweisung mit Token, wie `select * from customers where customer_id = ?`

Beim Analysieren der Datenbank-Performance kann es nützlich sein, SQL-Anweisungen, die sich nur durch ihre Parameter unterscheiden, als ein logisches Element zu betrachten. In diesem Fall können Sie `db.sql_tokenized` beim Abfragen verwenden. In manchen Fällen, insbesondere wenn Sie an Explain-Plänen interessiert sind, ist es jedoch sinnvoller, die vollständigen SQL-Anweisungen mit Parametern zu untersuchen und die Abfrage nach `db.sql` zu gruppieren. Zwischen SQL-Anweisungen mit Token und vollständigen SQL-Anweisungen besteht eine Beziehung der Über-/Unterordnung. Mehrere vollständige (untergeordnete) SQL-Anweisungen befinden sich unter derselben (übergeordneten) SQL-Anweisung mit Token.

Der Befehl in diesem Beispiel ähnelt dem Befehl unter Abrufen des DB-Lastdurchschnitts für Top-Warteereignisse. Die Datei query.json enthält hingegen Folgendes.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
```

```
]
```

Im folgenden Beispiel wird verwende `db.sql_tokenized`.

Für Linux, macOS oder Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-29T00:00:00Z \
   --end-time   2018-10-30T00:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

Für Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-29T00:00:00Z ^
   --end-time   2018-10-30T00:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

In diesem Beispiel läuft die Abfrage über 24 Stunden, mit einem einstündigen Zeitraum in Sekunden.

Das Beispiel gibt die Metrik `db.load.avg` und eine `GroupBy`-Sortierung der sieben Top-Warteereignisse an. Details zu gültigen Werten für dieses Beispiel finden Sie unter DimensionGroup in der Performance-Insights-API-Referenz.

Die Antwort sieht in etwa so aus:

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
```

```
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

Diese Antwort umfasst 11 Einträge in der `MetricList` (1 gesamt, 10 Top-SQL mit Token), wobei jeder Eintrage 24 `DataPoints` pro Stunde aufweist.

Für SQL-Anweisungen mit Token gibt es in jeder Dimensionsliste drei Einträge:

- `db.sql_tokenized.statement` – Die SQL-Anweisung mit Token.
- `db.sql_tokenized.db_id` – Entweder die native Datenbank-ID zum Verweisen auf die SQL-Anweisung oder eine synthetische ID, die von Performance Insights generiert wird, wenn keine native Datenbank-ID verfügbar ist. In diesem Beispiel wird die synthetische ID `pi-2372568224` zurückgegeben.
- `db.sql_tokenized.id` – Die ID der Abfrage innerhalb von Performance-Insights.

  In der AWS-Managementkonsole wird diese ID als Support-ID bezeichnet. Sie heißt so, weil es sich bei der ID um Daten handelt, die der AWS Support untersuchen kann, um Ihnen bei der Behebung eines Problems mit Ihrer Datenbank zu helfen. AWS nimmt die Sicherheit

und den Schutz Ihrer Daten sehr ernst. Fast alle Daten werden zur Aufbewahrung mit Ihrem AWS KMS-Schlüssel verschlüsselt. Daher sind diese Daten für keinen Benutzer innerhalb von AWS einsehbar. Im vorherigen Beispiel wird sowohl `tokenized.statement` als auch `tokenized.db_id` verschlüsselt gespeichert. Bei einem Problem mit Ihrer Datenbank kann der AWS Support Sie anhand der Support-ID unterstützen.

Beim Abfragen empfiehlt es sich ggf., eine `Group` in `GroupBy` anzugeben. Für eine präzisere Kontrolle der Daten, die zurückgegeben werden, sollten Sie allerdings die Dimensionsliste angeben. Wenn z. B. lediglich eine `db.sql_tokenized.statement` erforderlich ist, kann der query.json-Datei ein `Dimensions`-Attribut hinzugefügt werden.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

Abrufen des nach SQL gefilterten DB-Lastdurchschnitts

Die vorherige Abbildung zeigt, dass eine bestimmte Abfrage ausgewählt ist, und das Stapelflächendiagramm der durchschnittlichen aktiven Top-Sitzungen ist auf diese Abfrage beschränkt. Obwohl sich die Abfrage nach wie vor auf die sieben Top-Gesamtwarteereignisse bezieht, wird der Wert der Antwort gefiltert. Durch das Filtern werden nur die Sitzungen berücksichtigt, die mit dem entsprechenden Filter übereinstimmen.

Die entsprechende API-Abfrage in diesem Beispiel ähnelt dem Befehl unter Abrufen des DB-Lastdurchschnitts für Top-SQL-Anweisungen. Die Datei query.json enthält hingegen Folgendes.

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

Für Linux, macOS oder Unix:

```
aws pi get-resource-metrics \
```

```
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Für Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Die Antwort sieht in etwa so aus:

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
```

```
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05163360560093349
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
```

```
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
```

```
                }
            ]
        }
    ],
    "AlignedEndTime": 1556222400.0
} //end of response
```

In dieser Antwort werden alle Werte gemäß des Beitrags der SQL-Anweisung mit Token AKIAIOSFODNN7EXAMPLE gefiltert, die in der query.json-Datei angegeben ist. Die Schlüssel sind möglicherweise in einer anderen Reihenfolge angeordnet als bei einer Abfrage ohne Filter, da die gefilterte SQL-Anweisung von den fünf Top-Warteereignisse beeinflusst wurde.

Abrufen des Volltextes einer SQL-Anweisung

Im folgenden Beispiel wird der Volltext einer SQL-Anweisung für die DB-Instance `db-10BCD2EFGHIJ3KL4M5NO6PQRS5` abgerufen. `--group` ist `db.sql` und `--group-identifier` ist `db.sql.id`. In diesem Beispiel stellt *`my-sql-id`* eine SQL-ID dar, die durch Aufrufen von`pi get-resource-metrics` oder `pi describe-dimension-keys` abgerufen wurde.

Führen Sie den folgenden Befehl aus.

Für Linux, macOS oder Unix:

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
   --group db.sql \
   --group-identifier my-sql-id \
   --requested-dimensions statement
```

Für Windows:

```
aws pi get-dimension-key-details ^
   --service-type RDS ^
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
   --group db.sql ^
   --group-identifier my-sql-id ^
   --requested-dimensions statement
```

In diesem Beispiel sind die Dimensionsdetails verfügbar. Performance Insights ruft also den vollständigen Text der SQL-Anweisung ab, ohne ihn abzuschneiden.

```
{
    "Dimensions":[
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
    },
    ...
    ]
}
```

Erstellen eines Leistungsanalyseberichts für einen bestimmten Zeitraum

Im folgenden Beispiel wird ein Leistungsanalysebericht mit der Startzeit `1682969503` und der Endzeit `1682979503` für die`db-loadtest-0`-Datenbank erstellt.

```
aws pi create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --region us-west-2
```

Die Antwort ist der eindeutige Bezeichner `report-0234d3ed98e28fb17` für den Bericht.

```
{
   "AnalysisReportId": "report-0234d3ed98e28fb17"
}
```

Abrufen eines Leistungsanalyseberichts

Im folgenden Beispiel werden die Details des Analyseberichts für den Bericht `report-0d99cc91c4422ee61` abgerufen.

```
aws pi get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

Die Antwort enthält den Berichtsstatus, die ID, Zeitdetails und Einblicke.

```
        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```

Auflisten aller Leistungsanalyseberichte für die DB-Instance

Das folgende Beispiel listet alle verfügbaren Leistungsanalyseberichte für die `db-loadtest-0`-Datenbank auf.

```
aws pi list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--region us-west-2
```

In der Antwort werden alle Berichte mit der Berichts-ID, dem Status und den Details zum Zeitraum aufgeführt.

```
{
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
            "Status": "Succeeded",
```

```
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

Löschen eines Leistungsanalyseberichts

Im folgenden Beispiel wird der Analysebericht für die `db-loadtest-0`-Datenbank gelöscht.

```
aws pi delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

Hinzufügen eines Tags zu einem Leistungsanalysebericht

Im folgenden Beispiel wird ein Tag mit dem Schlüssel `name` und dem Wert `test-tag` zum Bericht `report-01ad15f9b88bcbd56` hinzugefügt.

```
aws pi tag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
```

```
--region us-west-2
```

Auflisten aller Tags für einen Leistungsanalysebericht

Das folgende Beispiel listet alle Tags für den Bericht `report-01ad15f9b88bcbd56` auf.

```
aws pi list-tags-for-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--region us-west-2
```

In der Antwort werden der Wert und der Schlüssel für alle dem Bericht hinzugefügten Tags aufgeführt:

```
{
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

Löschen der Tags eines Leistungsanalyseberichts

Im folgenden Beispiel wird das Tag `name` aus dem Bericht `report-01ad15f9b88bcbd56` gelöscht.

```
aws pi untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--region us-west-2
```

Nachdem das Tag gelöscht wurde, wird beim Abrufen der API `list-tags-for-resource` dieses Tag nicht mehr aufgelistet.

Protokollieren von Performance-Insights-An AWS CloudTrail

Performance Insights wird mit AWS CloudTrail ausgeführt, einem Service, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS-Service in Performance

Insights ausgeführt werden. CloudTrail erfasst alle API-Aufrufe für Performance Insights als Ereignisse. Diese Erfassung umfasst Aufrufe von der Amazon-RDS-Konsole und von Code-Aufrufen an die Performance-Insights-API-Operationen.

Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail-Ereignissen an einen Amazon S3-Bucket, einschließlich Ereignissen für Performance Insights, aktivieren. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse in der CloudTrail-Konsole trotzdem in Ereignisverlauf anzeigen. Anhand der von CloudTrail erfassten Informationen können Sie bestimmte Details festlegen. Diese Informationen umfassen die Anforderung an Performance Insights, die IP-Adresse, von der die Anforderung gesendet wurde, den Initiator sowie den Zeitpunkt der Anforderung. Sie enthalten auch weitere Angaben.

Weitere Informationen zu CloudTrail finden Sie im AWS CloudTrail-Benutzerhandbuch.

Arbeiten mit Performance-Insights-Informationen in CloudTrail

CloudTrail wird beim Erstellen Ihres AWS-Kontos für Sie aktiviert. Wenn Aktivität in Performance-Insights auftritt, wird diese zusammen mit anderen AWS-Service-Ereignissen in der CloudTrail-Konsole in einem CloudTrail-Ereignisprotokoll im Ereignisverlauf aufgezeichnet. Sie können die neusten Ereignisse in Ihr AWS-Konto herunterladen und dort suchen und anzeigen. Weitere Informationen finden Sie unter Anzeigen von Ereignissen mit dem CloudTrail-Ereignisverlauf im AWS CloudTrail-Benutzerhandbuch.

Erstellen Sie einen Trail für einen fortlaufenden Datensatz zu Ereignissen in Ihrem AWS-Konto, einschließlich Ereignissen für Performance Insights. Ein Trail ermöglicht es CloudTrail, Protokolldateien in einem Amazon-S3-Bucket bereitzustellen. Wenn Sie einen Trail in der Konsole anlegen, gilt dieser für alle AWS-Regionen. Der Trail protokolliert Ereignisse aus allen AWS-Regionen in der AWS-Partition und stellt die Protokolldateien in dem Amazon-S3-Bucket bereit, den Sie angeben. Darüber hinaus können Sie andere AWS-Services konfigurieren, um die in den CloudTrail-Protokollen erfassten Ereignisdaten weiter zu analysieren und entsprechend zu agieren. Weitere Informationen finden Sie in folgenden Themen im AWS CloudTrail-Benutzerhandbuch:

- Übersicht zum Erstellen eines Trails
- Von CloudTrail unterstützte Services und Integrationen
- Konfigurieren von Amazon-SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail-Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail-Protokolldateien aus mehreren Konten

Alle Performance-Insights-Vorgänge werden von CloudTrail protokolliert und in der Performance-Insights-API-Referenz dokumentiert. Zum Beispiel werden durch Aufrufe der `DescribeDimensionKeys`- und `GetResourceMetrics`-Operationen Einträge in den CloudTrail-Protokolldateien generiert.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Gibt an, ob die Anforderung mit Root- oder IAM-Benutzer-Anmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde

Weitere Informationen finden Sie unter dem CloudTrail userIdentity-Objekt.

Performance-Insights-Protokolldateieinträge

Ein Trail ist eine Konfiguration, durch die Ereignisse als Protokolldateien an den von Ihnen angegebenen Amazon-S3-Bucket übermittelt werden. CloudTrail-Protokolldateien können einen oder mehrere Einträge enthalten. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar. Jedes Ereignis enthält unter anderem Informationen über die angeforderte Operation, etwaige Anforderungsparameter und das Datum und die Uhrzeit der Operation. CloudTrail-Protokolleinträge sind kein geordnetes Stack-Trace der öffentlichen API-Aufrufe und erscheinen daher in keiner bestimmten Reihenfolge.

Das folgende Beispiel zeigt einen CloudTrail-Protokolleintrag, der die `GetResourceMetrics`-Operation demonstriert:

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
```

```
    "eventSource": "pi.amazonaws.com",
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
        "periodInSeconds": 60,
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


Performance-Insights-API und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)

Sie können mit AWS PrivateLink eine private Verbindung zwischen Ihrer VPC und Erkenntnisse zur Amazon-RDS-Leistung herstellen. Sie können auf Performance Insights zugreifen, als wäre es in Ihrer VPC, ohne die Verwendung eines Internet-Gateways, NAT-Geräts, einer VPN-Verbindung oder einer Direct Connect-Verbindung. Instances in Ihrer VPC benötigen keine öffentlichen IP-Adressen, um auf Performance Insights zuzugreifen.

Sie stellen diese private Verbindung her, indem Sie einen Schnittstellen-Endpunkt erstellen, der von AWS PrivateLink unterstützt wird. Wir erstellen eine Endpunkt-Netzwerkschnittstelle in jedem Subnetz, das Sie für den Schnittstellen-Endpunkt aktivieren. Hierbei handelt es sich um vom Anforderer verwaltete Netzwerkschnittstellen, die als Eingangspunkt für den Datenverkehr dienen, der für Performance Insights bestimmt ist.

Weitere Informationen finden Sie unter Zugriff auf AWS-Services über AWS PrivateLink im AWS PrivateLink-Leitfaden.

Überlegungen zu Performance Insights

Bevor Sie einen Schnittstellenendpunkt für Performance Insights einrichten, lesen Sie die Überlegungen im AWS PrivateLink-Leitfaden.

Performance Insights unterstützt Aufrufe an alle seine API-Aktionen über den Schnittstellenendpunkt.

Standardmäßig ist der vollständige Zugriff auf Performance Insights über den Endpunkt zulässig. Um den Datenverkehr zu Performance Insights über den Schnittstellenendpunkt zu steuern, ordnen Sie den Endpunkt-Netzwerkschnittstellen eine Sicherheitsgruppe zu.

Verfügbarkeit

Die Performance-Insights-API unterstützt derzeit VPC-Endpunkte in AWS-Regionen, die Performance Insights unterstützen. Weitere Informationen zu Performance Insights finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Performance Insights.

Erstellen eines Schnittstellenendpunkts für Performance Insights

Sie können einen Schnittstellenendpunkt für Performance Insights entweder über die Amazon-VPC-Konsole oder die AWS Command Line Interface (AWS CLI) erstellen. Weitere Informationen finden Sie unter Erstellen eines Schnittstellenendpunkts im AWS PrivateLink-Leitfaden.

Erstellen Sie einen Schnittstellenendpunkt für Performance Insights mit dem folgenden Service-Namen:

Wenn Sie privates DNS für den Schnittstellenendpunkt aktivieren, können Sie API-Anforderungen an Performance Insights mit dem standardmäßigen regionalen DNS-Namens stellen. Beispiel, `pi.us-east-1.amazonaws.com`.

Erstellen einer VPC-Endpunktrichtlinie für die Performance-Insights-API

Eine Endpunktrichtlinie ist eine IAM-Ressource, die Sie an einen Schnittstellen-Endpunkt anfügen können. Die Standard-Endpunktrichtlinie ermöglicht den vollständigen Zugriff auf Performance Insights über den Schnittstellen-Endpunkt. Um den Zugriff auf Performance Insights von Ihrer VPC aus zu steuern, fügen Sie eine benutzerdefinierte Endpunktrichtlinie an den Schnittstellenendpunkt an.

Eine Endpunktrichtlinie gibt die folgenden Informationen an:

- Die Prinzipale, die Aktionen ausführen können (AWS-Konten, IAM-Benutzer und IAM-Rollen).
- Aktionen, die ausgeführt werden können
- Die Ressourcen, auf denen die Aktionen ausgeführt werden können.

Weitere Informationen finden Sie unter Steuern des Zugriffs auf Services mit Endpunktrichtlinien im AWS PrivateLink-Leitfaden.

Beispiel: VPC-Endpunktrichtlinie für Performance-Insights-Aktionen

Im Folgenden finden Sie ein Beispiel für eine benutzerdefinierte Endpunktrichtlinie. Wenn Sie diese Richtlinie an Ihren Schnittstellenendpunkt anfügen, gewährt sie allen Prinzipalen auf allen Ressourcen den Zugriff auf die aufgeführten Performance-Insights-Aktionen.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreatePerformanceAnalysisReport",
            "rds:DeletePerformanceAnalysisReport",
            "rds:GetPerformanceAnalysisReport"
         ],
         "Resource":"*"
      }
   ]
}
```

VPC-Endpunkt-Richtlinie, die den gesamten Zugriff über ein angegebenes AWS-Konto verweigert

Die folgende VPC-Endpunktrichtlinie verweigert dem AWS-Konto `123456789012` jeglichen Zugriff auf Ressourcen, die den Endpunkt verwenden. Die Richtlinie erlaubt alle Aktionen von anderen Konten.

```
{
  "Statement": [
    {
      "Action": "*",
```

```
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

IP-Adressierung für Performance Insights

IP-Adressen ermöglichen es Ressourcen in Ihrer VPC untereinander und mit Ressourcen im Internet zu kommunizieren. Performance Insights unterstützt sowohl IPv4- als auch IPv6-Adressierungsprotokolle. Standardmäßig verwenden Performance Insights API und Amazon VPC das IPv4-Adressierungsprotokoll. Sie können dieses Standardverhalten nicht deaktivieren. Wenn Sie eine VPC erstellen, müssen Sie einen IPv4 CIDR-Block (einen privaten IPv4-Adressenbereich) angeben.

Sie können Ihrer VPC und Ihren Subnetzen optional einen IPv6-CIDR-Block zuweisen und IPv6-Adressen aus diesem Block den RDS-Ressourcen in Ihrem Subnetz zuweisen. Durch die Unterstützung des IPv6-Protokolls wird die Anzahl der unterstützten IP-Adressen erweitert. Durch die Verwendung des IPv6-Protokolls stellen Sie sicher, dass Sie über ausreichende verfügbare Adressen für das künftige Wachstum des Internets verfügen. Neue und vorhandene RDS-Ressourcen können IPv4- und IPv6-Adressen innerhalb Ihrer VPC verwenden. Das Konfigurieren, Sichern und Übersetzen des Netzwerkverkehrs zwischen den beiden Protokollen, die in verschiedenen Teilen einer Anwendung verwendet werden, können den Betriebsaufwand erhöhen. Sie können das IPv6-Protokoll für Amazon-RDS-Ressourcen standardisieren, um Ihre Netzwerkkonfiguration zu vereinfachen. Weitere Hinweise zu Service-Endpunkten und -Kontingenten finden Sie unter Endpunkte und Kontingente von Amazon Relational Database Service.

Weitere Informationen zur IP-Adressierung von Aurora finden Sie unter Aurora-IP-Adressierung.

Analysieren von Aurora-Leistungsanomalien mit Amazon DevOps Guru für Amazon RDS

Amazon DevOps Guru ist ein vollständig verwalteter Betriebsservice, der Entwicklern und Betreibern hilft, die Leistung und Verfügbarkeit ihrer Anwendungen zu verbessern. DevOpsGuru überträgt Ihnen die Aufgaben im Zusammenhang mit der Identifizierung betrieblicher Probleme, sodass Sie schnell Empfehlungen zur Verbesserung Ihrer Anwendung umsetzen können. Weitere Informationen finden Sie unter Was ist Amazon DevOps Guru? im Amazon DevOps Guru-Benutzerhandbuch.

DevOpsGuru erkennt, analysiert und gibt Empfehlungen für bestehende Betriebsprobleme für alle Amazon RDS-DB-Engines. DevOpsGuru for RDS erweitert diese Funktion, indem es maschinelles Lernen auf Performance Insights Insights-Metriken für Amazon Aurora anwendet. Diese Überwachungsfunktionen ermöglichen es DevOps Guru for RDS, Leistungsengpässe zu erkennen und zu diagnostizieren und spezifische Korrekturmaßnahmen zu empfehlen. DevOpsGuru for RDS kann auch problematische Bedingungen in Ihren Aurora-Datenbanken () erkennen, bevor sie auftreten.

Sie können sich diese Empfehlungen jetzt in der RDS-Konsole ansehen. Weitere Informationen finden Sie unter Empfehlungen von Amazon Aurora.

> **Important**
>
> Amazon DevOps Guru ist nicht für Aurora Serverless DB-Cluster verfügbar.

Das folgende Video gibt einen Überblick über DevOps Guru for RDS.

Weitere Informationen zu diesem Thema finden Sie unter Amazon DevOps Guru for RDS unter der Haube.

Themen

- Vorteile von DevOps Guru für RDS
- Wie funktioniert DevOps Guru for RDS
- DevOpsGuru für RDS einrichten

Vorteile von DevOps Guru für RDS

Wenn Sie für eine Datenbank von Amazon Aurora verantwortlich sind, wissen Sie möglicherweise nicht, dass ein Ereignis oder eine Regression auftritt, die sich auf diese Datenbank auswirkt. Wenn Sie von dem Problem erfahren, wissen Sie möglicherweise nicht, warum es auftritt und was Sie dagegen tun können. Anstatt sich an einen Datenbankadministrator (DBA) zu wenden, um Hilfe zu erhalten oder sich auf Tools von Drittanbietern zu verlassen, können Sie den Empfehlungen von DevOps Guru for RDS folgen.

Die detaillierte Analyse von DevOps Guru for RDS bietet Ihnen die folgenden Vorteile:

Schnelle Diagnose

DevOpsGuru for RDS überwacht und analysiert kontinuierlich die Datenbanktelemetrie. Performance Insights, Enhanced Monitoring und Amazon CloudWatch sammeln Telemetriedaten für Ihre . DevOpsGuru for RDS verwendet statistische Techniken und Techniken des maschinellen Lernens, um diese Daten zu analysieren und Anomalien zu erkennen. Weitere Informationen zu Telemetriedaten finden Sie unter Überwachung mit Performance Insights auf Amazon Aurora und Überwachen von Betriebssystem-Metriken mit „Enhanced Monitoring“ (Erweiterte Überwachung) im Amazon-Aurora-Benutzerhandbuch .

Schnelle Auflösung

Jede Anomalie identifiziert das Leistungsproblem und schlägt Möglichkeiten für Untersuchungen oder Korrekturmaßnahmen vor. DevOpsGuru for RDS könnte Ihnen beispielsweise empfehlen, bestimmte Warteereignisse zu untersuchen. Oder es empfiehlt sich, Ihre Anwendungspooleinstellungen zu optimieren, um die Anzahl der Datenbankverbindungen zu begrenzen. Basierend auf diesen Empfehlungen können Sie Leistungsprobleme schneller beheben als durch eine manuelle Fehlerbehebung.

Proaktive Einblicke

DevOpsGuru for RDS verwendet Metriken aus Ihren Ressourcen, um potenziell problematisches Verhalten zu erkennen, bevor es zu einem größeren Problem wird. Es kann beispielsweise erkennen, wenn Ihre Datenbank immer mehr temporäre Tabellen auf der Festplatte verwendet, was sich negativ auf die Leistung auswirken könnte. DevOps Guru gibt Ihnen dann Empfehlungen, die Ihnen helfen, Probleme zu lösen, bevor sie zu größeren Problemen werden.

Fundierte Kenntnisse der Amazon-Ingenieure und Machine Learning

DevOpsGuru for RDS setzt auf maschinelles Lernen (ML) und fortschrittliche mathematische Formeln, um Leistungsprobleme zu erkennen und Engpässe zu beheben. Die

Datenbankingenieure von Amazon haben zur Entwicklung der Ergebnisse von DevOps Guru for RDS beigetragen, die auf viele Jahre der Verwaltung von Hunderttausenden von Datenbanken zurückzuführen sind. Durch die Nutzung dieses kollektiven Wissens kann DevOps Guru for RDS Ihnen bewährte Verfahren vermitteln.

Wie funktioniert DevOps Guru for RDS

DevOpsGuru for RDS sammelt Daten über Ihre Aurora von Amazon RDS Performance Insights. Die wichtigste Metrik ist. `DBLoad` DevOpsGuru for RDS nutzt die Performance Insights Insights-Metriken, analysiert sie mit maschinellem Lernen und veröffentlicht die Erkenntnisse im Dashboard.

Ein Insight ist eine Sammlung verwandter Anomalien, die von Guru entdeckt wurden. DevOps

In DevOps Guru for RDS ist eine Anomalie ein Muster, das von dem abweicht, was als normale Leistung für Ihre Amazon Aurora angesehen wird.

Proaktive Einblicke

Ein proaktiver Einblick informiert Sie über problematisches Verhalten, bevor es auftritt. Er enthält Anomalien mit Empfehlungen und zugehörigen Metriken, die Ihnen helfen, Auffälligkeiten in Ihren Datenbanken von Amazon Aurora anzugehen, bevor sie zu größeren Problemen werden. Diese Erkenntnisse werden im Guru-Dashboard veröffentlicht. DevOps

DevOpsGuru könnte beispielsweise feststellen, dass Ihre Aurora PostgreSQL-Datenbank viele temporäre Tabellen auf der Festplatte erstellt. Wenn dieser Trend nicht angegangen wird, kann er zu Leistungsproblemen führen. Jeder proaktive Einblick beinhaltet Empfehlungen für korrektives Verhalten und Links zu relevanten Themen in Optimierung von Aurora MySQL mit proaktiven Einblicken von Amazon DevOps Guru oder Optimierung von Aurora PostgreSQL mit proaktiven Einblicken von Amazon DevOps Guru. Weitere Informationen finden Sie unter Working with Insights in DevOps Guru im Amazon DevOps Guru-Benutzerhandbuch.

Reaktive Einblicke

Ein reaktiver Einblick identifiziert anomales Verhalten, sobald es auftritt. Wenn DevOps Guru for RDS Leistungsprobleme in Ihren Amazon Aurora feststellt, veröffentlicht es einen reaktiven Einblick im DevOps Guru-Dashboard. Weitere Informationen finden Sie unter Working with Insights in DevOps Guru im Amazon DevOps Guru-Benutzerhandbuch.

Kausale Anomalien

Eine kausale Anomalie ist eine Anomalie der obersten Ebene innerhalb eines Einblicks. Die Datenbanklast (DB-Last) ist die ursächliche Anomalie für DevOps Guru for RDS.

Eine Anomalie misst die Leistungseinbußen durch Zuweisen eines Schweregrads von Hoch, Mittel oder Niedrig. Weitere Informationen finden Sie unter Wichtige Konzepte für DevOps Guru for RDS im Amazon DevOps Guru-Benutzerhandbuch.

Wenn DevOps Guru eine aktuelle Anomalie in Ihrer DB-Instance feststellt, werden Sie auf der Datenbankseite der RDS-Konsole benachrichtigt. Die Konsole warnt Sie auch bei Anomalien, die in den letzten 24 Stunden aufgetreten sind. Um von der RDS-Konsole zur Anomalieseite zu gelangen, wählen Sie den Link in der Warnmeldung. Die RDS-Konsole warnt Sie auch auf der Seite für Ihren DB-Cluster von Amazon Aurora .

Kontextbezogene Anomalien

Eine kontextbezogene Anomalie ist ein Befund innerhalb der Datenbanklast (DB-Last), der zu einem reaktiven Einblick gehört. Jede kontextbezogene Anomalie beschreibt ein bestimmtes Leistungsproblem von Amazon Aurora, das untersucht werden muss. DevOpsGuru for RDS könnte Ihnen beispielsweise empfehlen, eine Erhöhung der CPU-Kapazität in Betracht zu ziehen oder Warteereignisse zu untersuchen, die zur DB-Auslastung beitragen.

> **Important**
>
> Wir empfehlen Ihnen alle Änderungen in einer Test-Instance zu prüfen, bevor Sie eine produktive Instance ändern. Auf diese Weise verstehen Sie die Auswirkungen der Änderung.

Weitere Informationen finden Sie unter Analysieren von Anomalien in Amazon RDS im Amazon DevOps Guru-Benutzerhandbuch.

DevOpsGuru für RDS einrichten

Führen Sie die folgenden Aufgaben aus, um DevOps Guru for Amazon RDS die Veröffentlichung von Erkenntnissen zu ermöglichen.

Themen

- Konfiguration von IAM-Zugriffsrichtlinien für Guru for RDS DevOps

- Aktivieren von Performance Insights für Ihre DB-Instances von Aurora
- DevOpsGuru einschalten und die Ressourcenabdeckung angeben

Konfiguration von IAM-Zugriffsrichtlinien für Guru for RDS DevOps

Um Benachrichtigungen von DevOps Guru in der RDS-Konsole anzuzeigen, muss Ihr AWS Identity and Access Management (IAM-) Benutzer oder Ihre Rolle über eine der folgenden Richtlinien verfügen:

- Die AWS verwaltete Richtlinie `AmazonDevOpsGuruConsoleFullAccess`
- Die AWS verwaltete Richtlinie `AmazonDevOpsGuruConsoleReadOnlyAccess` und eine der folgenden Richtlinien:
  - Die AWS verwaltete Richtlinie `AmazonRDSFullAccess`
  - Eine vom Kunden verwaltete Richtlinie, die `pi:GetResourceMetrics` und `pi:DescribeDimensionKeys` einschließt

Weitere Informationen finden Sie unter Konfigurieren von Zugriffsrichtlinien für Performance Insights.

Aktivieren von Performance Insights für Ihre DB-Instances von Aurora

DevOpsGuru for RDS verlässt sich bei seinen Daten auf Performance Insights. Ohne Performance Insights veröffentlicht DevOps Guru Anomalien, beinhaltet aber keine detaillierten Analysen und Empfehlungen.

Wenn Sie einen Aurora-DB-Cluster erstellen oder eine Cluster-Instance ändern, können Sie Performance Insights aktivieren. Weitere Informationen finden Sie unter Aktivieren und Deaktivieren von Performance Insights für Aurora.

DevOpsGuru einschalten und die Ressourcenabdeckung angeben

Sie können DevOps Guru aktivieren, damit es Ihre Amazon Aurora auf eine der folgenden Arten überwacht.

Themen

- DevOpsGuru in der RDS-Konsole einschalten
- Hinzufügen von Ressourcen für Aurora in der Guru-Konsole DevOps
- Hinzufügen von Aurora mit CloudFormation

DevOpsGuru in der RDS-Konsole einschalten

Sie können in der Amazon RDS-Konsole mehrere Wege wählen, um DevOps Guru zu aktivieren.

Themen

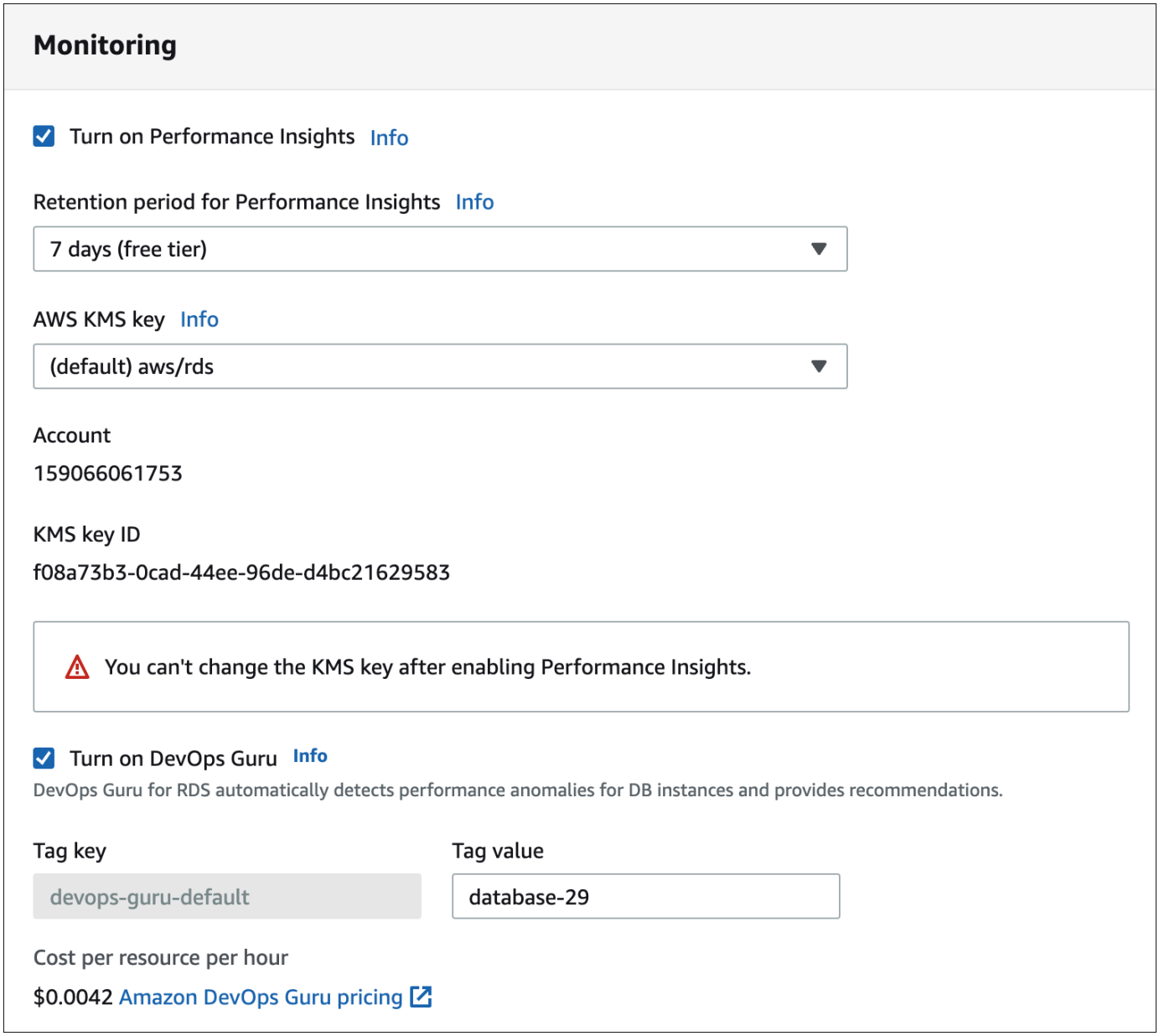
- DevOpsGuru einschalten, wenn Sie eine Aurora erstellen
- DevOpsGuru über das Benachrichtigungsbanner einschalten
- Du reagierst auf einen Berechtigungsfehler, wenn du DevOps Guru einschaltest

DevOpsGuru einschalten, wenn Sie eine Aurora erstellen

Der Erstellungs-Workflow umfasst eine Einstellung, mit der die DevOps Guru-Abdeckung für Ihre Datenbank aktiviert wird. Diese Einstellung ist standardmäßig aktiviert, wenn Sie die Vorlage Production (Produktion) auswählen.

Um DevOps Guru zu aktivieren, wenn Sie eine Aurora erstellen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Führen Sie die Schritte unter Erstellen eines DB-Clusters bis auf den Schritt aus, in dem Sie Überwachungseinstellungen wählen.
3. Wählen Sie unter Monitoring (Überwachung) die Option Turn on Performance Insights (Performance Insights aktivieren) aus. Damit DevOps Guru for RDS eine detaillierte Analyse von Leistungsanomalien bereitstellen kann, muss Performance Insights aktiviert sein.
4. Wählen Sie Turn on Guru. DevOps



5. Erstellen Sie ein Tag für Ihre Datenbank, damit DevOps Guru es überwachen kann. Gehen Sie wie folgt vor:

   - Geben Sie im Textfeld für Tag key (Tag-Schlüssel) einen Namen ein, der mit **`Devops-Guru-`** beginnt.
   - Geben Sie im Textfeld für Tag value (Tag-Wert) einen beliebigen Wert ein. Wenn Sie z. B. **`rds-database-1`** als Name Ihrer Datenbank von Aurora eingeben, können Sie auch **`rds-database-1`** als Tag-Wert eingeben.

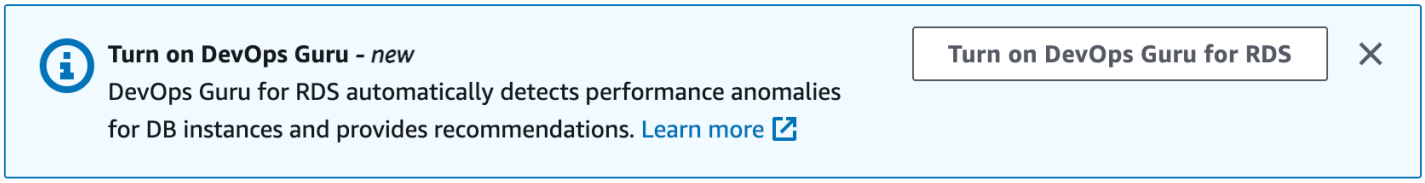
Weitere Informationen zu Tags finden Sie unter „Verwenden Sie Tags, um Ressourcen in Ihren DevOps Guru-Anwendungen zu identifizieren“ im Amazon DevOps Guru-Benutzerhandbuch.

6. Führen Sie die verbleibenden Schritte unter Erstellen eines DB-Clusters aus.

DevOpsGuru über das Benachrichtigungsbanner einschalten

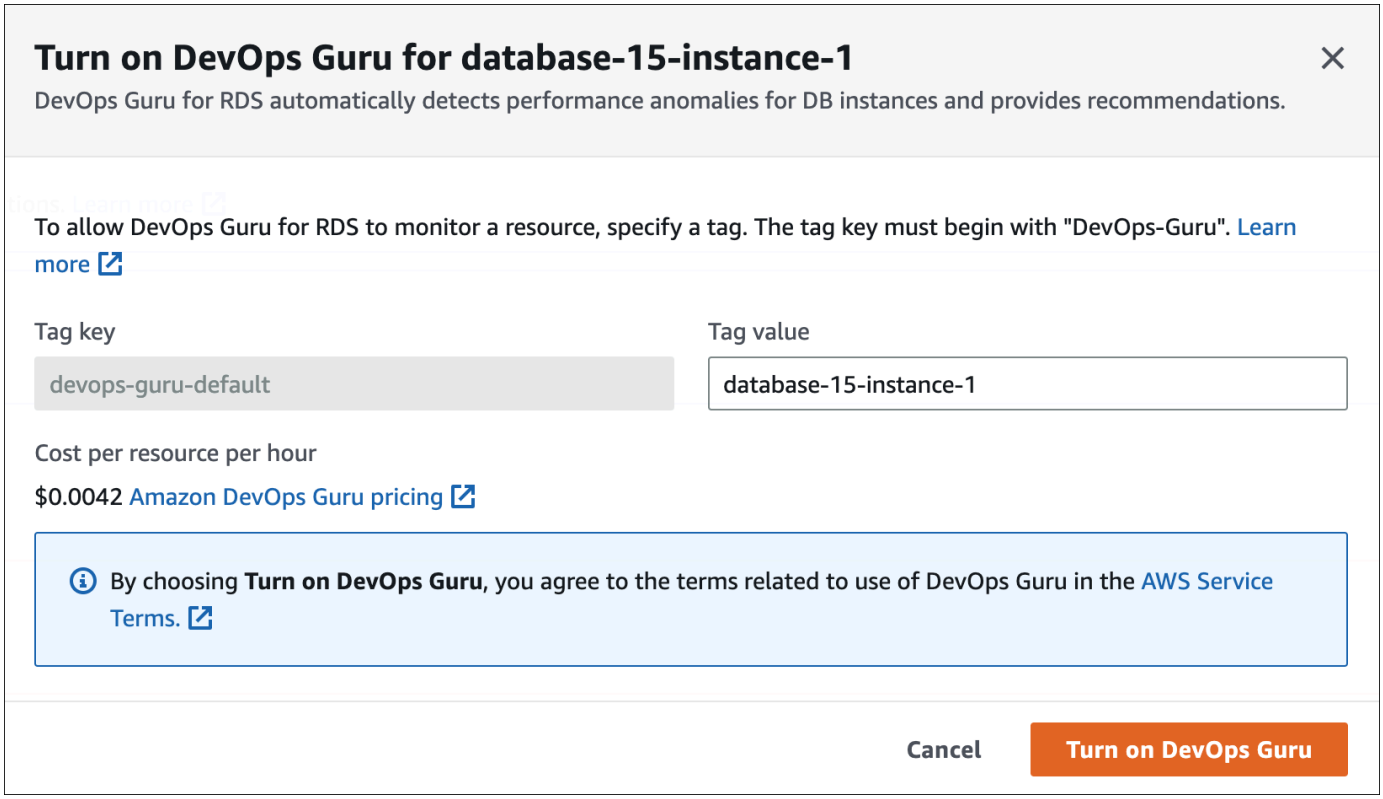
Wenn Ihre Ressourcen nicht von DevOps Guru abgedeckt werden, benachrichtigt Sie Amazon RDS mit einem Banner an den folgenden Stellen:

- Auf der Registerkarte Monitoring (Überwachung) einer DB-Cluster-Instance
- Im Performance-Insights-Dashboard



So aktivieren Sie DevOps Guru für Ihre Aurora

1. Wählen Sie im Banner Turn on DevOps Guru for RDS aus.
2. Geben Sie einen Schlüsselnamen und einen Wert für das Tag ein. Weitere Informationen zu Tags finden Sie unter „Verwenden Sie Tags, um Ressourcen in Ihren DevOps Guru-Anwendungen zu identifizieren“ im Amazon DevOps Guru-Benutzerhandbuch.



3. Wähle DevOpsGuru einschalten.

Du reagierst auf einen Berechtigungsfehler, wenn du DevOps Guru einschaltest

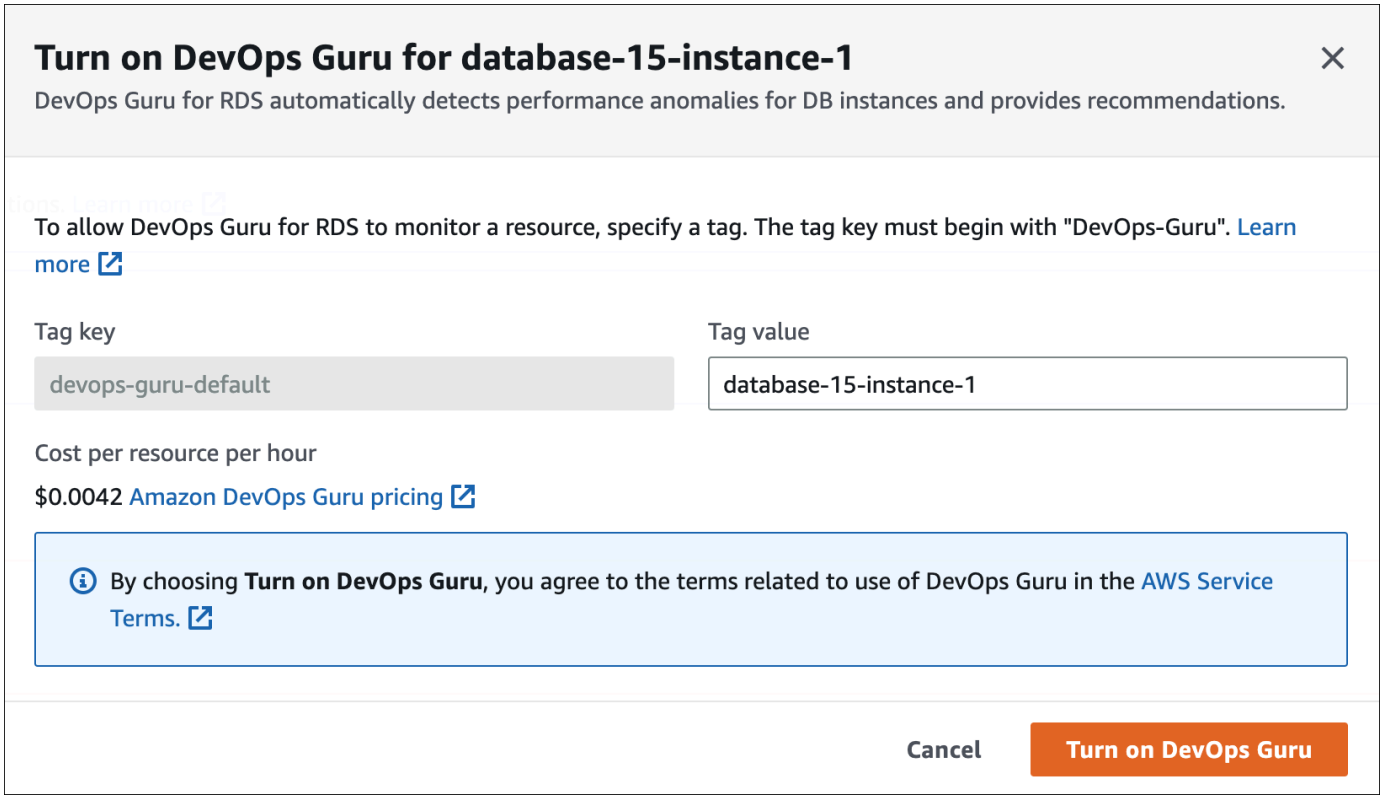
Wenn Sie DevOps Guru bei der Erstellung einer Datenbank von der RDS-Konsole aus aktivieren, zeigt RDS möglicherweise das folgende Banner mit fehlenden Berechtigungen an.

So reagieren Sie auf einen Berechtigungsfehler

1. Gewähren Sie Ihrem IAM-Benutzer oder Ihrer Rolle die vom Benutzer verwaltete Rolle `AmazonDevOpsGuruConsoleFullAccess`. Weitere Informationen finden Sie unter Konfiguration von IAM-Zugriffsrichtlinien für Guru for RDS DevOps.
2. Öffnen Sie die RDS-Konsole.
3. Wählen Sie im Navigationsbereich Performance-Insights aus.
4. Wählen Sie eine DB-Instance in dem Cluster aus, den Sie soeben erstellt haben.
5. Wählen Sie den Schalter, um DevOpsGuru für RDS einzuschalten.

DevOps Guru for RDS

6. Wählen Sie einen Tag-Wert aus. Weitere Informationen finden Sie unter „Verwenden Sie Tags, um Ressourcen in Ihren DevOps Guru-Anwendungen zu identifizieren“ im Amazon DevOps Guru-Benutzerhandbuch.



7. Wählen Sie DevOpsGuru einschalten.

Hinzufügen von Ressourcen für Aurora in der Guru-Konsole DevOps

Sie können die Abdeckung Ihrer DevOps Guru-Ressourcen auf der DevOps Guru-Konsole angeben. Folgen Sie dem unter Spezifizieren Sie den Umfang Ihrer DevOps Guru-Ressourcen im Amazon DevOps Guru-Benutzerhandbuch beschriebenen Schritt. Wählen Sie eine der folgenden Optionen aus, wenn Sie Ihre analysierten Ressourcen bearbeiten:

- Wählen Sie Alle Kontoressourcen, um alle unterstützten Ressourcen, einschließlich der Aurora , in Ihrer Region AWS-Konto und Ihrer Region zu analysieren.
- Wählen Sie CloudFormation Stacks aus, um die Aurora zu analysieren, die sich in Stacks Ihrer Wahl befinden. Weitere Informationen finden Sie unter Verwenden von AWS CloudFormation Stacks zur Identifizierung von Ressourcen in Ihren DevOps Guru-Anwendungen im Amazon DevOps Guru-Benutzerhandbuch.

- Klicken Sie auf Tags, um die Datenbanken von Aurora zu analysieren, die Sie mit einem Tag versehen haben. Weitere Informationen finden Sie unter Verwenden von Tags zur Identifizierung von Ressourcen in Ihren DevOps Guru-Anwendungen im Amazon DevOps Guru-Benutzerhandbuch.

Weitere Informationen finden Sie unter Enable DevOps Guru im Amazon DevOps Guru-Benutzerhandbuch.

Hinzufügen von Aurora mit CloudFormation

Sie können Tags verwenden, um Ihren Vorlagen die Abdeckung Ihrer Aurora hinzuzufügen. CloudFormation Das folgende Verfahren setzt voraus, dass Sie eine CloudFormation Vorlage sowohl für Ihre Aurora Ihren DevOps Guru-Stack haben.

So geben Sie eine Aurora mithilfe eines Tags an CloudFormation

1. Definieren Sie in der CloudFormation Vorlage für Ihre DB-Instance ein Tag mit einem Schlüssel/Wert-Paar.

   Das folgende Beispiel weist `Devops-guru-cfn-default` einer Aurora-DB-Instance den Wert `my-aurora-db-instance1`.

   ```
   MyAuroraDBInstance1:
     Type: "AWS::RDS::DBInstance"
     Properties:
       DBClusterIdentifier: my-aurora-db-cluster
       DBInstanceIdentifier: my-aurora-db-instance1
       Tags:
         - Key: Devops-guru-cfn-default
           Value: devopsguru-my-aurora-db-instance1
   ```

2. Geben Sie in der CloudFormation Vorlage für Ihren DevOps Guru-Stack dasselbe Tag in Ihrem Ressourcensammlungsfilter an.

   Im folgenden Beispiel wird DevOps Guru so konfiguriert, dass die Ressource mit dem Tag-Wert `my-aurora-db-instance1` abgedeckt wird.

   ```
   DevOpsGuruResourceCollection:
     Type: AWS::DevOpsGuru::ResourceCollection
     Properties:
       ResourceCollectionFilter:
   ```

```
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "devopsguru-my-aurora-db-instance1"
```

Das folgende Beispiel deckt alle Ressourcen innerhalb der Anwendungsgrenze `Devops-guru-cfn-default` ab.

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "*"
```

Weitere Informationen finden Sie unter AWS::DevOpsGuru::ResourceCollectionund AWS: :RDS:: DBInstance im CloudFormation Benutzerhandbuch.

Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung)

Mit „Enhanced Monitoring“ (Erweiterte Überwachung) können Sie das Betriebssystem Ihrer DB-Instance in Echtzeit überwachen. Wenn Sie sehen möchten, wie verschiedene Prozesse oder Threads die CPU verwenden, sind Enhanced Monitoring-Metriken nützlich.

Themen

- Überblick über „Enhanced Monitoring“ (Erweiterte Überwachung)
- Einrichten und Aktivieren von „Enhanced Monitoring“ (Erweiterte Überwachung)
- Anzeigen von Betriebssystem-Metriken in der RDS-Konsole
- Anzeigen von Betriebssystemmetriken mit CloudWatch Logs

Überblick über „Enhanced Monitoring“ (Erweiterte Überwachung)

Amazon RDS stellt in Echtzeit Metriken für das Betriebssystem bereit, auf dem Ihre DB-Instance ausgeführt wird. Sie können alle Systemmetriken und Prozessinformationen für Ihre RDS-DB-Instances in der Konsole anzeigen. Sie können verwalten, welche Metriken Sie für jede Instance überwachen möchten, und das Dashboard entsprechend Ihren Anforderungen anpassen. Beschreibungen der Metriken für Enhanced Monitoring finden Sie unter Betriebssystemmetriken in „Erweiterte Überwachung“.

RDS liefert die Metriken von „Enhanced Monitoring“ (Erweiterte Überwachung) in Ihr Amazon CloudWatch Logs-Konto. Sie können Metrikfilter in CloudWatch aus CloudWatch Logs erstellen und die Diagramme im CloudWatch-Dashboard anzeigen. Außerdem können Sie die Enhanced Monitoring JSON-Ausgabe aus Amazon CloudWatch Logs in einem Überwachungssystem Ihrer Wahl verwenden. Weitere Informationen finden Sie unterEnhanced Monitoring (Erweiterte Überwachung) in den Amazon-RDS-FAQs.

Themen

- Unterschiede zwischen Metriken für CloudWatch und für „Enhanced Monitoring“ (Erweiterte Überwachung)
- Aufbewahrung von Enhanced Monitoring-Metriken
- Kosten für „Enhanced Monitoring“ (Erweiterte Überwachung)

Unterschiede zwischen Metriken für CloudWatch und für „Enhanced Monitoring“ (Erweiterte Überwachung)

Ein Hypervisor erstellt und führt virtuelle Maschinen (VMs) aus. Mithilfe eines Hypervisors kann eine Instance mehrere Gast-VMs unterstützen, indem sie Arbeitsspeicher und CPU virtuell gemeinsam nutzt. CloudWatch sammelt Metriken über CPU-Auslastung vom Hypervisor für eine DB-Instance. Im Gegensatz dazu sammelt „Enhanced Monitoring“ (Erweiterte Überwachung) seine Metriken von einem Agenten auf der DB-Instance.

Möglicherweise finden Sie Unterschiede zwischen den Messungen CloudWatch und „Enhanced Monitoring“ (Erweiterte Überwachung), da die Hypervisor-Schicht einen geringen Arbeitsaufwand leistet. Die Unterschiede können größer sein, wenn Ihre DB-Instances kleinere Instance-Klassen verwenden. In diesem Szenario werden wahrscheinlich mehr virtuelle Maschinen (VMs) von der Hypervisorschicht auf einer einzelnen physischen Instance verwaltet.

Beschreibungen der Metriken für Enhanced Monitoring finden Sie unter Betriebssystemmetriken in „Erweiterte Überwachung“. Weitere Informationen über CloudWatch-Metriken finden Sie im Amazon CloudWatch-Benutzerhandbuch.

Aufbewahrung von Enhanced Monitoring-Metriken

Standardmäßig werden Enhanced Monitoring-Metriken 30 Tage lang im gespeichert CloudWatch Logs. Dieser Aufbewahrungszeitraum unterscheidet sich von typischen CloudWatch-Metriken.

Um zu ändern, für wie lange die Metriken im CloudWatch Logs gespeichert werden, ändern Sie in the CloudWatch-Konsole den Aufbewahrungszeitraum für die Protokollgruppe `RDSOSMetrics`. Weitere Informationen finden Sie unter Ändern der Aufbewahrung von Protokolldaten in CloudWatch-Protokollen im Amazon CloudWatch Logs User Guide.

Kosten für „Enhanced Monitoring“ (Erweiterte Überwachung)

Enhanced Monitoring-Metriken werden in den CloudWatch-Protokollen statt in Cloudwatch-Metriken gespeichert. Die Kosten für „Enhanced Monitoring“ (Erweiterte Überwachung) hängen von folgenden Faktoren ab:

- Ihnen werden nur Kosten für „Enhanced Monitoring“ (Erweiterte Überwachung) berechnet, wenn Sie das Kontingent für Datenübertragungen und Speicher überschreiten, das Amazon CloudWatch Logs bereitstellt. Die Gebühren basieren auf den Datenübertragungs- und Speicherraten von CloudWatch Logs.

- Die Menge der für eine RDS-Instance übertragenen Informationen ist direkt proportional zur definierten Granularität für „Enhanced Monitoring“ (Erweiterte Überwachung). Ein kürzeres Überwachungsintervall führt zu häufigeren Berichten über Betriebssystem-Metriken und erhöht Ihre Überwachungskosten. Um Kosten zu verwalten, legen Sie unterschiedliche Granularitäten für verschiedene Instances in Ihren Konten fest.
- Nutzungskosten für „Enhanced Monitoring“ (Erweiterte Überwachung) werden auf jede DB-Instance angewendet, für die Enhanced Monitoring (Erweiterte Überwachung) aktiviert ist. Die Überwachung einer großen Zahl von DB-Instances ist kostspieliger als die Überwachung von nur wenigen Instances.
- DB-Instances, die eine rechenintensivere Workload unterstützen, müssen mehr Aktivitäten von Betriebssystemprozessen melden und bedeuten höhere Kosten für „Enhanced Monitoring“ (Erweiterte Überwachung).

Weitere Informationen zu Preisen finden Sie unter Amazon CloudWatch-Preise.

Einrichten und Aktivieren von „Enhanced Monitoring“ (Erweiterte Überwachung)

Um „Enhanced Monitoring“ (Erweiterte Überwachung) zu verwenden, müssen Sie eine IAM-Rolle erstellen und dann „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren.

Themen

- So erstellen Sie eine IAM-Rolle für „Enhanced Monitoring“ (Erweiterte Überwachung)
- Aktivieren und Deaktivieren von „Erweiterte Überwachung“
- Schutz vor dem Confused-Deputy-Problem

So erstellen Sie eine IAM-Rolle für „Enhanced Monitoring“ (Erweiterte Überwachung)

Enhanced Monitoring erfordert die Erlaubnis, in Ihrem Namen zu handeln, um Betriebssystem-Metrikinformationen an CloudWatch Logs zu senden. Sie gewähren Enhanced Monitoring-Berechtigungen mithilfe einer AWS Identity and Access Management (IAM-) Rolle. Sie können diese Rolle entweder erstellen, wenn Sie „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren oder vorher erstellen.

Themen

- Erstellen der IAM-Rolle, wenn Sie „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren

- Erstellen der IAM-Rolle, bevor Sie „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren

Erstellen der IAM-Rolle, wenn Sie „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren

Wenn Sie „Enhanced Monitoring“ (Erweiterte Überwachung) in der RDS-Konsole aktivieren, kann Amazon RDS die erforderliche IAM-Rolle für Sie erstellen. Der Name der Rolle lautet `rds-monitoring-role`. RDS verwendet diese Rolle für die angegebene DB-Instance, das Lesereplikat oder den Multi-AZ-DB-Cluster.

So erstellen Sie die IAM-Rolle beim Aktivieren von „Enhanced Monitoring“ (Erweiterte Überwachung)

1. Führen Sie die Schritte unter au Aktivieren und Deaktivieren von „Erweiterte Überwachung“.
2. Setzen Sie die Überwachungsrolle in dem Schritt, in dem Sie eine Rolle auswählen auf Standard.

Erstellen der IAM-Rolle, bevor Sie „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren

Sie können die erforderliche Rolle erstellen, bevor Sie „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren. Wenn Sie „Enhanced Monitoring“ (Erweiterte Überwachung) aktivieren, geben Sie den Namen Ihrer neuen Rolle an. Sie müssen diese erforderliche Rolle erstellen, wenn Sie „Enhanced Monitoring“ (Erweiterte Überwachung) mithilfe der AWS CLI oder RDS API aktivieren.

Der Benutzer, der „Enhanced Monitoring“ (Erweiterte Überwachung) aktiviert, muss über die `PassRole`-Berechtigung verfügen. Weitere Informationen finden Sie unter Beispiel 2 unter Erteilen von Benutzerberechtigungen zur Übergabe einer Rolle an einen AWS Dienst im IAM-Benutzerhandbuch.

So erstellen Sie eine IAM-Rolle für „Enhanced Monitoring“ (Erweiterte Überwachung) in Amazon RDS

1. Öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com
2. Wählen Sie im Navigationsbereich Rollen aus.
3. Wählen Sie Create role (Rolle erstellen) aus.
4. Wählen Sie die Registerkarte AWS -Service und RDS in der Liste der Services aus.
5. Wählen Sie RDS – Enhanced Monitoring (RDS – erweiterte Überwachung) und Next (Weiter) aus.
6. Vergewissern Sie sich, dass in den Berechtigungsrichtlinien Amazon angezeigt wird RDSEnhancedMonitoringRole, und wählen Sie dann Weiter aus.

7. Geben Sie unter Role name (Rollenname) einen Namen für Ihre Rolle ein. Geben Sie z. B. ei **`emaccess`**.

   Die vertrauenswürdige Entität für Ihre Rolle ist der AWS Service monitoring.rds.amazonaws.com.
8. Wählen Sie Rolle erstellen aus.

Aktivieren und Deaktivieren von „Erweiterte Überwachung“

Sie können Enhanced Monitoring mithilfe der, oder RDS-API verwalten. AWS-Managementkonsole AWS CLI Sie können für jeden DB-Cluster unterschiedliche Granularitäten für die Metriksammlung festlegen. Sie können „Enhanced Monitoring“ auch für einen vorhandenen DB-Cluster über die Konsole aktivieren.

Konsole

Sie können „Enhanced Monitoring“·(Erweiterte·Überwachung) aktivieren, wenn Sie eine DB- einen Cluster oder ein Lesereplikat erstellen, oder wenn Sie eine DB- einen Cluster ändern. Wenn Sie eine DB-Instanceoder einen DB-Cluster ändern, um „Erweiterte·Überwachung“ zu aktivieren, müssen Sie Ihre DB-Instance nicht neu starten, damit die Änderung wirksam wird.

Sie können „Enhanced Monitoring“ in der RDS-Konsole aktivieren, wenn Sie eine der folgenden Aktionen auf der Seite Datenbanken ausführen:

- Erstellen einer DB-eines Clusters – Wählen Sie Datenbank erstellen aus.
- Erstellen eines Lesereplikats – Wählen Sie Aktionen und dann Lesereplikat erstellen aus.
- Eine DB-Instance oder einen DB-Cluster ändern – Wählen Sie Ändern aus.

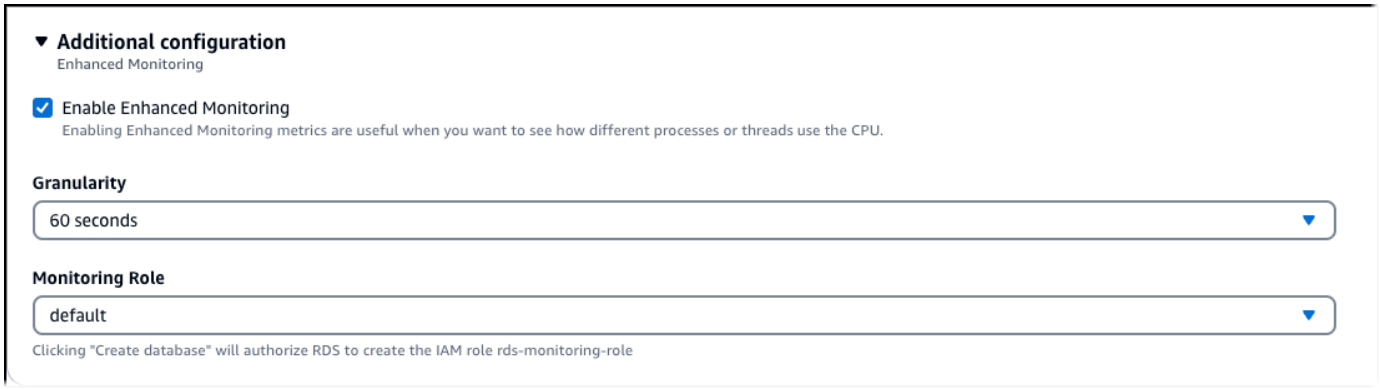
> **Note**
>
> Wenn Sie „Enhanced Monitoring“ für einen DB-Cluster aktivieren, können Sie „Enhanced Monitoring“ nicht für einzelne DB-Instances innerhalb des Clusters verwalten.

Wenn Sie „Erweiterte Überwachung“ für einzelne DB-Instances in einem DB-Cluster verwalten und die Granularität für verschiedene Instances auf unterschiedliche Werte eingestellt ist, ist Ihr DB-Cluster in Bezug auf „Erweiterte Überwachung“ heterogen. In solchen Fällen können Sie den DB-Cluster nicht ändern, um „Erweiterte Überwachung“ auf Cluster-Ebene zu verwalten.

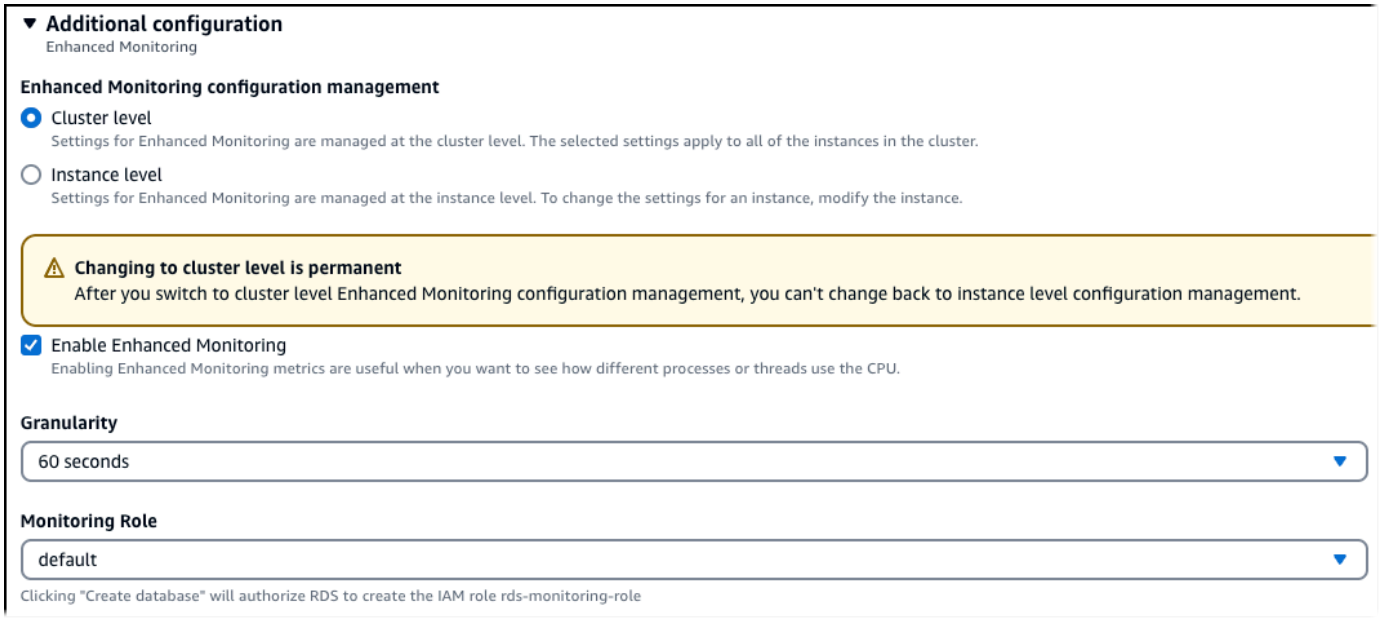
„Enhanced Monitoring“ (Erweiterte Überwachung) in der RDS-Konsole aktivieren/deaktivieren

1. Scrollen Sie zu Additional Configuration (Zusätzliche Konfiguration).
2. Wählen Sie unter Überwachung die Option Erweiterte Überwachung aktivieren für Ihre DB-Ihren DB-Cluster oder Ihr Lesereplikat aus. Wenn Sie „Erweiterte Überwachung“ auf Cluster-Ebene aktivieren, können Sie Einstellungen und Optionen für „Erweiterte Überwachung“ auf Cluster-Ebene verwalten. Einstellungen auf Cluster-Ebene gelten für alle DB-Instances des Clusters. Deaktivieren Sie die Option zur Deaktivierung von „Erweiterte Überwachung“ auf Cluster-Ebene. Sie können die Einstellungen für „Erweiterte Überwachung“ später für einzelne DB-Instances im Cluster ändern.

   Auf der Seite Datenbank erstellen können Sie auswählen, ob „Erweiterte Überwachung“ auf Cluster-Ebene aktiviert werden soll.



   Wenn Sie „Erweiterte Überwachung“ beim Erstellen eines Clusters nicht aktivieren, können Sie den Cluster auf der Seite DB-Cluster ändern ändern.



> **Note**
>
> Sie können „Erweiterte Überwachung“ nicht für eine einzelne DB-Instance in einem DB-Cluster verwalten, in dem „Erweiterte Überwachung“ bereits auf Cluster-Ebene verwaltet wird.

3. Sie können „Erweiterte Überwachung“ nicht auf Cluster-Ebene verwalten, wenn der Cluster in Bezug auf „Erweiterte Überwachung“ heterogen ist. Um „Erweiterte Überwachung“ auf Cluster-Ebene zu verwalten, ändern Sie die Einstellungen für „Erweiterte Überwachung“ für jede Instance so, dass sie übereinstimmen. Sie können jetzt festlegen, das „Erweiterte Überwachung“ auf Cluster-Ebene verwaltet werden soll, wenn Sie Ihren Cluster ändern.
4. Setzen Sie die Eigenschaft Monitoring Role auf die IAM-Rolle, die Sie erstellt haben, damit Amazon RDS für Sie mit Amazon CloudWatch Logs kommunizieren kann, oder wählen Sie Standard, damit RDS eine Rolle für Sie erstellt. `rds-monitoring-role`
5. Stellen Sie die Eigenschaft Granularität auf das Intervall (in Sekunden) zwischen Punkten ein, an denen Metriken für Ihre DB-Instance, Ihren DB-Cluster oder Ihr Lesereplikat erfasst werden. Die Eigenschaft Granularität kann auf einen der folgenden Werte eingestellt werden: `1`, `5`, `10`, `15`, `30` oder `60`.

   Die schnellste Aktualisierung der RDS-Konsole erfolgt alle 5 Sekunden. Wenn Sie die Granularität in der RDS-Konsole auf 1 Sekunde einstellen, sehen Sie die aktualisierten Metriken dennoch nur alle 5 Sekunden. Mithilfe von Logs können Sie Metrik-Updates innerhalb von einer CloudWatch Sekunde abrufen.

AWS CLI

Um Enhanced Monitoring mit den AWS CLI folgenden Befehlen zu aktivieren, setzen Sie die `--monitoring-interval` Option auf einen anderen Wert als `0` und setzen Sie die `--monitoring-role-arn` Option auf die Rolle, in So erstellen Sie eine IAM-Rolle für „Enhanced Monitoring“ (Erweiterte Überwachung) der Sie sie erstellt haben.

- create-db-cluster
- modify-db-cluster
- create-db-instance
- create-db-instance-read-replikat
- modify-db-instance

Die Option `--monitoring-interval` gibt das Intervall in Sekunden zwischen den Punkten an, an denen Enhanced Monitoring-Metriken erfasst werden. Gültige Werte für die Option sind `0`, `1`, `5`, `10`, `15`, `30` und `60`.

Um Enhanced Monitoring mit dem zu deaktivieren AWS CLI, setzen Sie die `--monitoring-interval` Option `0` in diesen Befehlen auf.

Example

Im folgenden Beispiel wird „Enhanced Monitoring“·(Erweiterte·Überwachung) für eine DB-Instance aktiviert:

Für Linux, macOS oder Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Für Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

Im folgenden Beispiel wird „Erweiterte Überwachung“·für einen DB-Cluster aktiviert:

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

Im folgenden Beispiel wird „Enhanced Monitoring“·(Erweiterte·Überwachung) für ein Multi-AZ-DB-Cluster aktiviert:

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

RDS-API

Um „Enhanced Monitoring“ (Erweiterte Überwachung) mithilfe der RDS API zu aktivieren, setzen Sie den Parameter `MonitoringInterval` auf einen anderen Wert als `0` und legen Sie den Parameter

`MonitoringRoleArn` auf die Rolle fest, die Sie in So erstellen Sie eine IAM-Rolle für „Enhanced Monitoring“ (Erweiterte Überwachung) erstellt haben. Legen Sie diese Parameter in den folgenden Aktionen fest:

- CreateDBCluster
- Modifizieren SieDBCluster
- CreateDBInstance
- CreateDBInstanceReadReplica
- Modifizieren SieDBInstance

Der Parameter `MonitoringInterval` gibt das Intervall in Sekunden zwischen den Punkten an, an denen Enhanced Monitoring-Metriken erfasst werden. Gültige Werte sind: `0`, `1`, `5`, `10`, `15`, `30` und `60`.

Um „Enhanced Monitoring“ (Erweiterte Überwachung) mit Hilfe der RDS API zu deaktivieren, setzen Sie `MonitoringInterval` auf `0`.

Schutz vor dem Confused-Deputy-Problem

Das Problem des verwirrten Stellvertreters ist ein Sicherheitsproblem, bei dem eine Entität, die keine Berechtigung zur Durchführung einer Aktion hat, eine privilegiertere Entität zur Durchführung der Aktion zwingen kann. In AWS kann ein dienstübergreifendes Identitätswechsels zum Problem des verwirrten Stellvertreters führen. Ein dienstübergreifender Identitätswechsel kann auftreten, wenn ein Dienst (der Anruf-Dienst) einen anderen Dienst anruft (den aufgerufenen Dienst). Der aufrufende Service kann manipuliert werden, um seine Berechtigungen zu verwenden, um Aktionen auf die Ressourcen eines anderen Kunden auszuführen, für die er sonst keine Zugriffsberechtigung haben sollte. Um dies zu verhindern, bietet AWS Tools, mit denen Sie Ihre Daten für alle Services mit Serviceprinzipalen schützen können, die Zugriff auf Ressourcen in Ihrem Konto erhalten haben. Weitere Informationen finden Sie unter Confused-Deputy-Problem.

Um die Berechtigungen einzuschränken, die Amazon RDS einem anderen Service für eine Ressource gewährt, empfehlen wir die globalen Bedingungskontextschlüssel `aws:SourceArn` und `aws:SourceAccount` in einer Vertrauensrichtlinie für Ihre Enhanced-Monitoring-Rolle. Wenn Sie beide globalen Bedingungskontextschlüssel verwenden, müssen diese dieselbe Konto-ID verwenden.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontextschlüssels `aws:SourceArn` mit dem vollständigen ARN der Ressource. Setzen Sie für Amazon RDS `aws:SourceArn` auf `arn:aws:rds:`*`Region`*`:`*`my-account-id`*`:db:`*`dbname`*.

Im folgenden Beispiel werden die globalen Bedingungskontextschlüssel `aws:SourceArn` und `aws:SourceAccount` in einer Vertrauensrichtlinie verwendet, um das Confused-Deputy-Problem zu verhindern.

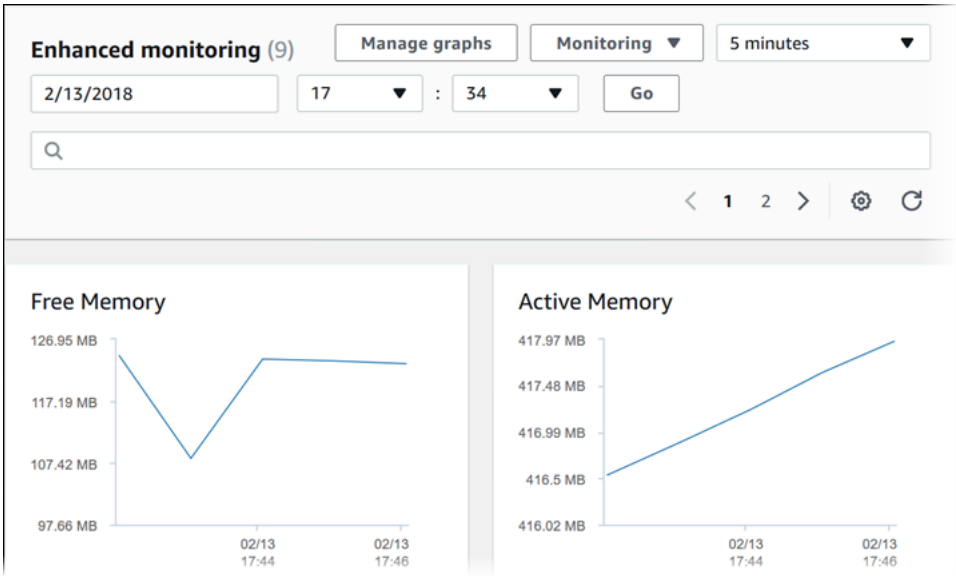
JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "monitoring.rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:rds:Region:my-account-id:db:dbname"
        },
        "StringEquals": {
          "aws:SourceAccount": "my-account-id"
        }
      }
    }
  ]
}
```

Anzeigen von Betriebssystem-Metriken in der RDS-Konsole

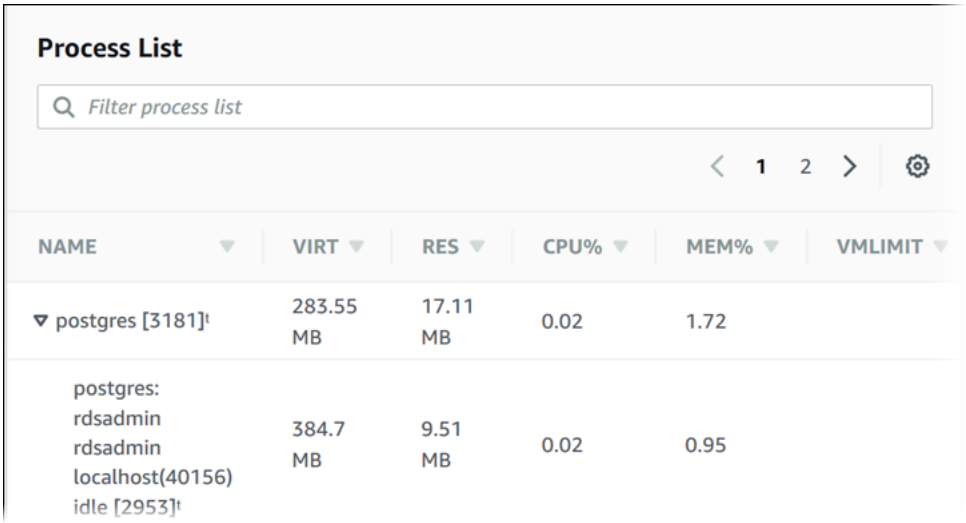
Sie können Betriebssystemmetriken anzeigen, die von Enhanced Monitoring in der RDS-Konsole gemeldet werden, indem Sie Enhanced monitoring (Erweiterte Überwachung) unter Monitoring (Überwachung) auswählen.

Das folgende Beispiel zeigt die Seite „Enhanced Monitoring“ an. Beschreibungen der Metriken für Enhanced Monitoring finden Sie unter Betriebssystemmetriken in „Erweiterte Überwachung“.



Wenn Sie Details für die auf Ihrer DB-Instance ausgeführten Prozesse ansehen möchten, wählen Sie die Betriebssystem-Prozessliste für Überwachung aus.

Die Ansicht Process List (Prozessliste) wird nachstehend angezeigt.



Die Enhanced Monitoring-Metriken, die in der Ansicht Process list (Prozessliste) gezeigt werden, sind wie folgt organisiert:

- RDS child processes (Untergeordnete RDS-Prozesse) – Zeigt eine Übersicht über die RDS-Prozesse, die die DB-Instance unterstützen, z. B. `aurora` für Amazon Aurora-DB-Cluster und . Prozess-Threads erscheinen unter dem übergeordneten Prozess. Prozess-Threads zeigen nur die CPU-Nutzung, da andere Metriken für alle Threads des Prozesses gleich sind. Die Konsole zeigt maximal 100 Prozesse und Threads. Das Ergebnis ist eine Kombination der CPU-

intensivsten und Memory-intensivsten Prozesse und Threads. Wenn über 50 Prozesse und über 50 Threads vorhanden sind, zeigt die Konsole für jede Kategorie die 50 mit dem höchsten Verbrauch. Anhand dieser Anzeige können Sie feststellen, welche Prozesse die größte Auswirkung auf die Performance haben.

- RDS-Prozesse — Zeigt eine Zusammenfassung der vom RDS-Management-Agenten verwendeten Ressourcen, der Diagnosen, der Überwachungsprozesse und anderer AWS Prozesse an, die zur Unterstützung von RDS-DB-Instances erforderlich sind.
- OS processes (Betriebssystemprozesse): Zeigt eine Übersicht über Kernel- und Systemprozesse, die im Allgemeinen einen geringen Einfluss auf die Performance haben.

Die aufgelisteten Elemente für jeden Prozess sind:

- VIRT: Zeigt die virtuelle Größe des Prozesses an.
- RES: Zeigt den tatsächlichen physischen Speicher an, der vom Prozess verwendet wird.
- CPU%: Zeigt den Prozentsatz der gesamten CPU-Bandbreite an, die vom Prozess verwendet wird.
- MEM%: Zeigt den Prozentsatz des Gesamtspeichers an, der vom Prozess verwendet wird.

Die in der RDS-Konsole angezeigten Überwachungsdaten werden aus Amazon CloudWatch Logs abgerufen. Sie können die Metriken für eine DB-Instance auch als Protokollstream aus CloudWatch Logs abrufen. Weitere Informationen finden Sie unter Anzeigen von Betriebssystemmetriken mit CloudWatch Logs.

Metriken für „Enhanced Monitoring“ (Erweiterte Überwachung) werden in folgenden Situationen nicht geliefert:

- Failover der DB-Instance.
- Ändern der Instance-Klasse für die DB-Instance (Skalierung der Datenverarbeitung).

Metriken für „Enhanced Monitoring“ (Erweiterte Überwachung) werden während eines Neustarts einer DB-Instance zurückgegeben, da nur die Datenbank-Engine neu gestartet wird. Metriken für das Betriebssystem werden weiterhin mitgeteilt.

Anzeigen von Betriebssystemmetriken mit CloudWatch Logs

Nach dem Aktivieren von „Enhanced Monitoring“·(Erweiterte·Überwachung) für Ihre DB- den Cluster können Sie die Metriken dafür mithilfe von CloudWatch Logs ansehen, wobei jeder Protokoll-Stream

eine einzelne überwachte DB-Instance oder ein DB-Cluster darstellt. Die Protokoll-Stream-Kennung ist die Ressourcenkennung (`DbiResourceId`) für die DB-Instance oder das DB-Cluster.

So zeigen Sie Enhanced Monitoring-Protokolldaten an

1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie bei Bedarf die AWS-Region, in der sich Ihre DB-Cluster befindet. Weitere Informationen finden Sie unter Regionen und Endpunkte in der Allgemeinen Amazon Web Services-Referenz.
3. Wählen Sie im Navigationsbereich Protokolle.
4. Wählen Sie RDSOSMetrics in der Liste der Protokollgruppen.
5. Wählen Sie aus der Liste den Protokoll-Stream, den Sie anzeigen möchten.

Amazon Aurora-Referenz für Metriken

In dieser Referenz finden Sie Beschreibungen von Amazon Aurora-Metriken für Amazon CloudWatch, Performance Insights und „Enhanced Monitoring“ (Erweiterte Überwachung).

Themen

- CloudWatch Amazon-Metriken für Amazon Aurora
- CloudWatch Amazon-Abmessungen für Aurora
- Verfügbarkeit von Aurora Metriken in der Amazon RDS Konsole
- Amazon-CloudWatch-Metriken für Erkenntnisse zur Amazon-RDS-Leistung
- Performance-Insights-Zählermetriken
- SQL-Statistiken für Performance Insights
- Betriebssystemmetriken in „Erweiterte Überwachung“

CloudWatch Amazon-Metriken für Amazon Aurora

Der `AWS/RDS`-Namespace umfasst die folgenden Metriken, die für Datenbankentitäten gelten, die in Amazon Aurora ausgeführt werden. Einige Metriken gelten entweder für Aurora MySQL, Aurora PostgreSQL oder beide. Darüber hinaus sind einige Metriken spezifisch für einen DB-Cluster, eine primäre DB-Instance, eine Replica-DB-Instance oder alle DB-Instances.

Informationen zu globalen Aurora-Datenbankmetriken finden Sie unter Amazon-CloudWatch-Metriken für die Schreibweiterleitung in Aurora MySQL und Amazon-CloudWatch-Metriken für die Schreibweiterleitung in Aurora PostgreSQL. Informationen zu parallelen Aurora-Abfragemetriken finden Sie unter Parallel Query für Aurora MySQL überwachen.

Themen

- Metriken auf Cluster-Ebene für Amazon Aurora
- Metriken auf Instance-Ebene für Amazon Aurora
- CloudWatch Amazon-Nutzungsmetriken für Amazon Aurora

Metriken auf Cluster-Ebene für Amazon Aurora

In der folgenden Tabelle werden Metriken beschrieben, die für Aurora-Cluster spezifisch sind.

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraGlobalDBDataTransferBytes`	In einer Aurora Global Database die Menge der Redo-Log-Daten, die von der AWS Quellregion in eine sekundäre AWS Region übertragen wurden. **Note** Diese Metrik ist nur in sekundären AWS-Regionen verfügbar.	Aurora MySQL und Aurora PostgreSQL	Bytes
`AuroraGlobalDBProgressLag`	In einer Aurora Global Database das Maß dafür, wie weit der sekundäre Cluster sowohl bei Benutzertransaktionen als auch bei Systemtransaktionen hinter dem primären Cluster zurückliegt. **Note** Diese Metrik ist nur in sekundären AWS-Regionen verfügbar.	Aurora MySQL und Aurora PostgreSQL	Millisekunden
`AuroraGlobalDBReplicatedWriteIO`	In einer Aurora Global Database die Anzahl der I/O Schreibvorgänge, die von der primären AWS Region auf das Cluster-Volume in einer sekundäre	Aurora MySQL und Aurora PostgreSQL	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
	n AWS Region repliziert wurden. Bei den Abrechnungsberechnungen für die sekundären AWS Regionen in einer globalen Datenbank werden Schreibvorgänge berücksichtigt`VolumeWriteIOPs` , die innerhalb des Clusters ausgeführt wurden. Bei den Abrechnungsberechnungen für die primäre AWS Region in einer globalen `VolumeWriteIOPs` Datenbank werden die Schreibaktivitäten innerhalb dieses Clusters und `AuroraGlobalDBReplicatedWriteIO` die regionsübergreifende Replikation innerhalb der globalen Datenbank berücksichtigt. **Note** Diese Metrik ist nur in sekundären AWS-Regionen verfügbar.		

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraGlobalDBReplicationLag`	In einer globalen Aurora-Datenbank, der Umfang der Verzögerung bei der Replikation von Updates aus der primären AWS -Region. **Note** Diese Metrik ist nur in sekundären AWS-Regionen verfügbar.	Aurora MySQL und Aurora PostgreSQL	Milliseku nden
`AuroraGlobalDBRPOLag`	In einer Aurora Global Database ist die Recovery Point Objective (RPO) Verzögerungszeit. Diese Metrik misst, wie weit der sekundäre Cluster bei Benutzertransaktionen hinter dem primären Cluster zurückliegt. **Note** Diese Metrik ist nur in sekundären AWS-Regionen verfügbar.	Aurora MySQL und Aurora PostgreSQL	Milliseku nden

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraVolumeBytesLeftTotal`	Der verbleibende verfügbare Speicherplatz für das Cluster-Volume. Während das Cluster-Volume wächst, verringert sich dieser Wert. Wenn sie Null erreicht, meldet der Cluster einen out-of-space Fehler. Wenn Sie feststellen möchten, ob sich Ihr Aurora-MySQL-Cluster dem Größenlimit von 128 Tebibyte (TiB) nähert, kann dieser Wert einfacher und zuverlässiger überwacht werden als `VolumeBytesUsed` . `AuroraVolumeBytesLeftTotal` berücksichtigt Speicherplatz, der für die interne Verwaltung und andere Zuordnungen verwendet wird, die sich nicht auf Ihre Speicherabrechnung auswirken.	Aurora MySQL	Byte
`BacktrackChangeRecordsCreationRate`	Die Anzahl der Datensätze zur Rückverfolgungsänderung, die über 5 Minuten für den DB-Cluster erstellt wurden.	Aurora MySQL	Anzahl pro 5 Minuten

Metrik	Beschreibung	Gilt für	Einheiten
`BacktrackChangeRecordsStored`	Die Anzahl der Datensätze zur Rückverfolgungsänderung, die von Ihrem DB-Cluster verwendet werden.	Aurora MySQL	Anzahl
`BackupRetentionPeriodStorageUsed`	Die Gesamtmenge des Backup-Speichers, der zur Unterstützung der point-in-time Wiederherstellungsfunktion innerhalb des Backup-Aufbewahrungsfensters des Aurora-DB-Clusters verwendet wird. Dieser Wert ist in dem von der `TotalBackupStorageBilled` - Metrik gemeldeten Gesamtwert enthalten. Er wird für jeden Aurora-Cluster separat berechnet. Detaillierte Anweisungen finden Sie unter Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora.	Aurora MySQL und Aurora PostgreSQL	Byte
`ServerlessDatabaseCapacity`	Die aktuelle Kapazität eines Aurora Serverless-DB-Clusters.	Aurora MySQL und Aurora PostgreSQL	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
`SnapshotStorageUsed`	Die gesamte Sicherungsspeicher, der von allen Aurora-Snapshots für einen Aurora-DB-Cluster außerhalb des Zeitfensters für die Aufbewahrung von Backups genutzt wird. Dieser Wert ist in dem von der `TotalBackupStorageBilled` - Metrik gemeldeten Gesamtwert enthalten. Er wird für jeden Aurora-Cluster separat berechnet. Detaillierte Anweisungen finden Sie unter Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora.	Aurora MySQL und Aurora PostgreSQL	Byte

Metrik	Beschreibung	Gilt für	Einheiten
`TotalBackupStorageBilled`	Die gesamte Sicherungsspeicher in Bytes, der Ihnen für einen bestimmten Aurora-DB-Cluster in Rechnung gestellt wird. Die Metrik umfasst den Sicherungsspeicher gemessen von den Metriken `BackupRetentionPeriodStorageUsed` und `SnapshotStorageUsed`. Diese Metrik wird für jeden Aurora-Cluster separat berechnet. Detaillierte Anweisungen finden Sie unter Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora.	Aurora MySQL und Aurora PostgreSQL	Bytes

Metrik	Beschreibung	Gilt für	Einheiten
`VolumeBytesUsed`	Die Menge des von Ihrem Aurora-DB-Cluster verwendeten Speicherp latzes. Dieser Wert wirkt sich auf die Kosten für den Aurora-DB-Cluster aus (Informat ionen zu Preisen erhalten Sie auf der Preisgest altungsseite zu Amazon RDS). Dieser Wert berücksichtigt einige interne Speicherz uweisungen nicht, die keine Auswirkungen auf die Speicherabrechnung haben. Bei Aurora MySQL können Sie out-of-space Probleme genauer antizipie ren, indem Sie testen, ob es `AuroraVolumeBytesL eftTotal` sich Null nähert, anstatt es mit dem Speicherlimit von 128 TiB zu `VolumeBytesUsed` vergleichen. Bei Clustern, bei denen es sich um Klone handelt, hängt der Wert dieser Metrik von der Menge der zum Klon hinzugefügten oder geänderten Daten ab.	Aurora MySQL und Aurora PostgreSQL	Bytes

Metrik	Beschreibung	Gilt für	Einheiten
	Die Metrik kann sich auch erhöhen oder verringern, wenn der ursprüngliche Cluster gelöscht wird oder wenn neue Klone hinzugefügt oder gelöscht werden. Details hierzu finden Sie unter Löschen eines Quell-Cluster-Volumes Es ist sinnlos, einen kleinen `--period`-Wert zu wählen, da die Metriken von Amazon RDS in Intervallen und nicht kontinuierlich erfasst werden.		

Metrik	Beschreibung	Gilt für	Einheiten
`VolumeReadIOPs`	Die Anzahl der in Rechnung gestellten I/O Lesevorgänge von einem Cluster-Volume innerhalb eines 5-Minuten-Intervalls. In Rechnung gestellte Leseoperationen werden auf Cluster-Volume-Ebene berechnet, aus allen Instances im Aurora-DB-Cluster zusammengestellt und dann in 5-minütigen Intervallen gemeldet. Der Wert wird anhand des Werts der Metrik für Read operations (Leseoperationen) über einen fünfminütigen Zeitraum berechnet. Sie können die Menge an in Rechnung gestellten Leseoperationen pro Sekunde bestimmen, indem Sie den Wert der Metrik Billed read operations (In Rechnung gestellte Operationen) durch 300 Sekunden teilen. Wenn beispielsweise die Metrik Billed read operations (In Rechnung gestellte Leseoperationen) den Wert 13 686 zurückgibt, wurden pro Sekunde 45 Leseopera	Aurora MySQL und Aurora PostgreSQL	Anzahl pro 5 Minuten

Metrik	Beschreibung	Gilt für	Einheiten
	tionen in Rechnung gestellt (13 686 / 300 = 45,62). In Rechnung gestellte Leseoperationen fallen für Abfragen für nicht im Buffer-Cache enthaltene Datenbankseiten an, die erst aus dem Speicher geladen werden müssen. Sie sehen evtl. Spitzenwerte in den in Rechnung gestellten Operationen, da Abfrageergebnisse aus dem Speicher gelesen und anschließend in den Buffer-Cache geladen werden. Es ist sinnlos, einen kleinen `--period`-Wert zu wählen, da die Metriken von Amazon RDS in Intervallen und nicht kontinuierlich erfasst werden. **Tip** Wenn Ihr Aurora MySQL Cluster Parallel Query verwendet, erfolgt möglicherweise eine Zunahme der `VolumeReadIOPS` -Werte. Parallel Query		

Metrik	Beschreibung	Gilt für	Einheiten
	verwendet den Bufferpool nicht. Obwohl die Abfragen schnell sind, kann diese optimierte Verarbeitung zu einem Anstieg der Lesevorgänge und damit verbundenen Gebühren führen.		
`VolumeWriteIOPs`	Die Anzahl der I/O Festplatt enschreibvorgänge auf das Cluster-Volume, die in Intervallen von 5 Minuten gemeldet werden. Eine ausführliche Beschreibung, wie in Rechnung gestellte Schreibvorgänge berechnet werden, finden Sie unter `VolumeReadIOPs` . Es ist sinnlos, einen kleinen `--period`-Wert zu wählen, da die Metriken von Amazon RDS in Intervallen und nicht kontinuierlich erfasst werden.	Aurora MySQL und Aurora PostgreSQL	Anzahl pro 5 Minuten

Metriken auf Instance-Ebene für Amazon Aurora

Die folgenden instanzspezifischen CloudWatch Amazon-Metriken gelten für alle Aurora MySQL- und Aurora PostgreSQL-Instances, sofern nicht anders angegeben.

Metrik	Beschreibung	Gilt für	Einheiten
`AbortedClients`	Die Anzahl der Clientverbindungen, die nicht ordnungsgemäß geschlossen wurden.	Aurora MySQL	Anzahl
`ActiveTransactions`	Die durchschnittliche Anzahl von aktuellen Transaktionen, die in einer Aurora-Datenbank-Instance pro Sekunde ausgeführt werden. Diese Metrik wird von Aurora standardmäßig nicht aktiviert. Um mit der Messung dieses Wertes zu beginnen, setzen Sie `innodb_monitor_enable='all'` in der DB-Parametergruppe für eine bestimmte DB-Instance.	Aurora MySQL	Anzahl pro Sekunde
`ACUUtilization`	Der Wert der `ServerlessDatabaseCapacity` -Metrik geteilt durch den maximalen ACU-Wert des DB-Clusters. Diese Metrik gilt nur für Aurora Serverless v2.	Aurora MySQL und Aurora PostgreSQL	Prozentsatz
`AuroraBinlogReplicaLag`	Wie lange ein DB-Cluster für die binäre Protokoll-Replikation, der in Aurora MySQL ausgeführt wird, hinter der binären Protokollreplikationsquelle zurückbleibt. Eine Verzögerung bedeutet,	Primär für Aurora MySQL	Sekunden

Metrik	Beschreibung	Gilt für	Einheiten
	dass die Quelle Datensätze schneller generiert, als das Replikat sie anwenden kann. Diese Metrik gibt je nach Engine-Version unterschi edliche Werte an: Aurora-MySQL-Version 2 Das `Seconds_B ehind_Master` - Feld des `SHOW SLAVE STATUS` von MySQL Aurora-MySQL-Version 3 `SHOW REPLICA STATUS` Sie können diese Metrik verwenden, um Fehler und Replikatverzögerung in einem Cluster zu überwachen, der als binäres Protokollreplikat fungiert. Der Metrikwert gibt Folgendes an: Einen hohen Wert Das Replikat verzögert die Replikationsquelle. **`0`** oder einen Wert nahe **`0`** Der Replikatprozess ist aktiv und aktuell.		

Metrik	Beschreibung	Gilt für	Einheiten
	-1 Aurora kann die Verzögerung nicht ermitteln, die während der Replikateinrichtung auftreten kann oder wenn sich das Replikat in einem Fehlerzustand befindet. Da die Binärprotokollrepl ikation nur auf der Writer-In stance des Clusters stattfind et, empfehlen wir, die Version dieser Metrik zu verwenden , die mit der WRITER-Rolle verknüpft ist. Weitere Hinweise zum Verwalten der Replikation finden Sie unter Replizier en von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg. Weitere Informationen zur Fehlerbehebung finden Sie unter Replikationsprobleme mit Amazon Aurora MySQL.		
`AuroraDMLRejectedM asterFull`	Die Anzahl der weitergel eiteten Abfragen, die abgelehnt wurden, weil die Sitzung auf der Writer-DB-Instance ausgelastet ist	Primär für Aurora-My SQL-Versi on 2	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraDMLRejectedWriterFull`	Die Anzahl der weitergeleiteten Abfragen, die abgelehnt wurden, weil die Sitzung auf der Writer-DB-Instance ausgelastet ist	Primär für Aurora-MySQL-Version 3	Anzahl
`AuroraEstimatedSharedMemoryBytes`	Die geschätzte Menge an freigegebenem Puffer oder Pufferpoolspeicher, die während des letzten konfigurierten Abfrageintervalls aktiv genutzt wurde.	Aurora PostgreSQL	Bytes
`AuroraMemoryHealthState`	Zeigt den Zustand des Speichers an. Der Wert `0` entspricht `NORMAL`. Der Wert `10` entspricht `RESERVED`, was bedeutet, dass sich der Server einem kritischen Speicherauslastungsgrad nähert. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.	Aurora MySQL Version 3.06.1 und höher	Messinstrument
`AuroraMemoryNumDeclinedSqlTotal`	Die inkrementelle Anzahl von Anfragen ging im Rahmen der (OOM-) Vermeidung zurück. out-of-memory Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.	Aurora MySQL Version 3.06.1 und höher	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraMemoryNumKilConnTotal`	Die inkrementelle Anzahl von Verbindungen, die im Rahmen der Vermeidung von Speichermangel geschlossen wurden. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.	Aurora MySQL Version 3.06.1 und höher	Anzahl
`AuroraMemoryNumKilQueryTotal`	Die inkrementelle Anzahl von Anfragen, die im Rahmen der Vermeidung von Speicherm angel beendet wurden. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.	Aurora MySQL Version 3.06.1 und höher	Anzahl
`AuroraMilliseconds SpentInOomRecovery`	Der Zeitraum, seitdem der Speicherzustand unter den Normalzustand gefallen ist. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.	Aurora MySQL Version 3.08.0 und höher	Milliseku nden

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraNumOomRecoveryTriggered`	Wie oft der Speicherzustand unter den Normalzustand gefallen ist. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.	Aurora MySQL Version 3.08.0 und höher	Anzahl
`AuroraNumOomRecoverySuccessful`	Gibt an, wie oft der Speicherzustand wieder in den Normalzustand versetzt wurde. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.	Aurora MySQL Version 3.08.0 und höher	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraOptimizedReadsCacheHitRatio`	Der Prozentsatz der vom Cache für optimierte Lesevorgänge verarbeiteten Anfragen. Der Wert wird mit der folgenden Formel berechnet: `orcache_blks_hit/(orcache_blks_hit + storage_blks_read)` Ein `AuroraOptimizedReadsCacheHitRatio` -Wert von 100 % bedeutet, dass alle Seiten aus dem Cache für optimierte Lesevorgänge gelesen wurden. Ein `AuroraOptimizedReadsCacheHitRatio` -Wert von `0` bedeutet, dass keine Seiten aus dem Cache für optimierte Lesevorgänge gelesen wurden.	Primärer Knoten für Aurora PostgreSQL	Prozentsatz
`AuroraReplicaLag`	Der Verzögerungszeitraum für eine Aurora-Replica, wenn Updates aus der primären Instance repliziert werden.	Replica für Aurora MySQL und Aurora PostgreSQL	Millisekunden

Metrik	Beschreibung	Gilt für	Einheiten
`AuroraReplicaLagMaximum`	Der maximale Verzögerungszeitraum zwischen der primären Instance und einer der Aurora-DB-Instances im DB-Cluster. Wenn Lesereplikate gelöscht oder umbenannt werden, kann es zu einem vorübergehenden Anstieg der Replikationsverzögerung kommen, da die alte Ressource einem Recycling unterzogen wird. Um eine genaue Darstellung der Replikationsverzögerung während dieses Zeitraums zu erhalten, empfehlen wir, die `AuroraReplicaLag` -Metrik für jede Lesereplikat-Instance zu überwachen.	Primärer Knoten für Aurora MySQL und Aurora PostgreSQL	Millisekunden
`AuroraReplicaLagMinimum`	Der minimale Verzögerungszeitraum zwischen der primären Instance und einer der Aurora-DB-Instances im DB-Cluster.	Primärer Knoten für Aurora MySQL und Aurora PostgreSQL	Millisekunden

Metrik	Beschreibung	Gilt für	Einheiten
AuroraSlowConnectionHandleCount	Die Anzahl der Verbindungen, die mindestens zwei Sekunden gewartet haben, um den Handshake zu starten. Diese Metrik gilt nur für Aurora MySQL Version 3.	Aurora MySQL	Anzahl
AuroraSlowHandshakeCount	Die Anzahl der Verbindungen, die mindestens 50 Millisekunden gewartet haben, um den Handshake zu beenden. Diese Metrik gilt nur für Aurora MySQL Version 3.	Aurora MySQL	Anzahl
BacktrackWindowActual	Der Unterschied zwischen dem Ziel-Backtrack-Fenster und dem tatsächlichen Backtrack-Fenster.	Primärer Knoten für Aurora MySQL	Minuten
BacktrackWindowAlert	Der Multiplikator, zu dem das tatsächliche Rückverfolgungsfenster für einen bestimmten Zeitraum kleiner als das Ziel-Rückverfolgungsfenster ist.	Primärer Knoten für Aurora MySQL	Anzahl
BlockedTransactions	Die durchschnittliche Anzahl an gesperrten Transaktionen in der Datenbank pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`BufferCacheHitRatio`	Der Prozentsatz der vom Buffer-Cache bedienten Anfragen.	Aurora MySQL und Aurora PostgreSQL	Prozentsatz
`CommitLatency`	Die durchschnittliche Dauer, die Engine und Speicher benötigen, um die Commit-Operationen abzuschließen.	Aurora MySQL und Aurora PostgreSQL	Milliseku nden
`CommitThroughput`	Durchschnittliche Anzahl der Commit-Operationen pro Sekunde.	Aurora MySQL und Aurora PostgreSQL	Anzahl pro Sekunde
`ConnectionAttempts`	Die Anzahl der Versuche, eine Verbindung mit einer Instance herzustellen, unabhängig davon, ob erfolgreich oder nicht.	Aurora MySQL	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
`CPUCreditBalance`	Die Anzahl der in 5-Minuten-Intervallen gemeldeten CPU-Guthaben, die eine Instance angesammelt hat. Sie können diese Metrik verwenden, um zu bestimmen, wie lange eine DB-Instance das normale Leistungslevel bei gegebener Leistungsrate übersteigen kann. Diese Metrik gilt nur für die folgenden Instance-Klassen: • Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3` und `db.t4g` • Aurora PostgreSQL: `db.t3` und `db.t4g` **Note** Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-	Aurora MySQL und Aurora PostgreSQL	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
	Instance-Klasse narten. Startguthaben funktioni eren in Amazon RDS genauso wie in Amazon EC2. Weitere Informationen finden Sie unter Launch credits (Startguthaben) im Amazon Elastic Compute Cloud-Ben utzerhandbuch für Linux-Ins tances.		

Metrik	Beschreibung	Gilt für	Einheiten
`CPUCreditUsage`	Die Anzahl der während des angegebenen Zeitraums verwendeten CPU-Guthaben (Meldung in 5-Minuten-Intervallen). Diese Metrik misst die Zeitspanne, in der physische Daten für die Verarbeitung von Anweisungen von virtuellen Geräten verwendet CPUs wurden, die der DB-Instance CPUs zugewiesen wurden. Diese Metrik gilt nur für die folgenden Instance-Klassen: • Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3` und `db.t4g` • Aurora PostgreSQL: `db.t3` und `db.t4g` **Note** Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-	Aurora MySQL und Aurora PostgreSQL	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
	Instance-Klassenarten.		
`CPUSurplusCreditBalance`	Die Anzahl überzähliger Guthaben, die von einer Unlimited-Instance verbraucht wurden, wenn ihr `CPUCreditBalance` -Wert null ist. Der `CPUSurplusCreditBalance` -Wert wird durch erworbene CPU-Guthaben abgezahlt. Wenn die Anzahl überzähliger Guthaben die Höchstzahl der Guthaben überschreitet, die die Instance in einem 24-Stunden-Zeitraum verdienen kann, fallen für die verbrauchten überzähligen Guthaben zusätzliche Gebühren an. Die Metriken für CPU-Guthaben sind nur mit einer fünfminütigen Frequenz verfügbar.	Aurora MySQL und Aurora PostgreSQL	Guthaben (vCPU-Minuten)

Metrik	Beschreibung	Gilt für	Einheiten
`CPUSurplusCreditsCharged`	Die Anzahl verbrauchter überzähliger Guthaben, die nicht durch verdiente CPU-Guthaben zurückgezahlt wurden, und für die deshalb eine zusätzliche Gebühr anfällt. Verbrauchte überzähliger Guthaben werden in Rechnung gestellt, wenn einer der folgenden Fälle auftritt: • Die ausgegebenen überzähligen Guthaben überschreiten die maximale Anzahl an Guthaben, die die Instance in einem 24-Stunden-Zeitraum verdienen kann. Über das Maximum hinaus ausgegebene überzählige Guthaben werden am Ende der Stunde abgerechnet. • Die Instance wird angehalten oder beendet. • Die Instance wird von `unlimited` in `standard` geändert. Die Metriken für CPU-Guthaben sind nur mit einer	Aurora MySQL und Aurora PostgreSQL	Guthaben (vCPU-Minuten)

Metrik	Beschreibung	Gilt für	Einheiten
	fünfminütigen Frequenz verfügbar.		
`CPUUtilization`	Prozentsatz des gegenwärtig von einer Aurora-DB-Instance benutzten CPU-Speichers.	Aurora MySQL und Aurora PostgreSQL	Prozentsatz
`DatabaseConnections`	Die Anzahl der Clientnet zwerkverbindungen zur Datenbank-Instance. Die Anzahl der Datenbank sitzungen kann höher als der Metrikwert sein, da der Metrikwert Folgendes nicht enthält: • Sitzungen, die keine Netzwerkverbindung mehr haben, die die Datenbank jedoch nicht bereinigt hat • Sitzungen, die von der Datenbank-Engine für eigene Zwecke erstellt wurden • Sitzungen, die durch die Funktionen zur parallelen Ausführung der Datenbank -Engine erstellt wurden • Vom Auftrags-Scheduler der Datenbank-Engine erstellte Sitzungen • Amazon-Aurora-Verb indungen	Aurora MySQL und Aurora PostgreSQL	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
DDLLatency	Die durchschnittliche Dauer von Anfragen wie beispielsweise „Erstellen“, „Ändern“ und „Verwerfen von Anfragen“.	Aurora MySQL	Milliseku nden
DDLThroughput	Die durchschnittliche Anzahl an DDL-Anfragen pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde
Deadlocks	Die durchschnittliche Anzahl an Deadlocks in der Datenbank pro Sekunde.	Aurora MySQL und Aurora PostgreSQL	Anzahl pro Sekunde
DeleteLatency	Die durchschnittliche Dauer von Löschvorgängen.	Aurora MySQL	Milliseku nden
DeleteThroughput	Durchschnittliche Anzahl Delete-Abfragen pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde
DiskQueueDepth	Die Anzahl der ausstehen den read/write Anfragen, die darauf warten, auf die Festplatte zuzugreifen.	Aurora MySQL und Aurora PostgreSQL	Anzahl
DMLLatency	Die durchschnittliche Dauer von Einfügungen, Aktualisi erungen und Löschungen.	Aurora MySQL	Milliseku nden
DMLThroughput	Durchschnittliche Anzahl von Einfügungen, Updates und Löschungen pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`EngineUptime`	Der Zeitraum, für den die Instance ausgeführt wurde.	Aurora MySQL und Aurora PostgreSQL	Sekunden
`FreeableMemory`	Verfügbarer Arbeitsspeicher. Für Aurora-MySQL- und Aurora-PostgreSQL-Datenbanken meldet diese Metrik den Wert des `MemAvailable` -Feldes als `/proc/meminfo` .	Aurora MySQL und Aurora PostgreSQL	Bytes
`FreeEphemeralStorage`	Die Menge des verfügbaren flüchtigen Speichers NVMe .	Aurora PostgreSQL	Bytes

Metrik	Beschreibung	Gilt für	Einheiten
`FreeLocalStorage`	Die Menge des verfügbaren lokalen Speichers. Im Gegensatz zu anderen DB-Engines gibt diese Metrik für Aurora DB-Instances die Menge des für jede DB-Instance verfügbaren Speichers an. Dieser Wert hängt von der DB-Instance-Klasse ab (Informationen zu den Preisen erhalten Sie auf der Preisgestaltungsseite zu Amazon RDS). Sie können die Menge an freiem Speicherplatz für eine Instance erhöhen, indem Sie eine größere DB-Instance-Klasse für Ihre Instance auswählen. (Dies gilt nicht für Aurora Serverless v2.)	Aurora MySQL und Aurora PostgreSQL	Bytes
`InsertLatency`	Die durchschnittliche Dauer von Einfügeoperationen.	Aurora MySQL	Millisekunden
`InsertThroughput`	Durchschnittliche Anzahl der Einfügeoperationen pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde
`LoginFailures`	Die durchschnittliche Anzahl fehlgeschlagener Anmeldeversuche pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`MaximumUsedTransactionIDs`	Das Alter der ältesten nicht vakuumierten Transaktions-ID in Transaktionen. Erreicht dieser Wert 2 146 483 648 (2^31 - 1 000 000), wird die Datenbank in den Nur-Lese-Modus gezwungen, um den Wraparound der Transaktions-ID zu vermeiden. Weitere Informationen finden Sie unter Transaktions-ID-Wraparound-Fehler vermeiden in der PostgreSQL-Dokumentation.	Aurora PostgreSQL	Anzahl
`NetworkReceiveThroughput`	Der Umfang des von Clients erhaltenen Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances im Aurora-DB-Cluster und dem Cluster-Volume.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde (Konsole zeigt Megabyte pro Sekunde an)
`NetworkThroughput`	Die Menge des von Clients empfangenen und gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances im Aurora-DB-Cluster und dem Cluster-Volume.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`NetworkTransmitThroughput`	Die Menge des von Clients gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances im DB-Cluster und dem Cluster-Volumen.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde (Konsole zeigt Megabyte pro Sekunde an)
`NumBinaryLogFiles`	Die Anzahl der generierten Binärprotokolldateien.	Aurora MySQL	Anzahl
`OldestReplicationSlotLag`	Der Verzögerungsgrößenwert des Replikats, das in Bezug auf die empfangenen Write-Ahead-Log-Daten (WAL) die höchste Verzögerung aufweist.	Aurora PostgreSQL	Bytes
`PurgeBoundary`	Transaktionsnummer, bis zu der eine InnoDB-Bereinigung zulässig ist. Wenn sich diese Metrik über einen längeren Zeitraum nicht erhöht, ist dies ein gutes Anzeichen dafür, dass die InnoDB-Bereinigung durch Transaktionen mit langer Laufzeit blockiert wird. Zur Untersuchung überprüfen Sie die aktiven Transaktionen auf Ihrem Aurora-MySQL-DB-Cluster.	Aurora MySQL Version 2, Version 2.11 und höher Aurora MySQL Version 3, Version 3.08 und höher	Anzahl

Metrik	Beschreibung	Gilt für	Einheiten
`PurgeFinishedPoint`	Transaktionsnummer, bis zu der eine InnoDB-Bereinigung ausgeführt wird. Diese Metrik kann Ihnen helfen, zu untersuchen, wie schnell die InnoDB-Bereinigung voranschreitet.	Aurora MySQL Version 2, Version 2.11 und höher Aurora MySQL Version 3, Version 3.08 und höher	Anzahl
`Queries`	Durchschnittliche Anzahl der ausgeführten Abfragen pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde
`RDSToAuroraPostgreSQLReplicaLag`	Die Verzögerung bei der Replikation von Updates aus einer primären RDS PostgreSQL-Instance in andere Knoten im Cluster.	Replica für Aurora PostgreSQL	Sekunden
`ReadIOPS`	Die durchschnittliche Anzahl von I/O Festplattenvorgängen pro Sekunde, wobei die Berichte jedoch getrennt in 1-Minuten-Intervallen gelesen und geschrieben werden.	Aurora MySQL und Aurora PostgreSQL	Anzahl pro Sekunde
`ReadIOPSEphemeralStorage`	Die durchschnittliche Anzahl von I/O Festplatten-Lesevorgängen in Ephemeral Storage NVMe .	Aurora PostgreSQL	Anzahl pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`ReadLatency`	Die durchschnittliche Zeit, die pro I/O Festplattenvorgang benötigt wird.	Aurora MySQL und Aurora PostgreSQL	Sekunden
`ReadLatencyEphemeralStorage`	Die durchschnittliche Zeit, die pro Festplatten-Lesevorgang für I/O temporären Speicher benötigt wird. NVMe	Aurora PostgreSQL	Sekunden
`ReadThroughput`	Die durchschnittliche Anzahl Byte, die pro Sekunde vom Datenträger gelesen werden.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde
`ReadThroughputEphemeralStorage`	Die durchschnittliche Anzahl der pro Sekunde von der Festplatte gelesenen Byte für NVMe Ephemeral Storage.	Aurora PostgreSQL	Bytes pro Sekunde
`ReplicationSlotDiskUsage`	Die Menge an Speicherplatz, die von Replikationsslotdateien belegt wird.	Aurora PostgreSQL	Bytes
`ResultSetCacheHitRatio`	Der Prozentsatz der vom Ergebnismengen-Cache bedienten Anfragen.	Aurora-MySQL-Version 2	Prozentsatz

Metrik	Beschreibung	Gilt für	Einheiten
`RollbackSegmentHistoryListLength`	Die Protokolle für das Rückgängigmachen, die festgeschriebene Transaktionen mit zum Löschen markierten Datensätzen aufzeichnen. Diese Datensätze sind für die Verarbeitung durch den InnoDB-Löschvorgang geplant.	Aurora MySQL	Anzahl
`RowLockTime`	Die Gesamtzeit für das Erfassen von Zeilensperren für InnoDB-Tabellen.	Aurora MySQL	Millisekunden
`SelectLatency`	Die durchschnittliche Zeit für ausgewählte Vorgänge.	Aurora MySQL	Millisekunden
`SelectThroughput`	Durchschnittliche Anzahl Select-Abfragen pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde
`ServerlessDatabaseCapacity`	Die aktuelle Kapazität eines Aurora Serverless-DB-Clusters.	Aurora MySQL und Aurora PostgreSQL	Anzahl
`StorageNetworkReceiveThroughput`	Der Umfang des vom Aurora-Speicheruntersystem erhaltenen Netzwerkdurchsatzes für jede Instance im DB-Cluster.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`StorageNetworkThroughput`	Der Umfang des vom Aurora-Speicheruntersystem erhaltenen und daran gesendeten Netzwerkd urchsatzes für jede Instance im Aurora-DB-Cluster.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde
`StorageNetworkTransmitThroughput`	Der Umfang des an das Aurora-Speicheruntersystem gesendeten Netzwerkd urchsatzes für jede Instance im Aurora-DB-Cluster.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde
`SumBinaryLogSize`	Die Gesamtgröße der Binärprotokolldateien.	Aurora MySQL	Byte
`SwapUsage`	Die Größe des verwendeten Auslagerungsbereichs. Diese Metrik ist für die folgenden DB-Instance-Klassen nicht verfügbar: • db.r3.*, db.r4.* und db.r7g.* (Aurora MySQL) • db.r7g.* (Aurora PostgreSQL)	Aurora MySQL und Aurora PostgreSQL	Bytes

Metrik	Beschreibung	Gilt für	Einheiten
`TempStorageIOPS`	Die Anzahl der IOPS sowohl für Lese- als auch Schreibvorgänge, die im lokalen Speicher durchgeführt werden, der der DB-Instance angefügt ist. Diese Metrik stellt eine Zählung dar und wird einmal pro Sekunde gemessen. Diese Metrik gilt nur für Aurora Serverless v2.	Aurora MySQL und Aurora PostgreSQL	Anzahl pro Sekunde
`TempStorageThroughput`	Die Menge der mit der DB-Instance verknüpften Daten, die in und aus dem lokalen Speicher übertragen wurden. Diese Metrik wird in Byte angegeben und einmal pro Sekunde gemessen. Diese Metrik gilt nur für Aurora Serverless v2.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde
`TransactionAgeMaximum`	Das Alter der ältesten aktiven ausgeführten Transaktion.	Aurora MySQL Version 3, Version 3.08 und höher	Sekunden

Metrik	Beschreibung	Gilt für	Einheiten
`TransactionLogsDiskUsage`	Der von Transaktionsprotokollen genutzte Festplattenplatz auf der DB-Instance von Aurora-PostgreSQL. Diese Metrik wird nur generiert, wenn Aurora PostgreSQL logische Replikation oder verwendet . AWS Database Migration Service Standardmäßig verwendet Aurora PostgreSQL Protokolldatensätze, nicht Transaktionsprotokolle. Wenn Transaktionsprotokolle nicht verwendet werden, lautet der Wert für diese Metrik `-1`.	Primärer Knoten für Aurora PostgreSQL	Bytes
`TruncateFinishedPoint`	Die Transaktions-ID, bis zu der die Kürzung rückgängig gemacht wird.	Aurora MySQL Version 2, Version 2.11 und höher Aurora MySQL Version 3, Version 3.08 und höher	Anzahl
`UpdateLatency`	Die durchschnittliche Zeit für Aktualisierungsvorgänge.	Aurora MySQL	Millisekunden
`UpdateThroughput`	Durchschnittliche Anzahl Updates pro Sekunde.	Aurora MySQL	Anzahl pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`WriteIOPS`	Die Anzahl der pro Sekunde generierten Aurora-Sp eicherdatensätze. Dies entspricht etwa der Anzahl der Protokolldatensätze, die von der Datenbank generiert werden. Diese entsprech en weder den 8K-Seiten -Schreibvorgängen noch den gesendeten Netzwerkp aketen.	Aurora MySQL und Aurora PostgreSQL	Anzahl pro Sekunde
`WriteIOPSEphemeral Storage`	Die durchschnittliche Anzahl von I/O Schreibvorgängen auf der Festplatte in den flüchtigen NVMe Speicher.	Aurora PostgreSQL	Anzahl pro Sekunde
`WriteLatency`	Die durchschnittliche Zeit, die pro I/O Festplattenvorgang benötigt wird.	Aurora MySQL und Aurora PostgreSQL	Sekunden
`WriteLatencyEpheme ralStorage`	Die durchschnittliche Zeit, die pro I/O Schreibvorgang auf der Festplatte für temporären Speicher benötigt wird. NVMe	Aurora PostgreSQL	Sekunden
`WriteThroughput`	Die durchschnittliche Anzahl von Bytes, die jede Sekunde in persistente Speicher geschrieben werden.	Aurora MySQL und Aurora PostgreSQL	Bytes pro Sekunde

Metrik	Beschreibung	Gilt für	Einheiten
`WriteThroughputEphemeralStorage`	Die durchschnittliche Anzahl der pro Sekunde auf die Festplatte geschriebenen Byte für kurzlebigen NVMe Speicher.	Aurora PostgreSQL	Bytes pro Sekunde

CloudWatch Amazon-Nutzungsmetriken für Amazon Aurora

Der `AWS/Usage` Namespace in Amazon CloudWatch enthält Nutzungsmetriken auf Kontoebene für Ihre Amazon RDS-Servicekontingenten. CloudWatch sammelt automatisch Nutzungsmetriken für alle. AWS-Regionen

Weitere Informationen finden Sie unter CloudWatch Nutzungsmetriken im CloudWatch Amazon-Benutzerhandbuch. Weitere Informationen zu Kontingenten finden Sie unter Kontingente und Beschränkungen für Amazon Aurora und Beantragen einer Kontingenterhöhung im Service-Quotas-Benutzerhandbuch.

Metrik	Description	Einheiten *
`AuthorizationsPerDBSecurityGroup`	Die Anzahl der Regeln für eingehenden Datenverkehr pro DB-Sicherheitsgruppe in Ihrem AWS-Konto. Der verwendete Wert ist die höchste Anzahl von Regeln für eingehenden Datenverkehr in einer DB-Sicherheitsgruppe im Konto. Andere DB-Sicherheitsgruppen im Konto haben möglicherweise eine geringere Anzahl von Regeln für eingehenden Datenverkehr.	Anzahl
`CustomEndpointsPerDBCluster`	Die Anzahl der benutzerdefinierten Endpunkte pro DB-Cluster in Ihrem AWS-Konto. Der verwendete Wert ist die höchste Anzahl von benutzerdefinierten Endpunkten in einem DB-Cluster im Konto. Andere DB-Cluster im Konto haben möglicherweise eine geringere Anzahl von benutzerdefinierten Endpunkten.	Anzahl

Metrik	Description	Einheiten *
DBCluster ParameterGroups	Die Anzahl der DB-Cluster-Parametergruppen in Ihrem AWS-Konto. In der Anzahl sind keine Standardparameterg ruppen enthalten.	Anzahl
DBClusterRoles	Die Anzahl der zugehörigen Rollen AWS Identity and Access Management (IAM) pro DB-Cluster in Ihrem AWS-Konto. Der verwendete Wert ist die höchste Anzahl zugeordneter IAM-Rollen für einen DB-Cluster im Konto. Andere DB-Cluster im Konto haben möglicherweise eine geringere Anzahl von zugeordneten IAM-Rollen.	Anzahl
DBClusters	Die Anzahl der Amazon-Aurora-DB-Cluster in Ihrem AWS-Konto.	Anzahl
DBInstanceRoles	Die Anzahl der zugehörigen Rollen AWS Identity and Access Management (IAM) pro DB-Instance in Ihrem. AWS-Konto Der verwendete Wert ist die höchste Anzahl zugeordneter IAM-Rollen für eine DB-Instance im Konto. Andere DB-Instances im Konto haben möglicherweise eine geringere Anzahl von zugeordneten IAM-Rollen.	Anzahl
DBInstances	Die Anzahl der DB-Instances in Ihrem AWS-Konto.	Anzahl
DBParamet erGroups	Die Anzahl der DB-Parametergruppen in Ihrem AWS-Konto . In der Anzahl sind keine Standard-DB-Parametergruppen enthalten.	Anzahl
DBSubnetGroups	Die Anzahl der DB-Subnetzgruppen in Ihrem AWS-Konto . In der Anzahl ist die Standardsubnetzgruppe nicht enthalten.	Anzahl
EventSubs criptions	Die Anzahl der Abonnements für Ereignisbenachrich tigungen in Ihrem AWS-Konto.	Anzahl

Metrik	Description	Einheiten *
`Integrations`	Die Anzahl der Null-ETL-Integrationen mit Amazon Redshift in Ihrem AWS-Konto.	Anzahl
`ManualClusterSnapshots`	Die Anzahl der manuell erstellten DB-Cluster-Snapshots in Ihrem AWS-Konto. In der Anzahl sind ungültige Snapshots nicht enthalten.	Anzahl
`OptionGroups`	Die Anzahl der Optionsgruppen in Ihrem AWS-Konto . In der Anzahl sind die Standardoptionsgruppen nicht enthalten.	Anzahl
`Proxies`	Die Anzahl der RDS-Proxys in Ihrem Konto. AWS	Anzahl
`ReadReplicasPerMaster`	Die Anzahl der Lesereplikate pro DB-Instance in Ihrem Konto. Der verwendete Wert ist die höchste Anzahl von Lesereplikaten für eine DB-Instance im Konto. Andere DB-Instances im Konto haben möglicherweise eine geringere Anzahl von Lesereplikaten.	Anzahl
`ReservedDBInstances`	Die Anzahl der reservierten DB-Instances in Ihrem AWS-Konto. In der Anzahl sind außer Betrieb genommene oder abgelehnte Instances nicht enthalten.	Anzahl
`SubnetsPerDBSubnetGroup`	Die Anzahl von Subnetzen pro DB-Subnetzgruppe in Ihrem AWS-Konto. Die höchste Anzahl von Subnetzen für eine DB-Subnetzgruppe im Konto. Andere DB-Subnetzgruppen im Konto haben möglicherweise eine geringere Anzahl von Subnetzen.	Anzahl

> **Note**
>
> Amazon RDS veröffentlicht keine Einheiten für Nutzungsmetriken CloudWatch. Die Einheiten werden nur in der Dokumentation angezeigt.

CloudWatch Amazon-Abmessungen für Aurora

Sie können Aurora-Metrikdaten filtern, indem Sie eine beliebige Dimension in der folgenden Tabelle verwenden.

Dimension	Filtert die angeforderten Daten für . . .
`DBInstanceIdentifier`	Eine angegebene DB-Instance.
`DBClusterIdentifier`	Ein angegebener Aurora-DB-Cluster.
`DBClusterIdentifier, Role`	Ein bestimmter Aurora-DB-Cluster, der die Metrik nach Instance-Rolle zusammenfasst (WRITER/READER). Sie können beispielsweise Metriken für alle READER-Instances eines Clusters zusammenfassen.
`DbClusterIdentifier, EngineName`	Eine spezifische Kombination von Aurora-DB-Cluster und Engine-Name. Sie können beispielsweise die `VolumeReadIOPs` -Metrik für den Cluster `ams1` und die Engine `aurora` anzeigen.
`DatabaseClass`	Alle Instances in einer Datenbankklasse. Sie können beispielsweise Metriken für alle Instances der Datenbankklasse zusammenfasse `db.r5.large` .
`EngineName`	Nur der identifizierte Engine-Name. Sie können beispielsweise Metriken für alle Instances mit dem Engine-Namen zusammenfasse `aurora-postgresql` .
`SourceRegion`	Nur die angegebene Region. Sie können beispielsweise Metriken für alle DB-Instances in der Region `us-east-1` zusammenfassen.

Verfügbarkeit von Aurora Metriken in der Amazon RDS Konsole

Nicht alle Metriken, die von Amazon Aurora bereitgestellt werden, sind in der Amazon-RDS-Konsole verfügbar. Sie können diese Metriken mittels Tools anzeigen, beispielsweise der AWS-CLI und der

CloudWatch-API. Zudem werden einige Metriken in der Amazon-RDS-Konsole entweder für eine bestimmte Instance-Klasse oder mit verschiedenen Namen und Messeinheiten angezeigt.

Themen

- Aurora-Metriken, die in der Ansicht „Last Hour“ (Letzte Stunde) verfügbar sind
- Aurora Metriken in bestimmten Fällen verfügbar
- Aurora-Metriken, die nicht in der Konsole verfügbar sind

Aurora-Metriken, die in der Ansicht „Last Hour“ (Letzte Stunde) verfügbar sind

Sie können eine Teilmenge von kategorisierten Aurora-Metriken in der Standard-Ansicht für die letzte Stunde in der Amazon-RDS-Konsole anzeigen. Die folgende Tabelle listet die Kategorien und zugehörigen Metriken auf, die in der Amazon-RDS-Konsole für eine Aurora-Instance angezeigt werden.

Kategorie	Metriken

SQL	`ActiveTransactions`
	`BlockedTransactions`
	`BufferCacheHitRatio`
	`CommitLatency`
	`CommitThroughput`
	`DatabaseConnections`
	`DDLLatency`
	`DDLThroughput`
	`Deadlocks`
	`DMLLatency`
	`DMLThroughput`
	`LoginFailures`

Kategorie	Metriken

	`ResultSetCacheHitRatio`
	`SelectLatency`
	`SelectThroughput`

System (System)	`AuroraReplicaLag`
	`AuroraReplicaLagMaximum`
	`AuroraReplicaLagMinimum`
	`CPUCreditBalance`
	`CPUCreditUsage`
	`CPUUtilization`
	`FreeableMemory`
	`FreeLocalStorage` (Dies gilt nicht für Aurora Serverless v2.)
	`NetworkReceiveThroughput`

Bereitstellung	`AuroraReplicaLag`
	`BufferCacheHitRatio`
	`ResultSetCacheHitRatio`
	`SelectThroughput`

Aurora Metriken in bestimmten Fällen verfügbar

Zusätzlich werden einige Aurora-Metriken entweder für eine bestimmte Instance-Klasse oder nur für DB-Instances mit verschiedenen Namen und verschiedenen Messeinheiten angezeigt.

- Die Metriken `CPUCreditBalance` und `CPUCreditUsage` werden nur für Aurora MySQL-`db.t2`-Instance-Klassen und für Aurora PostgreSQL-`db.t3`-Instance-Klassen angezeigt.

- Eine Auflistung der Metriken, die mit verschiedenen Namen angezeigt werden:

Metrik	Anzeigename
`AuroraReplicaLagMaximum`	Maximalverzögerung des Replicas
`AuroraReplicaLagMinimum`	Minimalverzögerung des Replicas
`DDLThroughput`	DDL
`NetworkReceiveThroughput`	Netzwerkdurchsatz
`VolumeBytesUsed`	[Fakturiertes] Volumen der verwendeten Bytes
`VolumeReadIOPs`	[Fakturiertes] Volumen gelesene IOPS
`VolumeWriteIOPs`	[Fakturiertes] Volumen geschriebene IOPS

- Die folgenden Metriken gelten für das gesamte Aurora-DB-Cluster, werden jedoch nur angezeigt, wenn DB-Instances für ein Aurora-DB-Cluster in der Amazon-RDS-Konsole angezeigt werden:
  - `VolumeBytesUsed`
  - `VolumeReadIOPs`
  - `VolumeWriteIOPs`
- Die folgenden Metriken werden in der Amazon-RDS-Konsole in Megabytes, anstatt in Bytes angezeigt:
  - `FreeableMemory`
  - `FreeLocalStorage`
  - `NetworkReceiveThroughput`
  - `NetworkTransmitThroughput`
- Die folgenden Metriken gelten für einen DB-Cluster von Aurora PostgreSQL mit Aurora-optimierten Lesevorgängen:
  - `AuroraOptimizedReadsCacheHitRatio`
  - `FreeEphemeralStorage`
  - `ReadIOPSEphemeralStorage`
  - `ReadLatencyEphemeralStorage`
  - `ReadThroughputEphemeralStorage`

- `WriteIOPSEphemeralStorage`
- `WriteLatencyEphemeralStorage`
- `WriteThroughputEphemeralStorage`

Aurora-Metriken, die nicht in der Konsole verfügbar sind

Die folgenden Aurora Metriken sind in der Amazon RDS Konsole nicht verfügbar:

- `AuroraBinlogReplicaLag`
- `DeleteLatency`
- `DeleteThroughput`
- `EngineUptime`
- `InsertLatency`
- `InsertThroughput`
- `NetworkThroughput`
- `Queries`
- `UpdateLatency`
- `UpdateThroughput`

Amazon-CloudWatch-Metriken für Erkenntnisse zur Amazon-RDS-Leistung

Performance Insights veröffentlicht automatisch Metriken für Amazon CloudWatch. Dieselben Daten können aus Performance Insights heraus abgefragt werden, aber mithilfe der Metriken in CloudWatch können CloudWatch-Alarme leicht hinzugefügt werden. Die Metriken können auch leicht zu vorhandenen CloudWatch-Dashboards hinzugefügt werden.

Metrik	Beschreibung
DBLoad	Anzahl der aktiven Sitzungen für die Datenbank . In der Regel sind Sie an den Daten für die durchschnittliche Anzahl der aktiven Sitzungen interessiert. Diese Daten werden in Performan ce Insights als `db.load.avg` abgefragt.

Metrik	Beschreibung
DBLoadCPU	Anzahl aktiver Sitzungen mit dem Warteereignistyp CPU Diese Daten werden in Performance Insights als `db.load.avg` abgefragt, gefiltert durch den Warteereignistyp `CPU`.
DBLoadNonCPU	Anzahl aktiver Sitzungen mit einem anderen Warteereignistyp als CPU
DBLoadRelativeToNumVCPUs	Das Verhältnis der DB-Last zur Anzahl der virtuellen CPUs für die Datenbank.

> **Note**
>
> Diese Metriken werden nur dann zu CloudWatch veröffentlicht, wenn es einen Workload auf der DB-Instance gibt.

Sie können diese Metriken mit der CloudWatch-Konsole, der AWS CLI oder der CloudWatch-API untersuchen. Sie können auch andere Performance-Insights-Zählermetriken mithilfe einer speziellen mathematischen Metrikfunktion untersuchen. Weitere Informationen finden Sie unter Abfragen anderer Performance-Insights-Zählermetriken in CloudWatch.

Um beispielsweise die statistischen Daten zur Metrik `DBLoad` anzuzeigen, führen Sie den Befehl get-metric-statistics aus.

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
    --end-time 1532036185 \
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

Dieses Beispiel generiert eine Ausgabe wie die folgende.

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
  {
  "Timestamp": "2021-07-19T21:31:00Z",
  "Unit": "None",
  "Average": 1.5
  },
  {
  "Timestamp": "2021-07-19T21:32:00Z",
  "Unit": "None",
  "Average": 1.8
  },
  {
  "Timestamp": "2021-07-19T21:29:00Z",
  "Unit": "None",
  "Average": 3.0
  },
  {
  "Timestamp": "2021-07-19T21:33:00Z",
  "Unit": "None",
  "Average": 2.4
  }
  ],
  "Label": "DBLoad"
  }
```

Weitere Informationen über CloudWatch finden Sie unter Was ist Amazon CloudWatch? im Amazon CloudWatch-Benutzerhandbuch.

Abfragen anderer Performance-Insights-Zählermetriken in CloudWatch

> **Note**
>
> Wenn Sie den erweiterten Modus von Database Insights aktivieren, veröffentlicht Amazon RDS Performance-Insights-Zählermetriken für Amazon CloudWatch. Mit Database Insights müssen Sie die metrische mathematische Funktion `DB_PERF_INSIGHTS` nicht verwenden. Sie können das Dashboard von CloudWatch Database Insights verwenden, um nach Performance-Insights-Zählermetriken zu suchen, sie abzufragen und Alarme festzulegen.

Sie können RDS-Performance-Insights-Metriken von CloudWatch abfragen, Alarme ausgeben und Grafiken erstellen. Sie können auf Informationen zu Ihrem DB-Cluster zugreifen, indem Sie die `DB_PERF_INSIGHTS` metrische mathematische Funktion für CloudWatch verwenden. Mit dieser Funktion können Sie die Performance-Insights-Metriken verwenden, die nicht direkt an CloudWatch gemeldet werden, um eine neue Zeitreihe zu erstellen.

Sie können die neue mathematische Metrikfunktion verwenden, indem Sie in der CloudWatch-Konsole im Bildschirm Metrik auswählen auf das Dropdown-Menü Mathematik hinzufügen klicken. Sie können damit Alarme und Grafiken zu Performance-Insights-Metriken oder zu Kombinationen von CloudWatch- und Performance-Insights-Metriken erstellen, einschließlich hochauflösender Alarme für Metriken unter einer Minute. Sie können die Funktion auch programmgesteuert verwenden, indem Sie den metrischen mathematischen Ausdruck in `get-metric-data`-Anforderung aufnehmen. Weitere Informationen finden Sie unter Syntax und Funktionen von Metrikberechnungen und Erstellen eines Alarms für Performance-Insights-Zählermetriken aus einer AWS-Datenbank.

Performance-Insights-Zählermetriken

Zählermetriken sind Betriebssystem- und Datenbank-Performance-Metriken im Performance-Insights-Dashboard. Um Leistungsprobleme zu identifizieren und zu analysieren, können Sie Zählermetriken mit der DB-Last korrelieren. Sie müssen der Metrik eine Statistikfunktion hinzufügen, um die Metrikwerte abzurufen. Die unterstützten Funktionen für die Metrik `os.memory.active` sind beispielsweise `.avg`, `.min`, `.max`,`.sum` und `.sample_count`.

Die Zählermetriken werden einmal pro Minute erfasst. Die Erfassung der Betriebssystemmetriken hängt davon ab, ob die erweiterte Überwachung aktiviert oder deaktiviert ist. Wenn die erweiterte

Überwachung deaktiviert ist, werden die Betriebssystemmetriken einmal pro Minute erfasst. Ist die erweiterte Überwachung aktiviert, werden die Betriebssystemmetriken für den ausgewählten Zeitraum erfasst. Weitere Informationen zum Aktivieren und Deaktivieren der erweiterten Überwachung finden Sie unter Aktivieren und Deaktivieren von „Erweiterte Überwachung“.

Themen

- Performance-Insights-Betriebssystemzähler
- Performance-Insights-Zähler für Aurora MySQL
- Performance-Insights-Zähler für Aurora PostgreSQL

Performance-Insights-Betriebssystemzähler

Die folgenden Betriebssystemzähler, denen `os` vorangestellt ist, sind bei Performance Insights für Aurora PostgreSQL und Aurora MySQL verfügbar.

Sie können die `ListAvailableResourceMetrics`-API für die Liste der verfügbaren Zählermetriken für Ihre DB-Instance verwenden. Weitere Informationen finden Sie ListAvailableResourceMetricsim Amazon RDS Performance Insights API-Referenzhandbuch.

Zähler	Typ	Einheit	Metrik	Description
Aktiv	Arbeitsspeicher	Kilobytes	os.memory .active	Umfang des zugewiesenen Arbeitsspeichers in Kilobyte.
Puffer	Arbeitsspeicher	Kilobytes	os.memory .buffers	Die Speicherm enge, die für die Pufferung von I/O Anfragen vor dem Schreiben auf das Speicherg erät verwendet wird, in Kilobyte.

Zähler	Typ	Einheit	Metrik	Description
Cached	Arbeitsspeicher	Kilobytes	os.memory .cached	Die Größe des verwendeten Arbeitsspeichers für das Caching von Dateisyst em-basierter I/O in Kilobyte.
DB Cache	Arbeitsspeicher	Bytes	os.memory .db.cache	Die Größe des Arbeitssp eichers, die vom Datenbank prozess einschlie ßlich tmpfs (shmem) für den Seiten-Cache verwendet wird, in Byte.
DB Resident Set Size	Arbeitsspeicher	Bytes	os.memory.db. residentSetSize	Die Größe des Arbeitssp eichers, die vom Datenbank prozess ohne tmpfs (shmem) für den anonymen Cache und den Swap-Cache verwendet wird, in Byte.

Zähler	Typ	Einheit	Metrik	Description
DB Swap	Arbeitsspeicher	Bytes	os.memory .db.swap	Die Größe des Arbeitssp eichers, die vom Datenbank prozess für Swap verwendet wird, in Byte.
Dirty	Arbeitsspeicher	Kilobytes	os.memory.dirty	Menge an Memory-Pages im RAM, die geändert, aber noch nicht in den entsprech enden Datenbloc k im Speicher geschrieben wurden, in Kilobyte.
Kostenfrei	Arbeitsspeicher	Kilobytes	os.memory.free	Umfang des nicht zugewiese nen Arbeitssp eichers in Kilobyte.
Huge Pages frei	Arbeitsspeicher	Seiten	Betriebss ystemspeicher. hugePagesFree	Die Anzahl von freien "Huge Pages". "Huge Pages" sind eine Funktion des Linux-Kernel.

Zähler	Typ	Einheit	Metrik	Description
Huge Pages reserviert	Arbeitsspeicher	Seiten	Betriebssystemspeicher.hugePagesRsvd	Die Anzahl von gebundenen "Huge Pages".
Größe Huge Pages	Arbeitsspeicher	Kilobytes	Betriebssystemspeicher.hugePagesSize	Die Größe für jede Huge Pages-Einheit, in Kilobyte.
Überschuss Huge Pages	Arbeitsspeicher	Seiten	Betriebssystemspeicher.hugePagesSurp	Die Anzahl der verfügbaren überzähligen Huge Pages.
Huge Pages insgesamt	Arbeitsspeicher	Seiten	Betriebssystemspeicher.hugePagesTotal	Die Gesamtzahl der Huge Pages.
Inaktiv	Arbeitsspeicher	Kilobytes	os.memory.inactive	Umfang der am seltensten verwendeten Memory-Pages in Kilobyte.
Mapped	Arbeitsspeicher	Kilobytes	os.memory.mapped	Die Gesamtmenge der Dateisysteminhalte, die Arbeitsspeicher in einem Prozess-Adressraum zugeordnet ist, in Kilobyte.

Zähler	Typ	Einheit	Metrik	Description
Out of Memory Kill Count	Arbeitsspeicher	Beendigungen	Betriebssystemspeicher.outOfMemoryKillCount	Die Anzahl der OOM-Kills im letzten Erfassungsintervall.
Seitentabellen	Arbeitsspeicher	Kilobytes	os.memory.pageTables	Umfang des von Page-Tabellen verwendeten Arbeitsspeichers in Kilobyte.
Slab	Arbeitsspeicher	Kilobytes	os.memory.slab	Umfang der wiederverwendbaren Kernel-Datenstrukturen in Kilobyte.
Gesamt	Arbeitsspeicher	Kilobytes	os.memory.total	Gesamtumfang des Arbeitsspeichers in Kilobyte.
Writeback	Arbeitsspeicher	Kilobytes	os.memory.writeback	Gesamtumfang an modifizierten Speicherbereichen im RAM, die noch in den Sicherungsspeicher geschrieben werden, in Kilobyte.

Zähler	Typ	Einheit	Metrik	Description
Guest	CPU-Auslastung	Prozentsatz	os.cpuUtilization.guest	Der prozentuale Anteil der von Gastprogrammen verwendeten CPU.
Inaktiv	CPU-Auslastung	Prozentsatz	os.cpuUtilization.idle	Der prozentuale Anteil der CPU, der sich im Leerlauf befindet.
Irq	CPU-Auslastung	Prozentsatz	os.cpuUtilization.irq	Der prozentuale Anteil der von Software-Interrupts verwendeten CPU.
Nice	CPU-Auslastung	Prozentsatz	os.cpuUtilization.nice	Der prozentuale Anteil der von Programmen mit niedrigster Priorität verwendeten CPU.
Steal	CPU-Auslastung	Prozentsatz	os.cpuUtilization.steal	Der prozentuale Anteil der von virtuellen Maschinen verwendeten CPU.

Zähler	Typ	Einheit	Metrik	Description
System (System)	CPU-Auslastung	Prozentsatz	os.cpuUtilization.system	Der prozentuale Anteil der vom Kernel verwendeten CPU.
Gesamt	CPU-Auslastung	Prozentsatz	os.cpuUtilization.total	Der prozentuale Anteil der insgesamt verwendeten CPU. Diese Angabe enthält den Nice-Wert.
Benutzer	CPU-Auslastung	Prozentsatz	os.cpuUtilization.user	Der prozentuale Anteil der von Benutzerprogrammen verwendeten CPU.
Wait	CPU-Auslastung	Prozentsatz	os.cpuUtilization.wait	Der Prozentsatz der CPUs, die beim Warten auf den I/O Zugriff nicht genutzt wurden.
Aurora Storage Aurora Storage Bytes Rx	Festplatten-IO	Bytes pro Sekunde	os.Diskio.AuroraStorage.auroraStorageBytesRx	Die Anzahl der Bytes, die pro Sekunde vom Aurora-Speicher empfangen werden.

Zähler	Typ	Einheit	Metrik	Description
Aurora Storage Aurora Storage Bytes Tx	Festplatten-IO	Bytes pro Sekunde	OS.Diskio .AuroraStorage. auroraSto rageBytesTx	Die Anzahl der Bytes, die pro Sekunde in den Aurora-Speicher hochgeladen werden.
Aurora Storage Disk Queue Depth	Festplatten-IO	Anforderungen	OS.Diskio .AuroraStorage. diskQueueDepth	Die Länge der Aurora-Speicher-Festplatten warteschlange.
Aurora Storage Read IOs PS	Festplatten-IO	Anforderungen pro Sekunde	os.Diskio .AuroraSt orage.PS lesen IOs	Die Anzahl der Lesevorgänge pro Sekunde.
Aurora Storage Read Latency	Festplatten-IO	Millisekunden	os.diskIO .auroraSt orage.rea dLatency	Die durchschn ittliche Latenz einer I/O Leseanforderung an den Aurora-Speicher in Millisekunden.
Aurora Storage Read Throughpu t	Festplatten-IO	Bytes pro Sekunde	os.diskIO .auroraSt orage.rea dThroughput	Die Menge des Netzwerkd urchsatzes, der von Anforderu ngen an den DB-Cluster verwendet wird, in Bytes pro Sekunde.

Zähler	Typ	Einheit	Metrik	Description
Aurora Storage Write IOs PS	Festplatten-IO	Anforderungen pro Sekunde	os.diskio.AuroraStorage.PS schreiben IOs	Die Anzahl der Schreibvorgänge pro Sekunde.
Aurora Storage Write Latency	Festplatten-IO	Millisekunden	os.diskIO.auroraStorage.writeLatency	Die durchschnittliche Latenz einer I/O Schreibanforderung an den Aurora-Speicher in Millisekunden.
Aurora Storage Write Throughput	Festplatten-IO	Bytes pro Sekunde	os.diskIO.auroraStorage.writeThroughput	Die Menge des Netzwerkdurchsatzes, der von Antworten vom DB-Cluster verwendet wird, in Bytes pro Sekunde.
Rdstemp Avg Queue Len	Festplatten-IO	Anforderungen	os.Diskio.rdstemp.avgQueueLen	Die Anzahl der Anfragen, die in der Warteschlange des I/O Geräts warten.
Rdstemp Avg Req Sz	Festplatten-IO	Anforderungen	os.DiskIO.rdstemp.avgReqSz	Die Anzahl der Anfragen, die in der Warteschlange des I/O Geräts warten.

Zähler	Typ	Einheit	Metrik	Description
Rdstemp Await	Festplatten-IO	Millisekunden	os.diskIO .rdstemp.await	Die erforderliche Anzahl an Millisekunden für die Antwort auf Anfragen, einschließlich Warteschlangen- und Servicedauer.
Rstempp Lesen Sie PS IOs	Festplatten-IO	Anforderungen	os.Diskio .rdstemp.PS lesen IOs	Die Anzahl der Lesevorgänge pro Sekunde.
Rdstemp Read KB	Festplatten-IO	Kilobytes	os.diskIO .rdstemp.readKb	Gesamtzahl der gelesenen Kilobyte.
Rdstemp Read KB PS	Festplatten-IO	Kilobytes pro Sekunde	os.diskIO .rdstemp. readKbPS	Die Anzahl der pro Sekunde gelesenen Kilobyte.
Rdstemp Rrqm PS	Festplatten-IO	Anforderungen pro Sekunde	os.diskIO .rdstemp.rrqmPS	Die Anzahl der zusammengeführten Leseanfragen in der Warteschlange pro Sekunde.
Rdstemp TPS	Festplatten-IO	Transaktionen pro Sekunde	os.diskIO .rdstemp.tps	Die Anzahl der I/O Transaktionen pro Sekunde.

Zähler	Typ	Einheit	Metrik	Description
Rdstemp Util	Festplatten-IO	Prozentsatz	os.diskIO .rdstemp.util	Der Prozentsatz der CPU-Zeit, während der Anfragen ausgegeben wurden.
Rstemp Write PS IOs	Festplatten-IO	Anforderungen pro Sekunde	os.DiskIO .rdstemp.PS schreiben IOs	Die Anzahl der Schreibvorgänge pro Sekunde.
Rdstemp Write KB	Festplatten-IO	Kilobytes	os.diskIO .rdstemp.writeKb	Gesamtzahl der geschriebenen Kilobyte.
Rdstemp Write KB PS	Festplatten-IO	Kilobytes pro Sekunde	os.diskIO .rdstemp. writeKbPS	Die Anzahl der pro Sekunde geschriebenen Kilobyte.
Rdstemp Wrqm PS	Festplatten-IO	Anforderungen pro Sekunde	os.diskIO .rdstemp. wrqmPS	Die Anzahl der zusammengeführten Schreibanfragen in der Warteschlange pro Sekunde.
Blocked	Aufgaben	Aufgaben	os.tasks.blocked	Die Anzahl von blockierten Aufgaben.
In Ausführung	Aufgaben	Aufgaben	os.tasks.running	Die Anzahl von laufenden Aufgaben.

Zähler	Typ	Einheit	Metrik	Description
Sleeping	Aufgaben	Aufgaben	os.tasks.sleeping	Die Anzahl von Aufgaben im Ruhezustand.
Angehalten	Aufgaben	Aufgaben	os.tasks.stopped	Die Anzahl von angehaltenen Aufgaben.
Gesamt	Aufgaben	Aufgaben	os.tasks.total	Die Gesamtanzahl der Aufgaben.
Zombie	Aufgaben	Aufgaben	os.tasks.zombie	Die Anzahl der untergeordneten Aufgaben, die unter einer aktiven übergeordneten Aufgabe inaktiv sind.
One	Durchschnittliche Auslastung Minute	Prozesse	OS. loadAveragеMinute. eins	Die Anzahl der Prozesse, die während der letzten Minute CPU-Zeit angefordert haben.
Fifteen	Durchschnittliche Auslastung Minute	Prozesse	os. loadAveragеMinute. fünfzehn	Die Anzahl der Prozesse, die während der letzten 15 Minuten CPU-Zeit angefordert haben.

Zähler	Typ	Einheit	Metrik	Description
Five	Durchschnittliche Auslastung Minute	Prozesse	os. loadAvera geMinute. fünf	Die Anzahl der Prozesse, die während der letzten 5 Minuten CPU-Zeit angefordert haben.
Cached	Auslagerung	Kilobytes	os.swap.cached	Umfang des Swap-Arbe itsspeichers, der als Cache-Spe icher verwendet wird, in Kilobyte.
Kostenfrei	Auslagerung	Kilobytes	os.swap.free	Die Menge des freien Swap-Arbeitsspeichers in Kilobyte.
In	Auslagerung	Kilobytes	os.swap.in	Die Menge des von der Festplatt e ausgelagerten Speichers in Kilobyte.
Out	Auslagerung	Kilobytes	os.swap.out	Die Menge des auf die Festplatt e ausgelagerten Speichers in Kilobyte.

Zähler	Typ	Einheit	Metrik	Description
Gesamt	Auslagerung	Kilobytes	os.swap.total	Die Gesamtmenge des verfügbaren Swap-Arbeitsspeichers in Kilobyte.
Max Files	Dateisystem	Dateien	os.fileSys.maxFiles	Die maximale Anzahl von Dateien, die für das Dateisystem auf allen Speichervolumes erstellt werden können.
Used Files	Dateisystem	Dateien	os.fileSys.usedFiles	Die Anzahl der Dateien im Dateisystem auf allen Speichervolumes.
Used File Percent	Dateisystem	Dateien	os.FileSys.usedFilePercent	Der Prozentsatz der verfügbaren Dateien, die auf allen Speichervolumes verwendet werden.

Zähler	Typ	Einheit	Metrik	Description
Used Percent	Dateisystem	Prozentsatz	os.fileSys.usedPercent	Der Prozentsatz des belegten Dateisystem-Festplattenspeichers auf allen Speichervolumes.
Used	Dateisystem	Kilobytes	os.fileSys.used	Die Menge an Festplattenspeicher, die von Dateien im Dateisystem auf allen Speichervolumes belegt wird, in Kilobyte.
Gesamt	Dateisystem	Kilobytes	os.fileSys.total	Der gesamte Festplattenspeicher, der für das Dateisystem auf allen Speichervolumes verfügbar ist, in Kilobyte.
Max Files	Dateisystem	Dateien	Betriebssystem.FileSys.<volumeName>.max-Dateien	Die maximale Anzahl von Dateien, die für das Speichervolume erstellt werden können.

Zähler	Typ	Einheit	Metrik	Description
Used Files	Dateisystem	Dateien	os.FileSys.<volumeName>.usedFiles	Die Anzahl der Dateien im Speichervolume.
Used File Percent	Dateisystem	Dateien	Betriebssystem.FileSys.<volumeName>.usedFilePercent	Der Prozentsatz der verfügbaren Dateien, die im Speichervolumen verwendet werden.
Used Percent	Dateisystem	Prozentsatz	os.FileSys.<volumeName>.usedPercent	Der Prozentsatz des belegten Speicherplatzes des Speichervolumens.
Used	Dateisystem	Kilobytes	os.FileSys.<volumeName>.gebraucht	Die Menge an Festplattenspeicher, die von Dateien auf dem Speichervolume belegt wird, in Kilobyte.
Gesamt	Dateisystem	Kilobytes	Betriebssystem.FileSys.<volumeName>.insgesamt	Der gesamte auf dem Speichervolume verfügbare Festplattenspeicher in Kilobyte.

Zähler	Typ	Einheit	Metrik	Description
Rx	Netzwerk	Bytes pro Sekunde	os.network.rx	Die Anzahl der pro Sekunde empfangenen Byte.
Tx	Netzwerk	Bytes pro Sekunde	os.network.tx	Die Anzahl der pro Sekunde hochgeladenen Byte.
Acu Utilization	Allgemeines	Prozentsatz	os.general.acuUtilization	Der Anteil der aktuellen Kapazität an der maximal konfigurierten Kapazität in Prozent.
Max Configured Acu	Allgemeines	ACUs	os.allgemein.maxConfiguredAcu	Die vom Benutzer konfigurierte maximale Kapazität in Aurora-Kapazitätseinheiten (ACUs).
Min Configured Acu	Allgemeines	ACUs	os.allgemein.minConfiguredAcu	Die vom Benutzer konfigurierte Mindestkapazität, in ACUs.

Zähler	Typ	Einheit	Metrik	Description
Num VCPUs	Allgemeines	v CPUs	os.general.num VCPUs	Die Anzahl der virtuellen CPUs (vCPUs) für die DB-Instance.
Serverles s Database Capacity	Allgemeines	ACUs	os.genera l. serverles sDatabase Capacity	Die aktuelle Kapazität der Instanz, in ACUs.

Performance-Insights-Zähler für Aurora MySQL

Die folgenden Datenbankzähler sind bei Performance Insights für Aurora MySQL verfügbar.

Themen

- Native Zähler für Aurora MySQL
- Nicht-native Zähler für Aurora MySQL

Native Zähler für Aurora MySQL

Native Metriken werden von der Datenbank-Engine und nicht von Amazon Aurora definiert. Sie finden Definitionen dieser nativen Metriken unter Serverstatusvariablen in der MySQL-Dokumentation.

Zähler	Typ	Einheit	Metrik
Com_analyze	SQL	Abfragen pro Sekunde	db.SQL.Com_analyze
Com_optimize	SQL	Abfragen pro Sekunde	db.SQL.Com_optimize

Zähler	Typ	Einheit	Metrik
Com_select	SQL	Abfragen pro Sekunde	db.SQL.Com_select
Innodb_rows_deleted	SQL	Zeilen pro Sekunde	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	Zeilen pro Sekunde	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	Zeilen pro Sekunde	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	Zeilen pro Sekunde	db.SQL.Innodb_rows_updated
Abfragen	SQL	Abfragen pro Sekunde	db.SQL.Queries
Fragen	SQL	Abfragen pro Sekunde	db.SQL.Questions
Select_full_join	SQL	Abfragen pro Sekunde	db.SQL.Select_full_join
Select_full_range_join	SQL	Abfragen pro Sekunde	db.SQL.Select_full_range_join
Bereich auswählen	SQL	Abfragen pro Sekunde	db.SQL.Select_range

Zähler	Typ	Einheit	Metrik
Select_range_check	SQL	Abfragen pro Sekunde	db.SQL.Select_range_check
Select_scan	SQL	Abfragen pro Sekunde	db.SQL.Select_scan
Slow_queries	SQL	Abfragen pro Sekunde	db.SQL.Slow_queries
Sort_merge_passes	SQL	Abfragen pro Sekunde	db.SQL.Sort_merge_passes
Sort_range	SQL	Abfragen pro Sekunde	db.SQL.Sort_range
Sort_rows	SQL	Abfragen pro Sekunde	db.SQL.Sort_rows
Sort_scan	SQL	Abfragen pro Sekunde	db.SQL.Sort_scan
Total_query_time	SQL	Millisekunden	db.SQL.Total_query_time
Table_locks_immediate	Sperren	Anforderungen pro Sekunde	db.Locks.Table_locks_immediate

Zähler	Typ	Einheit	Metrik
Table_locks_waited	Sperren	Anforderungen pro Sekunde	db.Locks.Table_locks_waited
Innodb_row_lock_time	Sperren	Millisekunden (durchschnittlich)	db.Locks.Innodb_row_lock_time
Aborted_clients	Benutzer	Verbindungen	db.Users.Aborted_clients
Aborted_connects	Benutzer	Verbindungen	db.Users.Aborted_connects
Verbindungen	Benutzer	Verbindungen	db.Users.Connections
External_threads_connected	Benutzer	Verbindungen	db.Users.External_threads_connected
max_connections	Benutzer	Verbindungen	db.Users.max_connections
Threads_connected	Benutzer	Verbindungen	db.Users.Threads_connected
Threads_created	Benutzer	Verbindungen	db.Users.Threads_created
Threads_running	Benutzer	Verbindungen	db.Users.Threads_running
Created_tmp_disk_tables	Temporäre Dateien	Tabellen pro Sekunde	db.Temp.Created_tmp_disk_tables

Zähler	Typ	Einheit	Metrik
Created_tmp_tables	Temporäre Dateien	Tabellen pro Sekunde	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages_data	Cache	Seiten	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	Cache	Seiten	db.Cache.Innodb_buffer_pool_pages_total
Innodb_buffer_pool_read_requests	Cache	Seiten pro Sekunde	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	Cache	Seiten pro Sekunde	db.Cache.Innodb_buffer_pool_reads
Opened_tables	Cache	Tabellen	db.Cache.Opened_tables
Opened_table_definitions	Cache	Tabellen	db.Cache.Opened_table_definitions
Qcache_hits	Cache	Abfragen	db.Cache.Qcache_hits

Nicht-native Zähler für Aurora MySQL

Nicht-native Zähler-Metriken sind durch Amazon RDS definierte Zähler. Eine nicht-native Metrik kann eine Metrik sein, die Sie mit einer bestimmten Abfrage erhalten. Eine nicht-native Metrik kann auch eine abgeleitete Metrik sein, bei der zwei oder mehrere native Zähler bei Berechnungen von Verhältnissen, Trefferraten oder Latenzen verwendet werden.

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
active_transactions	Transaktionen	db.Transactions.active_transaction	Die Gesamtzahl aktiver Transaktionen.	`SELECT COUNT(1) AS active_transactions FROM INFORMATION_SC`	

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
				HEMA.INNODB_TRX	
innodb_buffer_pool_hit_rate	Cache	db.Cac innoDB ffer_po _hit_rat	Der Prozentsatz der Lesevorgänge, den InnoDB aus dem Pufferpool verarbeiten konnte.	100 * innodb_buffer_pool_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)	
innodb_buffer_pool_hits	Cache	Seiten pro Sekunc	db.Cache.innoDB_buffer_pool_hits	Die Anzahl der Lesevorgänge, die InnoDB aus dem Pufferpool verarbeiten konnte.	innodb_buffer_pool_read_requests - innodb_buffer_pool_reads

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
innodb_buffer_pool_usage	Cache	Prozentz	db.Cache.innoDB_buffer_pool_usage	Die Prozentsatz des InnoDB-Pufferpools, der Daten (Seiten) enthält. **Note** Bei komprimierten Tabellen kann dieser Wert variieren. Weitere Informationen finden Sie in den Angaben zu `Innodb_buffer_pool_pages_data` und `Innodb_buffer_pool_pages_total` unter Serverstatusvariablen in der MySQL-Dokumentation.	`Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0`

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
innodb_deadlocks	Sperrer	db.Locl innodb_ adlocks	Die Gesamtzah l von Deadlocks .	`SELECT COUNT AS innodb_de adlocks FROM INFORMATION_SC HEMA.INNO DB_METRICS WHERE NAME= 'lock_dea dlocks'`	
innodb_lo ck_timeouts	Sperrer	db.Locl innodb_ ck_time ts	Die Gesamtanz ahl von Deadlocks , die das Zeitlimit überschri tten haben.	`SELECT COUNT AS innodb_lo ck_timeouts FROM INFOR MATION_SC HEMA.INNO DB_METRICS WHERE NAME= 'lock_tim eouts'`	
innodb_ro w_lock_waits	Sperrer	db.Locl innodb_ w_lock_ its	Die Gesamtanz ahl von Zeilenspe rren, die zu einer Wartezeit geführt haben.	`SELECT COUNT AS innodb_ro w_lock_waits FROM INFOR MATION_SC HEMA.INNO DB_METRICS WHERE NAME ='lock_ro w_lock_waits'`	

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
innodb_rows_changed	SQL	db.SQL nodb_r _chang	Die Gesamtzah l von InnoDB-Ze ilenvorgä ngen.	`db.SQL.In nodb_rows _inserted + db.SQL.In nodb_rows _deleted + db.SQL.I nnodb_row s_updated`	
query_cache_hit_rate	Cache	Prozen tz	db.Cache. query_cac he_hit_ra te	Die Trefferrate des MySQL-Ergebnissatz -Caches (Abfrage-Cache).	`Qcache_hits / (QCache_hits + Com_select) * 100`
temp_disk_table_percent	Tempo Dateier	db.Tem emp_di tables_ rcent	Der Prozentsa tz der temporäre n Tabellen, die vom Server bei der Ausführun g von Anweisung en auf der Festplatt e erstellt werden.	`(db.Temp. Created_t mp_disk_t ables / db.Temp.C reated_tm p_tables) * 100`	

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
trx_rseg_history_len	Transaktionen	Keine	db.Transactions.trx_rseg_history_len	Eine Liste der Undo-Protokollseiten für übernommene Transaktionen, die vom InnoDB-Transaktionssystem verwaltet wird, um die Parallelitätskontrolle für mehrere Versionen zu implementieren. Weitere Informationen zu Undo-Protokolleinträgen finden Sie unter https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html in der MySQL-Dokumentation.	`SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'`

Performance-Insights-Zähler für Aurora PostgreSQL

Die folgenden Datenbankzähler sind bei Performance Insights für Aurora PostgreSQL verfügbar.

Themen

- Native Zähler für Aurora PostgreSQL
- Nicht-native Zähler für Aurora PostgreSQL

Native Zähler für Aurora PostgreSQL

Native Metriken werden von der Datenbank-Engine und nicht von Amazon Aurora definiert. Definitionen dieser nativen Metriken finden Sie unter Anzeigen von Statistiken in der PostgreSQL-Dokumentation.

Zähler	Typ	Einheit	Metrik
tup_deleted	SQL	Tupel pro Sekunde	db.SQL.tup_deleted
tup_fetched	SQL	Tupel pro Sekunde	db.SQL.tup_fetched
tup_inserted	SQL	Tupel pro Sekunde	db.SQL.tup_inserted
tup_returned	SQL	Tupel pro Sekunde	db.SQL.tup_returned
tup_updated	SQL	Tupel pro Sekunde	db.SQL.tup_updated
blks_hit	Cache	Blöcke pro Sekunde	db.Cache.blks_hit
buffers_alloc	Cache	Blöcke pro Sekunde	db.Cache.buffers_alloc
buffers_checkpoint	Prüfpunkt	Blöcke pro Sekunde	db.Checkpoint.buffers_checkpoint
checkpoints_req	Prüfpunkt	Prüfpunkte pro Minute	db.Checkpoint.checkpoints_req
checkpoint_sync_time	Prüfpunkt	Millisekunden pro Prüfpunkt	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	Prüfpunkt	Prüfpunkte pro Minute	db.Checkpoint.checkpoints_timed
checkpoint_write_time	Prüfpunkt	Millisekunden pro Prüfpunkt	db.Checkpoint.checkpoint_write_time
maxwritten_clean	Prüfpunkt	Bgwriter-Bereinigungsstopps pro Minute	db.Checkpoint.maxwritten_clean

Zähler	Typ	Einheit	Metrik
Deadlocks	Gleichzeitigkeit	Deadlocks pro Minute	db.Concurrency.deadlocks
blk_read_time	I/O	Millisekunden	db.IO.blk_read_time
blks_read	I/O	Blöcke pro Sekunde	db.IO.blks_read
buffers_backend	I/O	Blöcke pro Sekunde	db.IO.buffers_backend
buffers_backend_fsync	I/O	Blöcke pro Sekunde	db.IO.buffers_backend_fsync
buffers_clean	I/O	Blöcke pro Sekunde	db.IO.buffers_clean
temp_bytes	Temporäre Dateien	Bytes pro Sekunde	db.Temp.temp_bytes
temp_files	Temporäre Dateien	Dateien pro Minute	db.Temp.temp_files
xact_commit	Transaktionen	Commits pro Sekunde	db.Transactions.xact_commit
xact_rollback	Transaktionen	Rollbacks pro Sekunde	db.Transactions.xact_rollback
numbackends	Benutzer	Verbindungen	db.User.numbackends
archived_count	WAL	Dateien pro Minute	db.WAL.archived_count

Nicht-native Zähler für Aurora PostgreSQL

Nicht-native Zähler-Metriken sind durch Amazon Aurora definierte Zähler. Eine nicht-native Metrik kann eine Metrik sein, die Sie mit einer bestimmten Abfrage erhalten. Eine nicht-native Metrik kann auch eine abgeleitete Metrik sein, bei der zwei oder mehrere native Zähler bei Berechnungen von Verhältnissen, Trefferraten oder Latenzen verwendet werden.

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
checkpoint_sync_latency	Prüfpunkt	Millisekunden	db.Checkpoint.checkpoint_sync_latency	Die Gesamtzeit, die auf den Teil der Prüfpunktverarbeitung aufgewendet wurde, bei dem Dateien auf der Festplatte synchronisiert werden.	`checkpoint_sync_time / (checkpoints_timed + checkpoints_req)`
checkpoint_write_latency	Prüfpunkt	Millisekunden	db.Checkpoint.checkpoint_write_latency	Die Gesamtzeit, die auf den Teil der Prüfpunktverarbeitung aufgewendet wurde, bei dem Dateien auf die Festplatte geschrieben werden.	`checkpoint_write_time / (checkpoints_timed + checkpoints_req)`
local_blks_read	I/O	Blöcke	db.IO.local_blks_read	Gesamtzahl der gelesenen lokalen Blöcke	Nicht zutreffend
local_blk_read_time	I/O	Millisekunden	db.IO.local_blk_read_time	Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit für das Lesen von lokalen Datendateiblöcken in Millisekunden aufgezeichnet. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.	Nicht zutreffend

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
num_blocked_sessions	Sperren	db.Locks.num_blocked_sessions	Die Anzahl der blockierten Sitzungen.	–	
orcache_blks_hit	I/O	Abfragen	db.IO.orcache_blks_hit	Gesamtzahl der Treffer von gemeinsam genutzten Blöcken aus dem Cache für optimierte Lesevorgänge	Nicht zutreffend
orcache_blk_read_time	I/O	Millisekunden	db.IO.orcache_blk_read_time	Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit für das Lesen von Datendateiblöcken aus dem Cache für optimierte Lesevorgänge in Millisekunden aufgezeichnet. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.	Nicht zutreffend

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
read_latency	I/O	Millisekunden	db.IO.read_latency	Die Zeit, die für das Lesen von Datendateiblöcken durch Backends in dieser Instance aufgewendet wurde.	`blk_read_time / blks_read`
storage_blks_read	I/O	Blöcke	db.IO.storage_blks_read	Gesamtzahl der gemeinsam genutzten Blöcke, die aus dem Aurora-Speicher gelesen wurden	Nicht zutreffend
storage_blk_read_time	I/O	Millisekunden	db.IO.storage_blk_read_time	Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit für das Lesen von Datendateiblöcken aus dem Aurora-Speicher in Millisekunden aufgezeichnet. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.	Nicht zutreffend
num_blocked_sessions	Sperren	db.Locks.num_blocked_sessions	Die Anzahl der blockierten Sitzungen.	–	

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
active_count	Status	Sitzungen	db.state.active_count	Die Anzahl der Sitzungen im Status `active`.	Nicht zutreffend
idle_count	Status	Sitzungen	db.state.idle_count	Die Anzahl der Sitzungen im Status `idle`.	Nicht zutreffend
idle_in_transaction_aborted_count	Status	Sitzungen	db.state.idle_in_transaction_aborted_count	Die Anzahl der Sitzungen im Status `idle in transaction (aborted)` .	Nicht zutreffend
idle_in_transaction_count	Status	Sitzungen	db.state.idle_in_transaction_count	Die Anzahl der Sitzungen im Status `idle in transaction` .	Nicht zutreffend
idle_in_transaction_max_time	Status	Sekunden	db.state.idle_in_transaction_max_time	Die Dauer der am längsten ausgeführten Transaktion im `idle in transaction` -Status, in Sekunden.	Nicht zutreffend
logical_reads	SQL	Blöcke	db.SQL.logical_reads	Gesamtzahl der Blöcke, auf die zugegriffen und die gelesen wurden.	`blks_hit + blks_read`
queries_started	SQL	Abfragen	db.SQL.queries	Die Anzahl der gestarteten Abfragen.	Nicht zutreffend
queries_finished	SQL	Abfragen	db.SQL.queries	Die Anzahl der abgeschlossenen Abfragen.	Nicht zutreffend

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
total_query_time	SQL	Millisekunden	db.SQL.total_query_time	Die Gesamtzeit, die für die Ausführung von Anweisungen aufgewendet wurde, in Millisekunden.	Nicht zutreffend
active_transactions	Transaktionen	Transaktionen	db.Transactions.active_transactions	Die Anzahl der aktiven Transaktionen.	Nicht zutreffend
blocked_transactions	Transaktionen	Transaktionen	db.Transactions.blocked_transactions	Die Anzahl der blockierten Transaktionen.	Nicht zutreffend
commit_latency	Transaktionen	Mikrosekunden	db.Transactions.commit_latency	Die durchschnittliche Dauer von Commit-Vorgängen.	`db.Transactions.duration_commits / db.Transactions.xact_commit`
duration_commits	Transaktionen	Millisekunden	db.Transactions.duration_commits	Die gesamte Transaktionszeit, die in der letzten Minute aufgewendet wurde, in Millisekunden.	Nicht zutreffend
max_used_xact_ids	Transaktionen	Transaktionen	db.Transactions.max_used_xact_ids	Die Anzahl der Transaktionen, die nicht bereinigt wurden.	Nicht zutreffend

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
oldest_inactive_logical_replication_slot_xid_age	Transaktionen	Länge	db.Transactions.oldest_inactive_logical_replication_slot_xid_age	Das Alter der ältesten Transaktion in einem inaktiven logischen Replikations-Slot.	Nicht zutreffend
oldest_active_logical_replication_slot_xid_age	Transaktionen	Länge	db.Transactions.oldest_active_logical_replication_slot_xid_age	Das Alter der ältesten Transaktion in einem aktiven logischen Replikations-Slot.	Nicht zutreffend
oldest_reader_feedback_xid_age	Transaktionen	Länge	db.Transactions.oldest_reader_feedback_xid_age	Das Alter der ältesten Transaktion einer Transaktion mit langer Laufzeit auf einer Aurora-Reader-Instance oder einer globalen Aurora-DB-Reader-Instance.	Nicht zutreffend
oldest_prepared_transaction_xid_age	Transaktionen	Länge	db.Transactions.oldest_prepared_transaction_xid_age	Das Alter der ältesten vorbereiteten Transaktion.	Nicht zutreffend

Zähler	Typ	Einheit	Metrik	Beschreibung	Definition
oldest_running_transaction_xid_age	Transaktionen	Länge	db.Transactions.oldest_running_transaction_xid_age	Das Alter der ältesten ausgeführten Transaktion.	Nicht zutreffend
max_connections	Benutzer	Benutzer	db.User.max_connections	Die maximale zulässige Anzahl von Verbindungen für eine Datenbank, wie im Parameter `max_connections` konfiguriert.	Nicht zutreffend
total_auth_attempts	Benutzer	Benutzer	db.User.total_auth_attempts	Die Anzahl der Versuche, eine Verbindung zu dieser Instance herzustellen.	Nicht zutreffend
archive_failed_count	WAL	Dateien pro Minute	db.WAL.archive_failed_count	Die Anzahl der fehlgeschlagenen Versuche, WAL-Dateien zu archivieren, in Dateien pro Minute.	Nicht zutreffend

SQL-Statistiken für Performance Insights

SQL-Statistiken sind leistungsbezogene Metriken zu SQL-Abfragen, die von Performance Insights gesammelt werden. Performance Insights sammelt Statistiken für jede Sekunde, in der eine Abfrage ausgeführt wird, und für jeden SQL-Aufruf. Die SQL-Statistiken bilden den Durchschnitt für den ausgewählten Zeitraum ab.

Ein SQL-Digest setzt sich aus allen Abfragen zusammen, die ein bestimmtes Muster haben, aber nicht unbedingt dieselben Literalwerte haben. Der Digest ersetzt Literalwerte durch ein

Fragezeichen. Beispiel, `SELECT * FROM emp WHERE lname= ?`. Dieser Digest kann die folgenden untergeordneten Abfragen enthalten:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Alle Engines unterstützen SQL-Statistiken für Digest-Abfragen.

Informationen zur Unterstützung dieser Funktion nach Region, DB-Engine und Instance-Klasse finden Sie unter DB-Engine-, Regions- und Instance-Klassenunterstützung von Amazon Aurora für Performance-Insights-Funktionen.

Themen

- SQL-Statistiken für Aurora MySQL
- SQL-Statistiken für Aurora PostgreSQL

SQL-Statistiken für Aurora MySQL

Aurora MySQL sammelt SQL-Statistiken nur auf Digest-Ebene. Auf der Statement-Ebene werden keine Statistiken angezeigt.

Themen

- Digest-Statistiken für Aurora MySQL
- Statistiken für Aurora MySQL
- Statistiken für Aurora MySQL
- Primäre Statistiken für Aurora MySQL

Digest-Statistiken für Aurora MySQL

Performance Insights sammelt SQL-Digest-Statistiken aus der `events_statements_summary_by_digest`-Tabelle. Die `events_statements_summary_by_digest`-Tabelle wird von der Datenbank verwaltet.

Diese Tabelle verfügt nicht über eine Bereinigungsrichtlinie. Die folgende Meldung wird in der AWS-Managementkonsole angezeigt, wenn die Tabelle voll ist:

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

In dieser Situation verfolgen Aurora MySQL nicht SQL-Abfragen. Um dieses Problem zu beheben, kürzt Performance Insights die Digest-Tabelle automatisch, wenn die folgenden Bedingungen beide erfüllt sind:

- Die Tabelle ist voll.
- Performance Insights verwaltet das Leistungsschema automatisch.

  Für die automatische Verwaltung muss der Parameter `performance_schema` auf den Wert `0` festgelegt werden und Source (Quelle) auf `user` eingestellt sein. Wenn Performance Insights die Leistung nicht automatisch verwaltet, finden Sie weitere Informationen unter Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL.

Überprüfen Sie in der AWS CLI die Quelle eines Parameterwerts, indem Sie den Befehl describe-db-parameters ausführen.

Statistiken für Aurora MySQL

Die folgenden SQL-Statistiken stehen für Aurora MySQL-DB-Cluster zur Verfügung.

Metrik	Einheit
db.sql_tokenized.stats.count_star_per_sec	Aufrufe pro Sekunde
db.sql_tokenized.stats.sum_timer_wait_per_sec	Durchschnitt Latenz pro Sekunde (in ms)
db.sql_tokenized.stats.sum_select_full_join_per_sec	Vollständigen Join pro Sekunde auswählen
db.sql_tokenized.stats.sum_select_range_check_per_sec	Bereichsprüfung pro Sekunde auswählen
db.sql_tokenized.stats.sum_select_scan_per_sec	Scan pro Sekunde auswählen

Metrik	Einheit
db.sql_tokenized.stats.sum_sort_merge_passes_per_sec	Zusammenführungsdurchläufe pro Sekunde sortieren
db.sql_tokenized.stats.sum_sort_scan_per_sec	Scans pro Sekunde sortieren
db.sql_tokenized.stats.sum_sort_range_per_sec	Bereiche pro Sekunde sortieren
db.sql_tokenized.stats.sum_sort_rows_per_sec	Zeilen pro Sekunde sortieren
db.sql_tokenized.stats.sum_rows_affected_per_sec	Betroffene Zeilen pro Sekunde
db.sql_tokenized.stats.sum_rows_examined_per_sec	Überprüfte Zeilen pro Sekunde
db.sql_tokenized.stats.sum_rows_sent_per_sec	Gesendete Zeilen pro Sekunde
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec	Temporäre Datenträgertabellen pro Sekunde erstellt
db.sql_tokenized.stats.sum_created_tmp_tables_per_sec	Temporäre Tabellen pro Sekunde erstellt
db.sql_tokenized.stats.sum_lock_time_per_sec	Sperrzeit pro Sekunde (in ms)

Statistiken für Aurora MySQL

Die folgenden Metriken stellen Statistiken pro einzelnen Abruf für SQL-Anweisungen bereit.

Metrik	Einheit
db.sql_tokenized.stats.sum_timer_wait_per_call	Durchschnitt Latenz pro Aufruf (in ms)
db.sql_tokenized.stats.sum_select_full_join_per_call	Vollständige Joins pro Aufruf auswählen

Metrik	Einheit
db.sql_tokenized.stats.sum_select_range_check_per_call	Bereichsprüfung pro Aufruf auswählen
db.sql_tokenized.stats.sum_select_scan_per_call	Scans pro Aufruf auswählen
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	Zusammenführungsdurchläufe pro Aufruf sortieren
db.sql_tokenized.stats.sum_sort_scan_per_call	Scans pro Aufruf sortieren
db.sql_tokenized.stats.sum_sort_range_per_call	Bereiche pro Aufruf sortieren
db.sql_tokenized.stats.sum_sort_rows_per_call	Zeilen pro Aufruf sortieren
db.sql_tokenized.stats.sum_rows_affected_per_call	Betroffene Zeilen pro Aufruf
db.sql_tokenized.stats.sum_rows_examined_per_call	Überprüfte Zeilen pro Aufruf
db.sql_tokenized.stats.sum_rows_sent_per_call	Gesendete Zeilen pro Aufruf
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call	Temporäre Datenträgertabellen pro Aufruf erstellt
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	Temporäre Tabellen pro Aufruf erstellt
db.sql_tokenized.stats.sum_lock_time_per_call	Sperrzeit pro Aufruf (in ms)

Primäre Statistiken für Aurora MySQL

Die folgenden SQL-Statistiken stehen für Aurora MySQL-DB-Cluster zur Verfügung.

Metrik	Einheit
db.sql_tokenized.stats.count_star	Calls
db.sql_tokenized.stats.sum_timer_wait	Wartezeit (in ms)
db.sql_tokenized.stats.sum_select_full_join	Vollständigen Join auswählen
db.sql_tokenized.stats.sum_select_range_check	Wählen Sie Bereichsprüfungen
db.sql_tokenized.stats.sum_select_scan	Scandurchläufe auswählen
db.sql_tokenized.stats.sum_sort_merge_passes	Zusammenführungsdurchläufe sortieren
db.sql_tokenized.stats.sum_sort_scan	Sortierdurchläufe
db.sql_tokenized.stats.sum_sort_range	Sortierbereiche
db.sql_tokenized.stats.sum_sort_rows	Zeilen sortieren
db.sql_tokenized.stats.sum_rows_affected	Betroffene Zeilen
db.sql_tokenized.stats.sum_rows_examined	Überprüfte Zeilen
db.sql_tokenized.stats.sum_rows_sent	Gesendete Zeilen
db.sql_tokenized.stats.sum_created_tmp_disk_tables	Temporäre Datenträgertabellen erstellt
db.sql_tokenized.stats.sum_created_tmp_tables	Temporäre Tabellen erstellt
db.sql_tokenized.stats.sum_lock_time	Sperrzeit (in ms)

SQL-Statistiken für Aurora PostgreSQL

Performance Insights sammelt für jeden SQL-Aufruf und für jede Sekunde, in der eine Abfrage ausgeführt wird, SQL-Statistiken. Alle Aurora-Engines sammeln Statistiken nur auf Digest-Ebene.

Im Folgenden finden Sie Informationen zu Statistiken auf Digest-Ebene für Aurora PostgreSQL

Themen

- Digest-Statistiken für Aurora PostgreSQL
- Sekundengenaue Digest-Statistiken für Aurora PostgreSQL
- Digest-Statistiken pro Aufruf für Aurora PostgreSQL
- Primäre Statistiken für Aurora PostgreSQL

Digest-Statistiken für Aurora PostgreSQL

Um SQL Digest-Statistiken anzeigen zu können, muss die Bibliothek `pg_stat_statements` geladen sein. Diese Bibliothek wird für Aurora-PostgreSQL-DB-Cluster, die mit PostgreSQL 10 kompatibel sind, standardmäßig geladen. Für Aurora-PostgreSQL-DB-Cluster, die mit PostgreSQL 9.6 kompatibel sind, müssen Sie diese Bibliothek manuell aktivieren. Zur manuellen Aktivierung fügen Sie in der DB-Parametergruppe, die der DB-Instance zugeordnet ist, `pg_stat_statements` zu `shared_preload_libraries` hinzu. Starten Sie anschließend die DB-Instance neu. Weitere Informationen finden Sie unter Parametergruppen für Amazon Aurora.

> **Note**
>
> Performance-Insights kann nur Statistiken für nicht abgeschnittene Abfragen in `pg_stat_activity` erfassen. Standardmäßig kürzen PostgreSQL-Datenbanken Abfragen, die länger als 1 024 Bytes sind. Um die Abfragegröße zu erhöhen, ändern Sie den Parameter `track_activity_query_size` in der DB-Parametergruppe, die mit Ihrer DB-Instance verknüpft ist. Wenn Sie diesen Parameter ändern, ist ein Neustart der DB-Instance erforderlich.

Sekundengenaue Digest-Statistiken für Aurora PostgreSQL

Die folgenden SQL Digest-Statistiken sind für Aurora PostgreSQL DB-Instances verfügbar.

Metrik	Einheit
db.sql_tokenized.stats.calls_per_sec	Aufrufe pro Sekunde
db.sql_tokenized.stats.rows_per_sec	Zeilen pro Sekunde

Metrik	Einheit
db.sql_tokenized.stats.total_time_per_sec	Durchschnitt der aktiven Ausführungen (Average active executions, AAE) pro Sekunde
db.sql_tokenized.stats.shared_blks_hit_per_sec	Blocktreffer pro Sekunde
db.sql_tokenized.stats.shared_blks_read_per_sec	Blocklesevorgänge pro Sekunde
db.sql_tokenized.stats.shared_blks_dirtied_per_sec	Blöcke kontaminiert pro Sekunde
db.sql_tokenized.stats.shared_blks_written_per_sec	Blockschreibvorgänge pro Sekunde
db.sql_tokenized.stats.local_blks_hit_per_sec	Lokale Blocktreffer pro Sekunde
db.sql_tokenized.stats.local_blks_read_per_sec	Lokale Blocklesevorgänge pro Sekunde
db.sql_tokenized.stats.local_blks_dirtied_per_sec	Lokale Blockkontaminierungen pro Sekunde
db.sql_tokenized.stats.local_blks_written_per_sec	Lokale Blockschreibvorgänge pro Sekunde
db.sql_tokenized.stats.temp_blks_written_per_sec	Temporäre Schreibvorgänge pro Sekunde
db.sql_tokenized.stats.temp_blks_read_per_sec	Temporäre Lesevorgänge pro Sekunde
db.sql_tokenized.stats.blk_read_time_per_sec	Durchschnitt gleichzeitige Lesevorgänge pro Sekunde
db.sql_tokenized.stats.blk_write_time_per_sec	Durchschnitt gleichzeitige Schreibvorgänge pro Sekunde

Digest-Statistiken pro Aufruf für Aurora PostgreSQL

Die folgenden Metriken stellen Statistiken pro einzelnen Abruf für SQL-Anweisungen bereit.

Metrik	Einheit
db.sql_tokenized.stats.rows_per_call	Zeilen pro Aufruf
db.sql_tokenized.stats.avg_latency_per_call	Durchschnitt Latenz pro Aufruf (in ms)
db.sql_tokenized.stats.shared_blks_hit_per_call	Blocktreffer pro Aufruf
db.sql_tokenized.stats.shared_blks_read_per_call	Blocklesevorgänge pro Aufruf
db.sql_tokenized.stats.shared_blks_written_per_call	Blockschreibvorgänge pro Aufruf
db.sql_tokenized.stats.shared_blks_dirtied_per_call	Blöcke kontaminiert pro Aufruf
db.sql_tokenized.stats.local_blks_hit_per_call	Lokale Blocktreffer pro Aufruf
db.sql_tokenized.stats.local_blks_read_per_call	Lokale Blocklesevorgänge pro Aufruf
db.sql_tokenized.stats.local_blks_dirtied_per_call	Lokale Blockkontaminierungen pro Aufruf
db.sql_tokenized.stats.local_blks_written_per_call	Lokale Blockschreibvorgänge pro Aufruf
db.sql_tokenized.stats.temp_blks_written_per_call	Temporäre Blockschreibvorgänge pro Aufruf
db.sql_tokenized.stats.temp_blks_read_per_call	Temporäre Blocklesevorgänge pro Aufruf
db.sql_tokenized.stats.blk_read_time_per_call	Lesezeit pro Aufruf (in ms)
db.sql_tokenized.stats.blk_write_time_per_call	Schreibzeit pro Aufruf (in ms)

Primäre Statistiken für Aurora PostgreSQL

Die folgenden SQL-Statistiken sind für Aurora-PostgreSQL-DB-Instances verfügbar.

Metrik	Einheit
db.sql_tokenized.stats.calls	Calls
db.sql_tokenized.stats.rows	Rows
db.sql_tokenized.stats.total_time	Zeit insgesamt (in ms)
db.sql_tokenized.stats.shared_blks_hit	Blocktreffer
db.sql_tokenized.stats.shared_blks_read	Blocklesevorgänge
db.sql_tokenized.stats.shared_blks_dirtied	Blockkontaminierungen
db.sql_tokenized.stats.shared_blks_written	Blockschreibvorgänge
db.sql_tokenized.stats.local_blks_hit	Lokale Blocktreffer
db.sql_tokenized.stats.local_blks_read	Lokale Blocklesevorgänge
db.sql_tokenized.stats.local_blks_dirtied	Lokale Blockkontaminierungen
db.sql_tokenized.stats.local_blks_written	Lokale Blockschreibvorgänge
db.sql_tokenized.stats.temp_blks_written	Temporäre Schreibvorgänge
db.sql_tokenized.stats.temp_blks_read	Temporäre Lesevorgänge
db.sql_tokenized.stats.blk_read_time	Durchschnitt gleichzeitiger Lesevorgänge (in ms)
db.sql_tokenized.stats.blk_write_time	Durchschnitt gleichzeitiger Schreibvorgänge (in ms)

Weitere Informationen zu diesen Metriken finden Sie unter pg_stat_statements in der PostgreSQL-Dokumentation.

Betriebssystemmetriken in „Erweiterte Überwachung“

Amazon Aurora stellt in Echtzeit Metriken für das Betriebssystem (BS) bereit, auf dem Ihr DB-Cluster ausgeführt wird. Aurora übermittelt die Metriken von Enhanced Monitoring an Ihr Amazon CloudWatch Logs-Konto. In den folgenden Tabellen sind die Betriebssystemmetriken aufgeführt, die mit Amazon CloudWatch Logs verfügbar sind.

Themen

- Betriebssystemmetriken für Aurora

Betriebssystemmetriken für Aurora

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
`General`	`engine`	Nicht zutreffend	Die Datenbank-Engine für die DB-Instance.
	`instanceID`	Nicht zutreffend	Die DB-Instance-Kennung.
	`instanceResourceID`	Nicht zutreffend	Eine unveränderliche Kennung für die DB-Instance, die für eine AWS Region eindeutig ist und auch als Log-Stream-ID verwendet wird.
	`numVCPUs`	Nicht zutreffend	Die Anzahl der virtuellen Dateien CPUs für die DB-Instance.
	`timestamp`	Nicht zutreffend	Die Uhrzeit, zu der die Metriken erfasst wurden.
	`uptime`	Nicht zutreffend	Die Zeitdauer, für die die DB-Instance aktiv war.
	`version`	Nicht zutreffend	Die Version des JSON-Formats für den Stream der Betriebssystem-Metriken.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
`cpuUtilization`	`guest`	CPU Gast	Der prozentuale Anteil der von Gastprogrammen verwendeten CPU.
	`idle`	CPU Leerlauf	Der prozentuale Anteil der CPU, der sich im Leerlauf befindet.
	`irq`	CPU IRQ	Der prozentuale Anteil der von Software-Interrupts verwendeten CPU.
	`nice`	CPU Nice	Der prozentuale Anteil der von Programmen mit niedrigster Priorität verwendeten CPU.
	`steal`	CPU Steal	Der prozentuale Anteil der von virtuellen Maschinen verwendeten CPU.
	`system`	CPU-System	Der prozentuale Anteil der vom Kernel verwendeten CPU.
	`total`	CPU insgesamt	Der prozentuale Anteil der insgesamt verwendeten CPU. Diese Angabe enthält den Wert `nice`.
	`user`	CPU Benutzer	Der prozentuale Anteil der von Benutzerprogrammen verwendeten CPU.
	`wait`	CPU Warten	Der Prozentsatz der CPUs, die beim Warten auf den I/O Zugriff nicht genutzt wurden.
`diskIO`	`avgQueueLen`	Durchschn. Warteschlangengröße	Die Anzahl der Anfragen, die in der Warteschlange des I/O Geräts warten.
	`avgReqSz`	Durchschn. Anforderungsgröße	Die durchschnittliche Anfragegröße in Kilobyte.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
	`await`	Disk I/O Await	Die erforderliche Anzahl an Millisekunden für die Antwort auf Anfragen, einschließlich Warteschlangen- und Servicedauer.
	`device`	Nicht zutreffend	Die Kennung des verwendeten Datenträgers.
	`readIOsPS`	Gelesene IO/s	Die Anzahl der Lesevorgänge pro Sekunde.
	`readKb`	Insgesamt gelesen	Gesamtzahl der gelesenen Kilobyte.
	`readKbPS`	Gelesene KB/s	Die Anzahl der pro Sekunde gelesenen Kilobyte.
	`readLatency`	Lese-Latenz	Die verstrichene Zeit in Millisekunden zwischen dem Einreichen einer I/O Leseanforderung und deren Abschluss. Diese Metrik ist nur für Amazon Aurora verfügbar.
	`readThroughput`	Lesedurchsatz	Die Menge des Netzwerkdurchsatzes, der von Anforderungen an den DB-Cluster verwendet wird, in Bytes pro Sekunde. Diese Metrik ist nur für Amazon Aurora verfügbar.
	`rrqmPS`	Rrqms	Die Anzahl der zusammengeführten Leseanfragen in der Warteschlange pro Sekunde.
	`tps`	TPS	Die Anzahl der Transaktionen pro Sekunde. I/O
	`util`	Disk I/O Util	Der Prozentsatz der CPU-Zeit, während der Anfragen ausgegeben wurden.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
	`writeIOsPS`	Geschriebene IO/s	Die Anzahl der Schreibvorgänge pro Sekunde.
	`writeKb`	Insgesamt geschrieben	Gesamtzahl der geschriebenen Kilobyte.
	`writeKbPS`	Geschriebene KB/s	Die Anzahl der pro Sekunde geschriebenen Kilobyte.
	`writeLatency`	Schreib-Latenz	Die durchschnittliche Zeit in Millisekunden zwischen dem Einreichen einer I/O Schreibanforderung und deren Abschluss. Diese Metrik ist nur für Amazon Aurora verfügbar.
	`writeThroughput`	Schreibdurchsatz	Die Menge des Netzwerkdurchsatzes, der von Antworten vom DB-Cluster verwendet wird, in Bytes pro Sekunde. Diese Metrik ist nur für Amazon Aurora verfügbar.
	`wrqmPS`	Wrqms	Die Anzahl der zusammengeführten Schreibanfragen in der Warteschlange pro Sekunde.
`mountPoint`	Nicht zutreffend	Der Pfad zum Dateisystem.	
`fileSys`	`maxFiles`	Max Inodes	Die maximale Anzahl an Dateien, die für das Dateisystem erstellt werden können.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
	`mountPoint`	Nicht zutreffend	Der Pfad zum Dateisystem. Wenn dies der Fall ist `/rdsdbdata*` , stellt es die Summe aller Speichervolumes dar.
	`name`	Nicht zutreffend	Der Name des Dateisystems.
	`total`	Dateisystem insgesamt	Die Gesamtmenge des für das Dateisystem verfügbaren Speicherplatzes in Kilobyte.
	`used`	Verwendetes Dateisystem	Der durch Dateien belegte Speicherplatz im Dateisystem in Kilobyte.
	`usedFilePercent`	Gebrauchte Inodes	Der Prozentsatz von verfügbaren Dateien, die in Gebrauch sind.
	`usedFiles`	Used%	Die Anzahl der Dateien im Dateisystem.
	`usedPercent`	Verwendetes Dateisystem	Der prozentuale Anteil des verwendeten Speicherplatzes für das Dateisystem.
`loadAverageMinute`	`fifteen`	Last Durchschn. 15 Min	Die Anzahl der Prozesse, die während der letzten 15 Minuten CPU-Zeit angefordert haben.
	`five`	Laden Durchschn. 5 Min	Die Anzahl der Prozesse, die während der letzten 5 Minuten CPU-Zeit angefordert haben.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
	one	Laden Durchschn. 1 Min	Die Anzahl der Prozesse, die während der letzten Minute CPU-Zeit angefordert haben.
memory	active	Aktiver Speicher	Umfang des zugewiesenen Arbeitsspeichers in Kilobyte.
	buffers	Gepufferter Speicher	Die Speichermenge, die für die Pufferung von I/O Anforderungen vor dem Schreiben auf das Speichergerät verwendet wird, in Kilobyte.
	cached	Zwischenspeicher	Umfang des verwendeten Arbeitsspeichers für Caching von Dateisystem-basierter I/O.
	dirty	Geänderter Speicher	Menge an Memory-Pages im RAM, die geändert, aber noch nicht in den entsprechenden Datenblock im Speicher geschrieben wurden, in Kilobyte.
	free	Freier Speicher	Umfang des nicht zugewiesenen Arbeitsspeichers in Kilobyte.
	hugePagesFree	Huge Pages frei	Die Anzahl von freien "Huge Pages". "Huge Pages" sind eine Funktion des Linux-Kernel.
	hugePagesRsvd	Huge Pages reserviert	Die Anzahl von gebundenen "Huge Pages".
	hugePagesSize	Größe Huge Pages	Die Größe für jede Huge Pages-Einheit, in Kilobyte.
	hugePagesSurp	Überschuss Huge Pages	Die Anzahl der verfügbaren überzähligen Huge Pages.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
	`hugePagesTotal`	Huge Pages insgesamt	Die Gesamtzahl der Huge Pages.
	`inactive`	Inaktiver Speicher	Umfang der am seltensten verwendeten Memory-Pages in Kilobyte.
	`mapped`	Zugeordneter Speicher	Die Gesamtmenge der Dateisysteminhalte, die Arbeitsspeicher in einem Prozess-Adressraum zugeordnet ist, in Kilobyte.
	`pageTables`	Seitentabellen	Umfang des von Page-Tabellen verwendeten Arbeitsspeichers in Kilobyte.
	`slab`	Plattenspeicher	Umfang der wiederverwendbaren Kernel-Datenstrukturen in Kilobyte.
	`total`	Gesamtspeicher	Gesamtumfang des Arbeitsspeichers in Kilobyte.
	`writeback`	Writeback-Speicher	Gesamtumfang an modifizierten Speicherbereichen im RAM, die noch in den Sicherungsspeicher geschrieben werden, in Kilobyte.
`network`	`interface`	Nicht zutreffend	Die Kennung für die Netzwerkschnittstelle, die für die DB-Instance verwendet wird.
	`rx`	RX	Die Anzahl der pro Sekunde empfangenen Byte.
	`tx`	TX	Die Anzahl der pro Sekunde hochgeladenen Byte.
`processList`	`cpuUsedPc`	% CPU	Prozentsatz der vom Prozess benutzten CPU.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
	`id`	Nicht zutreffend	Die ID des Prozesses.
	`memoryUsedPc`	RAM %	Prozentsatz des insgesamt vom Prozess benutzten Speichers.
	`name`	Nicht zutreffend	Der Name des Prozesses.
	`parentID`	Nicht zutreffend	Die Prozess-ID für den übergeordneten Prozess des aktuellen Prozesses.
	`rss`	RES	Umfang des dem Prozess zugeteilten RAM in Kilobyte.
	`tgid`	Nicht zutreffend	Die Thread-Gruppen-ID als Zahl, die die Prozess-ID darstellt, zu der ein Thread gehört. Diese Kennung wird verwendet, um Threads aus demselben Prozess zu gruppieren.
	`vss`	VIRT	Umfang des dem Prozess zugeteilten virtuellen Speichers in Kilobyte.
`swap`	`total`	Auslagerung	Die Menge des verfügbaren Swap-Arbeitsspeichers in Kilobyte.
	`in`	Auslagerungen in	Die Menge des von der Festplatte ausgelagerten Speichers in Kilobyte.
	`out`	Auslagerungen aus	Die Menge des auf die Festplatte ausgelagerten Speichers in Kilobyte.
	`free`	Kostenlose Auslagerung	Die Menge des freien Swap-Arbeitsspeichers in Kilobyte.

Group (Gruppieren)	Metrik	Name der Konsole	Beschreibung
	`cached`	Committed Auslagerung	Umfang des Swap-Arbeitsspeichers, der als Cache-Speicher verwendet wird, in Kilobyte.
`tasks`	`blocked`	Gesperrte Aufgaben	Die Anzahl von blockierten Aufgaben.
	`running`	Laufende Aufgaben	Die Anzahl von laufenden Aufgaben.
	`sleeping`	Ruhende Aufgaben	Die Anzahl von Aufgaben im Ruhezustand.
	`stopped`	Gestoppte Aufgaben	Die Anzahl von angehaltenen Aufgaben.
	`total`	Aufgaben insgesamt	Die Gesamtanzahl der Aufgaben.
	`zombie`	Aufgaben Zombie	Die Anzahl der untergeordneten Aufgaben, die unter einer aktiven übergeordneten Aufgabe inaktiv sind.

Überwachen von Ereignissen, Protokollen und Streams in einem Amazon Aurora-DB-Cluster

Wenn Sie die Amazon Aurora-Datenbanken und die anderen AWS-Lösungen überwachen, ist Ihr Ziel, Folgendes beizubehalten:

- Zuverlässigkeit
- Verfügbarkeit
- Leistung
- Sicherheit

Überwachung von Metriken in einem Amazon-Aurora-Cluster erklärt, wie Sie den Cluster mithilfe von Metriken überwachen. Eine Komplettlösung muss auch Datenbankereignisse, Protokolldateien und Aktivitäts-Streams überwachen. AWS stellt Ihnen die folgenden Überwachungstools zur Verfügung:

- Amazon EventBridge ist ein Serverless-Ereignisbus-Service, über den Sie Ihre Anwendungen einfach mit Daten aus einer Vielzahl von Quellen verbinden können. EventBridge stellt einen Stream von Echtzeitdaten aus Ihren eigenen Anwendungen, Software as a Service (SaaS)-Anwendungen und AWS-Services bereit. EventBridge leitet diese Daten an Ziele wie AWS Lambda weiter. Auf diese Weise können Sie Ereignisse überwachen, die in Services auftreten, und ereignisgesteuerte Architekturen erstellen. Weitere Informationen finden Sie im Benutzerhandbuch für Amazon EventBridge.
- Mit Amazon CloudWatch Logs können Sie Ihre Protokolldateien von Amazon-Aurora-Instances, AWS CloudTrail und anderen Quellen überwachen, speichern und darauf zugreifen. Amazon CloudWatch Logs kann Informationen in den Protokolldateien überwachen und Sie benachrichtigen, wenn bestimmte Schwellenwerte erreicht werden. Sie können Ihre Protokolldaten auch in einem sehr robusten Speicher archivieren. Weitere Informationen finden Sie im Amazon CloudWatch Logs User Guide.
- AWS CloudTrail erfasst API-Aufrufe und zugehörige Ereignisse, die von oder im Namen Ihrer AWS-Konto. CloudTrail sendet die Protokolldateien an einen Amazon-S3-Bucket, den Sie zuvor angegeben haben. Sie können die Benutzer und Konten, die AWS aufgerufen haben, identifizieren, sowie die Quell-IP-Adresse, von der diese Aufrufe stammen, und den Zeitpunkt der Aufrufe ermitteln. Weitere Informationen finden Sie im AWS CloudTrail-Benutzerhandbuch.

- Datenbankaktivitätsstreams ist eine Amazon-Aurora-Funktion, die einen Beinahe-Echtzeitdatenstrom der Aktivität in Ihrem DB-Cluster. bietet. Amazon Aurora sendet Aktivitäten per Push an einen Amazon Kinesis Data Stream. Der Kinesis Stream wird automatisch erstellt. In Kinesis können Sie AWS-Services wie Amazon Data Firehose und AWS Lambda konfigurieren, um den Stream zu verbrauchen und die Daten zu speichern.

Themen

- Anzeigen von Protokollen, Ereignissen und Streams in der Amazon-RDS-Konsole
- Überwachung von Amazon Aurora-Ereignissen
- Überwachen von Amazon Aurora-Protokolldateien
- Überwachung von Amazon Aurora-API-Aufrufen in AWS CloudTrail
- Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams
- Überwachen von Bedrohungen mit Amazon GuardDuty RDS Protection für Amazon Aurora

Anzeigen von Protokollen, Ereignissen und Streams in der Amazon-RDS-Konsole

Amazon RDS lässt sich in AWS-Services integrieren, um Informationen zu Protokollen, Ereignissen und Datenbankaktivitäts-Streams in der RDS-Konsole anzuzeigen.

Die Registerkarte Logs·&·events (Protokolle und Ereignisse) für den Aurora-DB-Cluster zeigt die folgenden Informationen an:

- Auto-Scaling-Richtlinien und -Aktivitäten – Zeigt Richtlinien und Aktivitäten in Bezug auf die Aurora-Auto-Scaling-Funktion an. Diese Informationen werden nur auf der Registerkarte Logs·&·events (Protokolle und Ereignisse) auf Cluster-Ebene angezeigt.
- Amazon-CloudWatch-Alarme – Zeigt alle metrischen Alarme, die Sie für die DB-Instance konfiguriert haben, in Ihrem Aurora-Cluster an. Wenn Sie keine Alarme konfiguriert haben, können Sie sie in der RDS-Konsole erstellen.
- Aktuelle Ereignisse – Zeigt eine Übersicht über Ereignisse (Umgebungsänderungen) für Ihre Aurora--DB-Instance oder Cluster an. Weitere Informationen finden Sie unter Anzeigen von Amazon-RDS-Ereignissen.

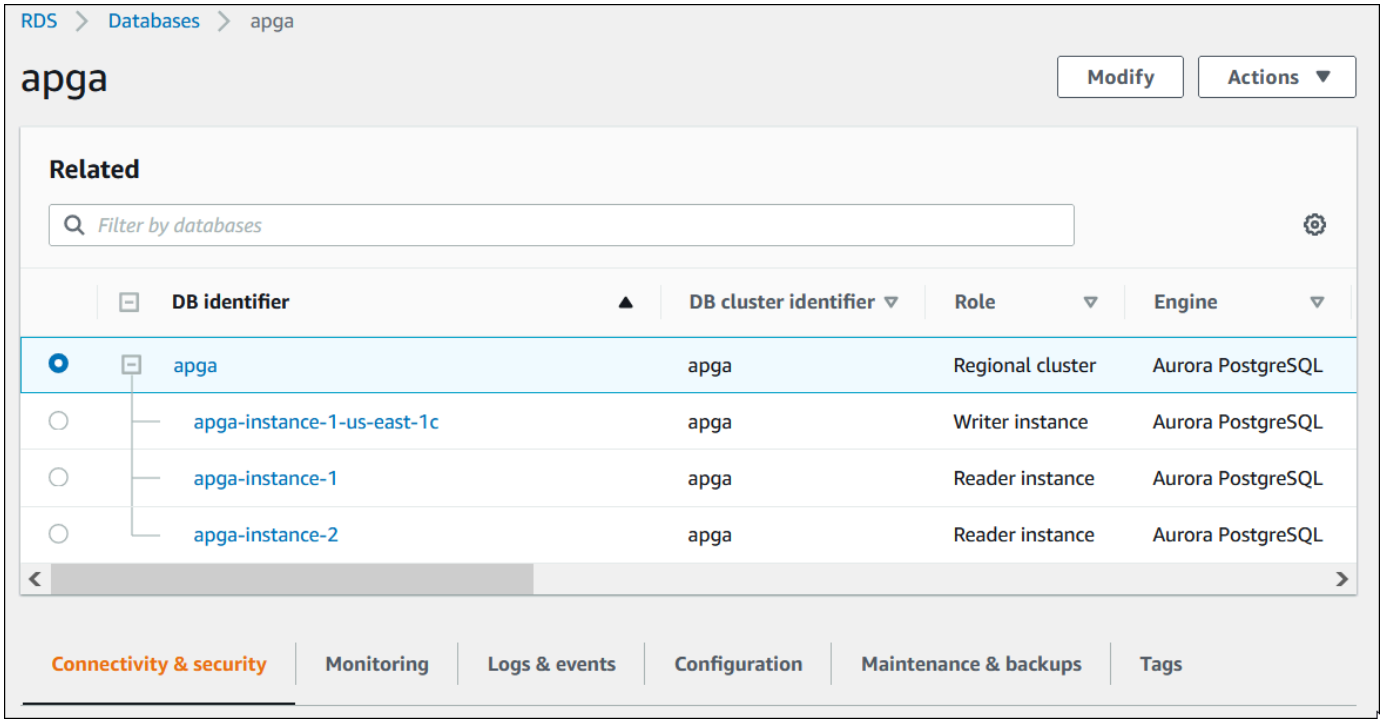
- Protokolle – Zeigt Datenbankprotokolldateien an, die von einer DB-Instance in Ihrem Aurora-Cluster generiert wurden. Weitere Informationen finden Sie unter Überwachen von Amazon Aurora-Protokolldateien.

Die Registerkarte Konfiguration zeigt Informationen über Datenbankaktivitäts-Streams an.

Zeigen Sie Protokolle, Ereignisse und Streams für Ihre Aurora-DB--Cluster in der RDS-Konsole wie folgt an:

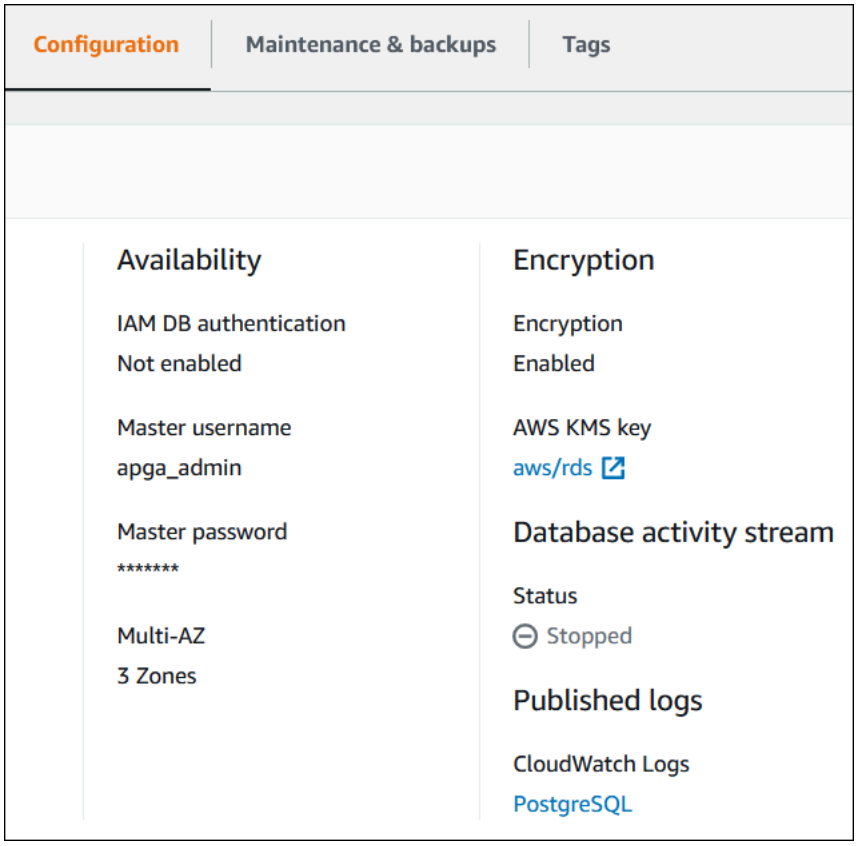
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Namen des Aurora-DB--Cluster an, das Sie überwachen möchten.

   Der Bereich Datenbanken wird angezeigt. Das folgende Beispiel zeigt eine Amazon-Aurora-PostgreSQL-DB-Cluster namens apga an.



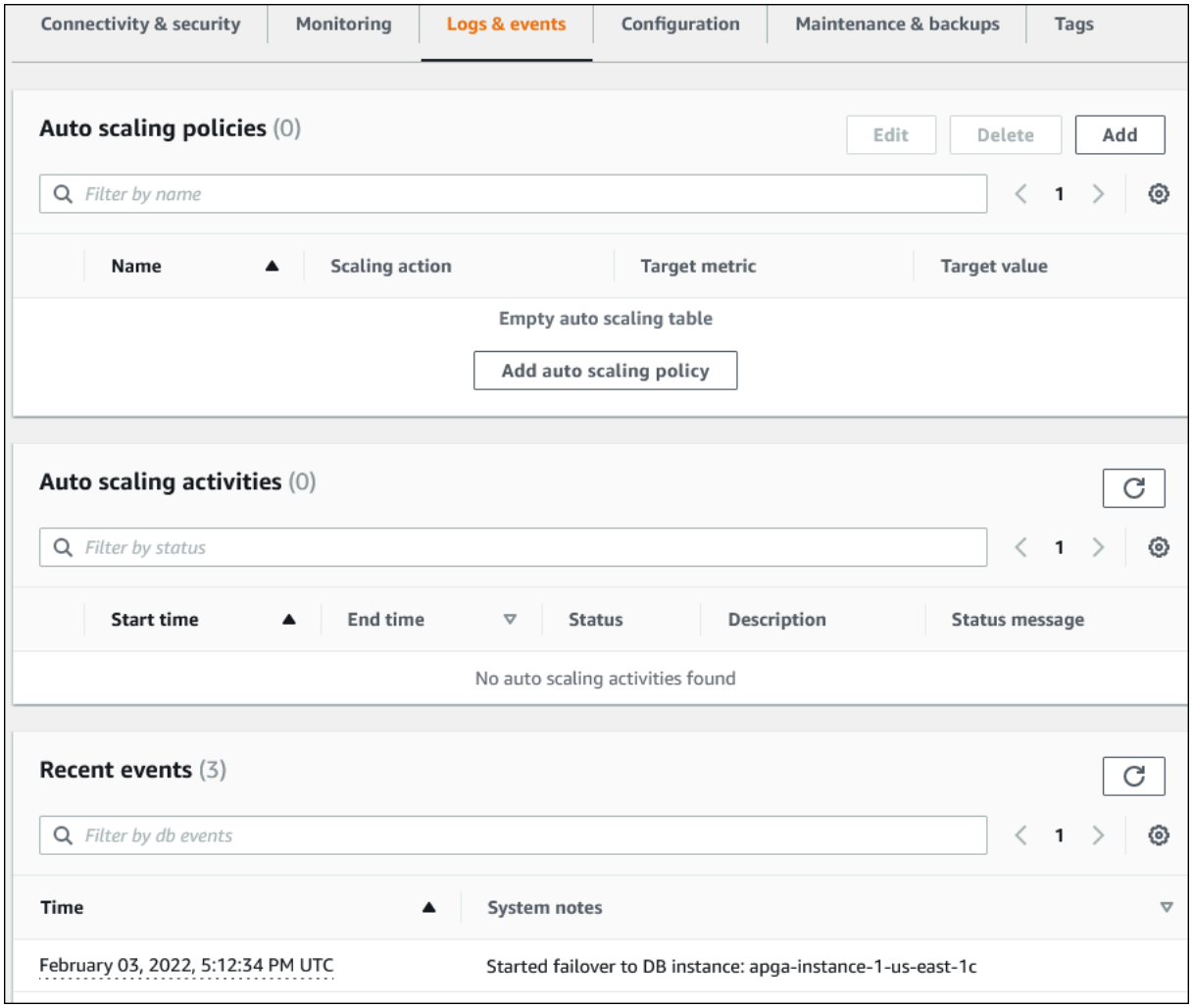
4. Scrollen Sie nach unten und wählen Sie Konfiguration aus.

   Das folgende Beispiel zeigt den Status der Datenbankaktivitäts-Streams für Ihren Cluster an.



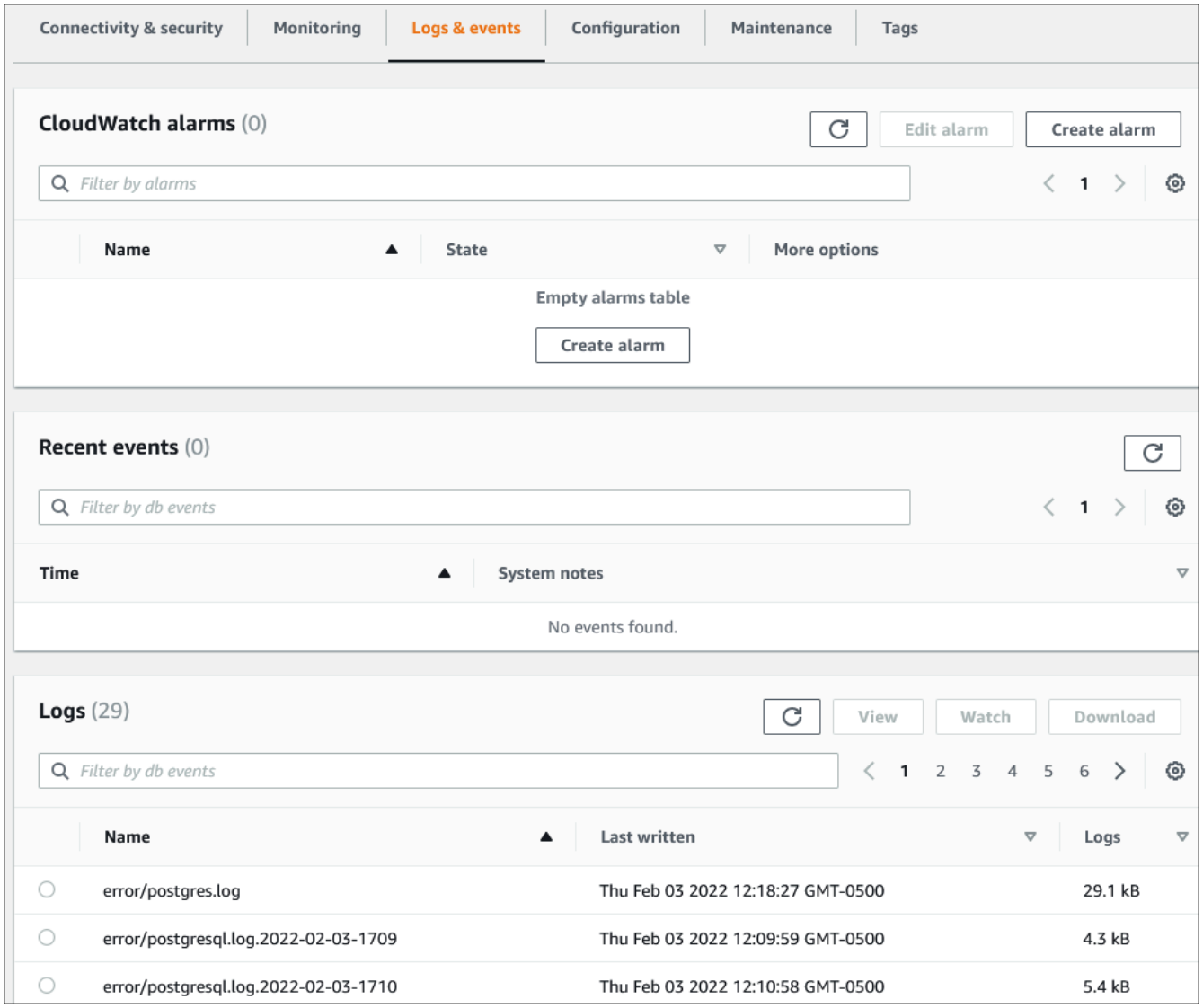
5. Wählen Sie Logs & Events (Protokolle und Ereignisse).

   Der Abschnitt Logs & Events (Protokolle und Ereignisse) wird angezeigt.



6. Wählen Sie eine DB-Instance in Ihrem Aurora-Cluster und dann Logs & Events (Protokolle & Ereignisse) für die Instance.

   Das folgende Beispiel zeigt, dass der Inhalt zwischen der DB-Instance-Seite und der DB-Cluster-Seite unterschiedlich ist. Auf der Seite der DB-Instance werden Protokolle und Alarme angezeigt.


Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance
Tags
CloudWatch alarms (0)
Edit alarm
Create alarm
Filter by alarms
1
Name
State
More options
Empty alarms table
Create alarm
Recent events (0)
Filter by db events
1
Time
System notes
No events found.
Logs (29)
View
Watch
Download
Filter by db events
1 2 3 4 5 6
Name
Last written
Logs
error/postgres.log
Thu Feb 03 2022 12:18:27 GMT-0500
29.1 kB
error/postgresql.log.2022-02-03-1709
Thu Feb 03 2022 12:09:59 GMT-0500
4.3 kB
error/postgresql.log.2022-02-03-1710
Thu Feb 03 2022 12:10:58 GMT-0500
5.4 kB

Überwachung von Amazon Aurora-Ereignissen

Ein Ereignis weist auf eine Änderung in einer Umgebung hin. Dabei kann es sich um eine AWS-Umgebung, SaaS-Partnerservices oder -Anwendungen oder Ihre eigenen benutzerdefinierten Anwendungen oder Services handeln. Beschreibungen der Aurora-Ereignisse finden Sie unter Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora.

Themen

- Überblick über Ereignisse für Aurora
- Anzeigen von Amazon-RDS-Ereignissen
- Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen
- Erstellen einer Regel, die bei einem Amazon Aurora-Ereignis ausgelöst wird
- Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora

Überblick über Ereignisse für Aurora

Ein RDS event (RDS-Ereignis) weist auf eine Änderung der AuroraUmgebung hin. Beispielsweise generiert Amazon Aurora ein Ereignis, wenn ein DB-Cluster gepatcht wird ausgeführt ändert. Amazon Aurora liefert Ereignisse in nahezu Echtzeit an EventBridge.

> **Note**
>
> Amazon RDS sendet Ereignisse nach bestem Bemühen aus. Wir empfehlen Ihnen, keine Programme zu schreiben, die von der Reihenfolge oder dem Vorhandensein von Benachrichtigungsereignissen abhängen, da diese möglicherweise nicht in der richtigen Reihenfolge vorliegen oder fehlen.

Amazon RDS zeichnet Ereignisse auf, die sich auf die folgenden Ressourcen beziehen:

- DB-Cluster

  Eine Liste der Cluster-Ereignisse finden Sie unter DB-Cluster-Ereignisse.
- DB-Instances

  Eine Liste der DB-Instance-Ereignisse finden Sie unter DB-Instance-Ereignisse.
- DB-Parametergruppen

Eine Liste der Ereignisse der DB-Parametergruppe finden Sie unter DB-Parametergruppenereignisse.

- DB-Sicherheitsgruppen

  Eine Liste der Ereignisse der DB-Sicherheitsgruppe finden Sie unter DB-Sicherheitsgruppenereignisse.

- DB-Cluster-Snapshots

  Eine Liste der DB-Cluster-Snapshot-Ereignisse finden Sie unter DB-Cluster-Snapshot-Ereignisse.

- RDS-Proxy-Ereignisse

  Eine Liste der RDS-Proxy-Ereignisse finden Sie unter RDS-Proxy-Ereignisse.

- Blau/Grün-Bereitstellungsereignisse

  Eine Liste der Blau/Grün-Bereitstellungsereignisse finden Sie unter Blau/Grün-Bereitstellungsereignisse.

Diese Informationen beinhalten Folgendes:

- Das Datum und die Uhrzeit der Veranstaltung
- Der Quellname und der Quelltyp des Ereignisses
- Eine Nachricht, die mit dem Ereignis verknüpft ist
- Ereignisbenachrichtigungen enthalten Tags zum Zeitpunkt, an dem die Nachricht gesendet wurde, und geben möglicherweise nicht die Tags zum Zeitpunkt des Eintritts des Ereignisses wieder.

Anzeigen von Amazon-RDS-Ereignissen

Sie können die folgenden Ereignisinformationen für Ihre Amazon Aurora-Ressourcen abrufen:

- Ressourcenname
- Ressourcentyp
- Zeitpunkt des Ereignisses
- Zusammenfassung der Ereignisbenachrichtigung

Sie können in den folgenden Bereichen der AWS-Managementkonsole auf Ereignisse zugreifen:

- Die Registerkarte Ereignisse, auf der Ereignisse der letzten 24 Stunden angezeigt werden.
- Die Tabelle Aktuelle Ereignisse im Abschnitt Protokolle und Ereignisse auf der Registerkarte Datenbanken, auf der Ereignisse der letzten 2 Wochen angezeigt werden können.

Sie können Ereignisse auch abrufen, indem Sie den AWS CLI-Befehl describe-events oder die RDS-API-Operation DescribeEvents verwenden. Wenn Sie AWS CLI oder RDS-API verwenden, um Ereignisse anzusehen, können Sie Ereignisse aus den letzten 14 Tagen abrufen.

> **Note**
>
> Wenn Sie Ereignisse länger speichern müssen, können Sie Amazon-RDS-Ereignisse an EventBridge senden. Weitere Informationen finden Sie unter Erstellen einer Regel, die bei einem Amazon Aurora-Ereignis ausgelöst wird.

Beschreibungen der Amazon-Aurora-Ereignisse finden Sie unter Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora.

Informationen dazu, wie Sie auf detaillierte Informationen zu Ereignissen mithilfe von AWS CloudTrail, einschließlich Anforderungsparameter, zugreifen, finden Sie unter CloudTrail-Ereignisse.

Konsole

So können Sie alle Amazon-RDS-Ereignisse der letzten 24 Stunden ansehen

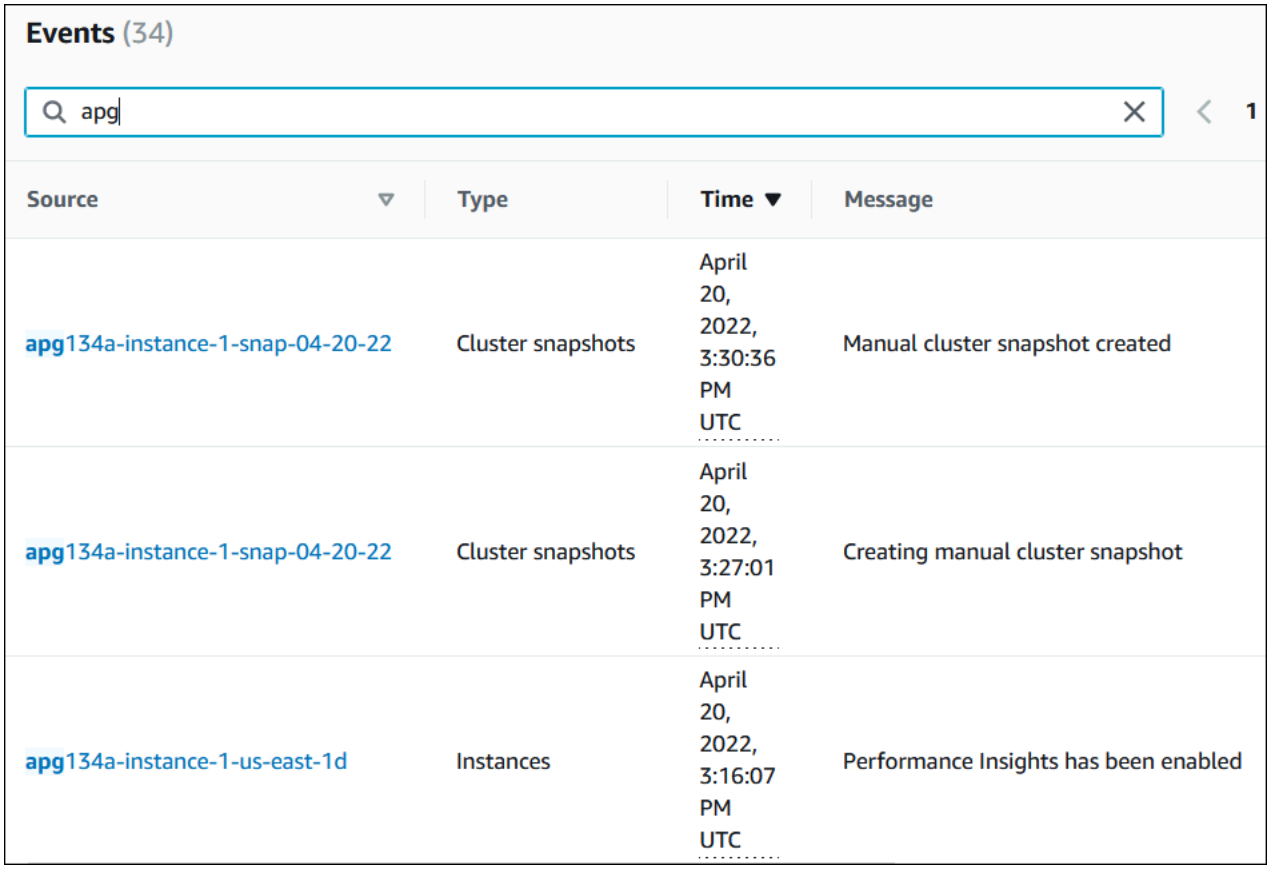
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich die Option Events.

   Die verfügbaren Ereignisse erscheinen in einer Liste.

3. (Optional) Geben Sie einen Suchbegriff ein, um Ihre Ergebnisse zu filtern.

   Das folgende Beispiel zeigt eine Liste von Ereignissen, die nach den Zeichen **apg** gefiltert sind.



AWS CLI

Wenn Sie alle Ereignisse anzeigen möchten, die in der letzten Stunde generiert wurden, rufen Sie describe-events ohne Parameter auf.

```
aws rds describe-events
```

Die folgende Beispielausgabe zeigt, dass eine DB-Cluster-Instance gestoppt wurde.

```
{
    "Events": [
        {
            "EventCategories": [
```

```
                "recovery"
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
            "Date": "2022-04-20T15:02:38.416Z",
            "Message": "Recovery of the DB instance has started. Recovery time will
 vary with the amount of data to be recovered.",
            "SourceIdentifier": "mycluster-instance-1"
        }, ...
```

Sie können alle Amazon-RDS-Ereignisse der letzten 10.080 Minuten (7 Tage) ansehen, indem Sie den AWS CLI-Befehl describe-events aufrufen und den Parameter `--duration` auf `10080` festlegen.

```
aws rds describe-events --duration 10080
```

Das folgende Beispiel zeigt die Ereignisse im angegebenen Zeitraum für die DB-Instance *`test-instance`*.

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

Die folgende Beispielausgabe zeigt den Status eines Backups.

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
```

```
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        }
    ]
}
```

API

Sie können alle Amazon-RDS-Instance-Ereignisse der letzten 14 Tage anzeigen, indem Sie die RDS-API-Operation DescribeEvents aufrufen und den Parameter `Duration` auf `20160` festlegen.

Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen

Amazon RDS verwendet Amazon Simple Notification Service (Amazon SNS), um Benachrichtigungen zu senden, wenn ein Amazon-RDS-Ereignis stattfindet. Diese Benachrichtigungen können jedes von Amazon SNS für eine AWS-Region unterstützte Format aufweisen, wie zum Beispiel eine E-Mail, eine SMS oder ein Anruf an einen HTTP-Endpunkt.

Themen

- Überblick über die Amazon-RDS-Ereignisbenachrichtigung
- Erteilen von Berechtigungen zum Veröffentlichen von Benachrichtigungen in einem Amazon-SNS-Thema
- Abonnieren von Amazon-RDS-Ereignisbenachrichtigungen
- Tags und Attribute von Amazon-RDS-Ereignisbenachrichtigungen
- Auflisten von Abonnements für Amazon-RDS-Ereignisbenachrichtigungen
- Ändern eines Abonnements für Amazon-RDS-Ereignisbenachrichtigungen
- Hinzufügen einer Quell-ID zu einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen
- Entfernen einer Quell-ID aus einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen
- Auflisten von Kategorien für Amazon-RDS-Ereignisbenachrichtigungen
- Löschen eines Abonnements für Amazon-RDS-Ereignisbenachrichtigungen

Überblick über die Amazon-RDS-Ereignisbenachrichtigung

Amazon RDS gruppiert Ereignisse in Kategorien, die Sie abonnieren können, damit Sie benachrichtigt werden, wenn ein Ereignis in dieser Kategorie eintritt.

Themen

- RDS-Ressourcen, die für ein Ereignisabonnement in Frage kommen
- Grundlegendes Verfahren zum Abonnieren von Amazon-RDS-Ereignisbenachrichtigungen
- Zustellung von RDS-Ereignisbenachrichtigungen
- Fakturierung für Amazon-RDS-Ereignisbenachrichtigungen
- Beispiele für Aurora-Ereignisse, die Amazon EventBridge verwenden

RDS-Ressourcen, die für ein Ereignisabonnement in Frage kommen

Bei Amazon Aurora treten Ereignisse sowohl auf Ebene des DB-Clusters als auch auf Ebene der DB-Instance auf. Sie können eine Veranstaltungskategorie für die folgenden Ressourcen abonnieren:

- DB-Instance
- DB-Cluster
- DB-Cluster-Snapshot
- DB-Parametergruppe
- DB-Sicherheitsgruppe
- RDS-Proxy
- Kundenspezifische Motorversionen

Wenn Sie zum Beispiel die Backup-Kategorie für eine bestimmte DB-Instance abonnieren, werden Sie immer dann benachrichtigt, wenn ein Backup-bezogenes Ereignis eintritt, das die DB-Instance betrifft. Wenn Sie eine Konfigurationsänderungskategorie für eine DB-Instance abonnieren, werden Sie benachrichtigt, sobald die DB-Instance geändert wird. Außerdem erhalten Sie eine Benachrichtigung, wenn ein Abonnement für Ereignisbenachrichtigungen geändert wird.

Möglicherweise möchten Sie mehrere verschiedene Abonnements erstellen. Sie könnten beispielsweise ein Abonnement erstellen, das alle Ereignisbenachrichtigungen für alle DB-Instances empfängt, und ein anderes, das nur kritische Ereignisse für eine Teilmenge der DB-Instances enthält. Geben Sie für das zweite Abonnement eine oder mehrere DB-Instances im Filter an.

Grundlegendes Verfahren zum Abonnieren von Amazon-RDS-Ereignisbenachrichtigungen

Gehen Sie wie folgt vor, um Amazon-RDS-Ereignisbenachrichtigungen zu abonnieren:

1. Sie erstellen ein Amazon-RDS-Ereignisbenachrichtigungs-Abonnement über die Amazon-RDS-Konsole, AWS CLI oder API.

   Amazon RDS verwendet den ARN eines Amazon SNS-Themas, um die einzelnen Abonnements zu ermitteln. Die Amazon-RDS-Konsole erstellt einen ARN für Sie, wenn Sie ein Abonnement erstellen. Erstellen Sie den ARN mit Hilfe der Amazon SNS-Konsole, der AWS CLI oder der Amazon SNS-API.
2. Amazon RDS sendet eine Bestätigungs-E-Mail oder SMS-Nachricht an die Adressen, die Sie mit Ihrem Abonnement übermittelt haben.

3. Klicken Sie auf den Link in der erhaltenen Benachrichtigung, um das Abonnement zu bestätigen.
4. Die Amazon-RDS-Konsole aktualisiert den Abschnitt My Event Subscriptions (Meine Ereignisabonnements) mit dem Status Ihres Abonnements.
5. Amazon RDS sendet Benachrichtigungen an die Adressen, die Sie beim Erstellen des Abonnements angegeben haben.

Informationen über Identity and Access Management bei Verwendung von Amazon SNS finden Sie unterIdentity and Access Management in Amazon SNSimAmazon Simple Notification Service-Entwicklerhandbuchaus.

Sie können AWS Lambda verwenden, um Ereignisbenachrichtigungen von einer DB-Instance zu verarbeiten. Weitere Informationen finden Sie unter Verwenden von AWS Lambda mit Amazon RDS im AWS Lambda-Entwicklerhandbuch.

Zustellung von RDS-Ereignisbenachrichtigungen

Amazon RDS sendet Benachrichtigungen an die Adressen, die Sie beim Erstellen des Abonnements angeben. Die Benachrichtigung kann Nachrichtenattribute mit einschließen, die strukturierte Metadaten zu der Nachricht zur Verfügung stellen. Weitere Informationen über Nachrichtenattribute finden Sie unter Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora.

Es kann bis zu fünf Minuten dauern, bis Ereignisbenachrichtigungen zugestellt werden.

> **Important**
>
> Amazon RDS garantiert nicht die Reihenfolge der Ereignisse, die in einem Ereignisstrom gesendet werden. Die Reihenfolge der Ereignisse kann sich ändern.

Wenn Amazon SNS eine Benachrichtigung an einen abonnierten HTTP- oder HTTPS-Endpunkt sendet, enthält der Nachrichtentext der POST-Nachricht, die an den Endpunkt gesendet wurde, ein JSON-Dokument. Weitere Informationen finden Sie unter Amazon SNS-Nachrichten- und -JSON-Formate im Amazon Simple Notification Service-Entwicklerhandbuch.

Sie können SNS so konfigurieren, dass Sie mit Textnachrichten benachrichtigt werden. Weitere Informationen finden Sie unter Mobile Textnachrichten (SMS) im Amazon Simple Notification Service Developer Guide.

Um Benachrichtigungen zu deaktivieren, ohne ein Abonnement zu löschen, wählen SieNeinfürEnabledin der Amazon-RDS-Konsole. Oder Sie können die`Enabled`-Parameter auf`false`Verwendung vonAWS CLIoder Amazon-RDS-API.

Fakturierung für Amazon-RDS-Ereignisbenachrichtigungen

Die Fakturierung für Amazon-RDS-Ereignisbenachrichtigungen erfolgt über Amazon SNS. Bei Verwendung von Ereignisbenachrichtigungen fallen Amazon-SNS-Gebühren an. Weitere Informationen zur Abrechnung von Amazon SNS finden Sie unter Preise für Amazon Simple Notification Service.

Beispiele für Aurora-Ereignisse, die Amazon EventBridge verwenden

Die folgenden Beispiele veranschaulichen verschiedene Arten von Aurora-Ereignissen im JSON-Format. Ein Tutorial, das veranschaulicht, wie Sie Ereignisse im JSON-Format erfassen und anzeigen, finden Sie unter Tutorial: Protokollieren von Statusänderungen an einer DB-Instance mit Amazon EventBridge.

Themen

- Beispiel für ein DB-Cluster-Ereignis
- Beispiel für ein Ereignis der DB-Parametergruppe
- Beispiel für ein DB-Cluster-Snapshot-Ereignis

Beispiel für ein DB-Cluster-Ereignis

Es folgt das Beispiel eines DB-Cluster-Ereignisses im JSON-Format. Das Ereignis zeigt, dass der Cluster mit dem Namen `my-db-cluster` gepatcht wurde. Die Ereignis-ID ist `RDS-EVENT-0173`.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"
  ],
```

```
  "detail": {
    "EventCategories": [
      "notification"
    ],
    "SourceType": "CLUSTER",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Database cluster has been patched",
    "SourceIdentifier": "my-db-cluster",
    "EventID": "RDS-EVENT-0173"
  }
}
```

Beispiel für ein Ereignis der DB-Parametergruppe

Der folgende Code ist ein Beispiel für ein DB-Parametergruppenereignis im JSON-Format. Das Ereignis zeigt, dass der Parameter `time_zone` in der Parametergruppe `my-db-param-group` aktualisiert wurde. Die Ereignis-ID lautet RDS-EVENT-0037.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

Beispiel für ein DB-Cluster-Snapshot-Ereignis

Es folgt das Beispiel eines DB-Cluster-Snapshot-Ereignisses im JSON-Forma. Das Ereignis zeigt das Erstellen des Snapshots mit dem Namen `my-db-cluster-snapshot`. Die Ereignis-ID lautet RDS-EVENT-0074.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "backup"
    ],
    "SourceType": "CLUSTER_SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-
cluster-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "my-db-cluster-snapshot",
    "Message": "Creating manual cluster snapshot",
    "EventID": "RDS-EVENT-0074"
  }
}
```

Erteilen von Berechtigungen zum Veröffentlichen von Benachrichtigungen in einem Amazon-SNS-Thema

Um Amazon RDS Berechtigungen zur Veröffentlichung von Benachrichtigungen zu einem Amazon Simple Notification Service (Amazon SNS) -Thema zu gewähren, fügen Sie dem Zielthema eine AWS Identity and Access Management (IAM) -Richtlinie hinzu. Weitere Informationen zu Berechtigungen finden Sie unter Beispielfälle für die Zugriffskontrolle von Amazon Simple Notification Service im Entwicklerhandbuch für Amazon Simple Notification .

Standardmäßig verfügt ein Amazon-SNS-Thema über eine Richtlinie, die es allen Amazon-RDS-Ressourcen innerhalb desselben Kontos ermöglicht, Benachrichtigungen darin zu veröffentlichen. Sie können eine benutzerdefinierte Richtlinie anfügen, um kontoübergreifende Benachrichtigungen zuzulassen oder den Zugriff auf bestimmte Ressourcen einzuschränken.

Im Folgenden finden Sie ein Beispiel für eine IAM-Richtlinie, die Sie dem Amazon-SNS-Zielthema anfügen. Sie beschränkt das Thema auf DB-Instances mit Namen, die dem angegebenen Präfix entsprechen. Geben Sie die folgenden Werte an, um diese Richtlinie zu verwenden:

- `Resource` – Den Amazon-Ressourcennamen (ARN) für das Amazon-SNS-Thema
- `SourceARN` – Ihren RDS-Ressourcen-ARN
- `SourceAccount`— Ihre ID AWS-Konto

Eine Liste der Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter Von Amazon RDS definierte Ressourcen in der Service Authorization Reference.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
```

```
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Abonnieren von Amazon-RDS-Ereignisbenachrichtigungen

Am einfachsten lässt sich ein Abonnement mit der RDS-Konsole erstellen. Wenn Sie Abonnements für Ereignisbenachrichtigungen mithilfe der CLI oder API erstellen möchten, müssen Sie ein Amazon Simple Notification Service-Thema erstellen und dieses Thema über die Amazon SNS-Konsole oder Amazon SNS-API abonnieren. Sie müssen sich auch den Amazon-Ressourcennamen (ARN) des Themas notieren, da dieser beim Übermitteln von CLI-Befehlen oder API-Operationen verwendet wird. Weitere Informationen zum Erstellen und Abonnieren eines SNS-Themas finden Sie unter Erste Schritte mit Amazon SNS im Amazon Simple Notification Service-Entwicklerhandbuch.

Sie können den Quelltyp festlegen, zu dem Sie Benachrichtigungen erhalten möchten, sowie die die Amazon-RDS-Quelle, die das Ereignis auslöst:

Source type (Quellentyp)

Der Quelltyp. Beispiel: Source type (Quellentyp) könnte Instances sein. Sie müssen einen Quellentyp auswählen.

Resources to include (Einzuschließende Ressourcen)

Die Amazon-RDS-Ressourcen, die die Ereignisse generieren. Sie könnten beispielsweise Select specific instances (Spezifische Instances auswählen) und dannmyDBInstance1 auswählen.

In der folgenden Tabelle wird das Ergebnis erläutert für den Fall, dass Sie ***Resources*** to include (Einzuschließende Ressourcen) auswählen bzw. nicht auswählen.

Einzuschließende Ressourcen	Beschreibung	Beispiel
Angegeben	RDS benachrichtigt Sie nur über alle Ereignisse für die angegebene Ressource.	Wenn der Source type (Quellentyp) Instances lautet und Ihre Ressource myDBInstance1 ist, informiert Sie RDS nur über alle Ereignisse für myDBInstance1.
Nicht angegeben	Sie erhalten eine Benachrichtigung von RDS über die Ereignisse für den	Wenn Ihr Source type (Quellentyp) Instances lautet, informiert Sie RDS über alle Ereignisse in

Einzuschl ießende Ressourcen	Beschreibung	Beispiel
	angegebenen Quellentyp für alle Ihre Amazon-RDS-Ressourcen.	Verbindung mit der Instance in Ihrem Konto.

Standardmäßig erhalten Abonnenten von Amazon-SNS-Themen jede Nachricht, die zum Thema veröffentlicht wird. Um eine Teilmenge der Nachrichten zu erhalten, müssen Abonnenten dem Themen-Abonnement eine Filterrichtlinie zuweisen. Weitere Informationen zur SNS-Nachrichtenfilterung finden Sie unter Amazon-SNS-Nachrichtenfilterung im Entwicklerhandbuch zu Amazon Simple Notification Service.

Konsole

So abonnieren Sie RDS-Ereignisbenachrichtigungen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Event subscriptions (Ereignisabonnements).
3. Wählen Sie im Bereich Ereignisabonnements Ereignisabonnement erstellen aus.
4. Geben Sie Ihre Abonnementdetails wie folgt ein:

    a. Geben Sie unter Name einen Namen für das Abonnement für Ereignisbenachrichtigungen ein.

    b. Führen Sie für den Abschnitt Send notifications to (Benachrichtigung senden an) einen der folgenden Schritte aus:

    - Wählen Sie New email topic (Neues E-Mail-Thema) aus. Geben Sie einen Namen für Ihr E-Mail-Thema und eine Liste der Empfänger ein. Wir empfehlen, dass Sie die Ereignisabonnements mit derselben E-Mail-Adresse konfigurieren wie Ihren primären Kontokontakt. Die Empfehlungen, Serviceereignisse und persönlichen Zustandsnachrichten werden über verschiedene Kanäle gesendet. Die Abonnements für dieselbe E-Mail-Adresse stellen sicher, dass alle Nachrichten an einem Ort zusammengefasst werden.

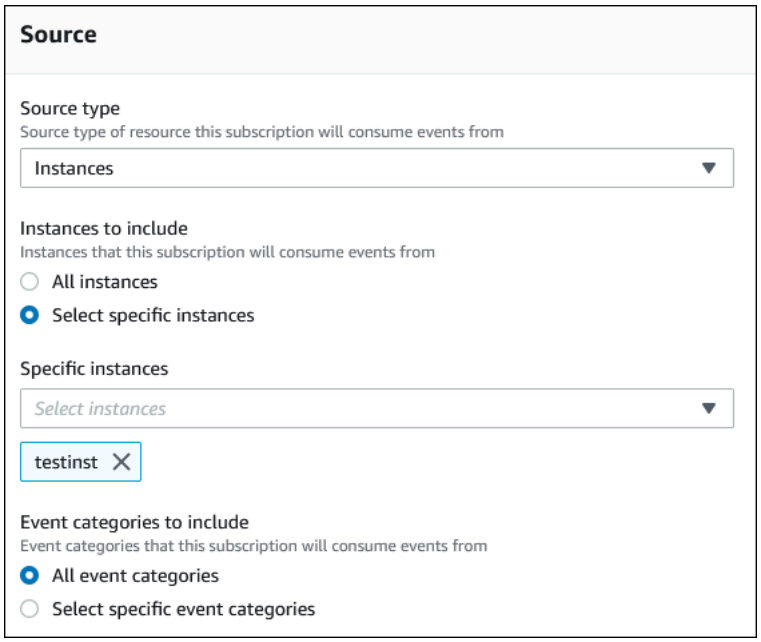
- Wählen Sie Amazon Resource Name (ARN) (Amazon-Ressourcenname (ARN)) aus. Wählen Sie dann einen vorhandenen Amazon-SNS-ARN für ein Amazon-SNS-Thema aus.

  Wenn Sie ein Thema verwenden möchten, das für die serverseitige Verschlüsselung (SSE) aktiviert wurde, gewähren Sie Amazon RDS die erforderlichen Berechtigungen für den Zugriff auf AWS KMS key. Weitere Informationen finden Sie unter Ermöglichen der Kompatibilität zwischen Ereignisquellen aus AWS-Services und verschlüsselten Themen im Entwicklerhandbuch für Amazon Simple Notification Service.

c. Wählen Sie unter Quelltyp einen Quelltyp aus. Wählen Sie beispielsweise Clusters (Cluster) oder Cluster snapshots (Cluster-Snapshots) aus.

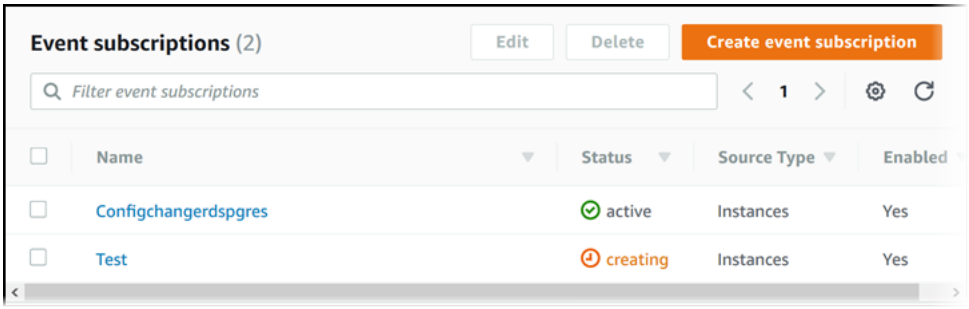
d. Wählen Sie die Ereigniskategorien und Ressourcen aus, für die Sie Ereignisbenachrichtigungen erhalten möchten.

   Im folgenden Beispiel werden Ereignisbenachrichtigungen für die DB-Instance mit dem Namen `testinst` konfiguriert.



e. Wählen Sie Erstellen aus.

In der Amazon-RDS-Konsole wird die Erstellung des Abonnements angezeigt.



AWS CLI

Um RDS-Ereignisbenachrichtigungen zu abonnieren, verwenden Sie den AWS CLI-Befehl `create-event-subscription`. Nutzen Sie die folgenden erforderlichen Parameter:

- `--subscription-name`
- `--sns-topic-arn`

Example

Für Linux, macOS oder Unix:

```
aws rds create-event-subscription \
    --subscription-name myeventsubscription \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

Für Windows:

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

Rufen Sie die Amazon-RDS-API-Funktion `CreateEventSubscription` auf, um Amazon-RDS-Ereignisbenachrichtigungen zu abonnieren. Nutzen Sie die folgenden erforderlichen Parameter:

- `SubscriptionName`
- `SnsTopicArn`

Tags und Attribute von Amazon-RDS-Ereignisbenachrichtigungen

Wenn Amazon RDS eine Ereignisbenachrichtigung an Amazon Simple Notification Service (SNS) oder Amazon EventBridge sendet, enthält die Benachrichtigung Nachrichtenattribute und Ereignis-Tags. RDS sendet die Nachrichtenattribute separat mit der Nachricht, während sich die Ereignis-Tags im Nachrichtentext befinden. Verwenden Sie die Nachrichtenattribute und die Amazon-RDS-Tags, um Ihren Ressourcen Metadaten hinzuzufügen. Sie können diese Tags mit Ihren eigenen Notationen über die DB-Instances, Aurora-Clustern usw. ändern. Weitere Informationen über das Markieren von Amazon-RDS-Ressourcen mit Tags finden Sie unter Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen.

Standardmäßig empfangen Amazon SNS und Amazon EventBridge jede an sie gesendete Nachricht. SNS und EventBridge können die Nachricht filtern und die Benachrichtigungen an den bevorzugten Kommunikationsmodus senden, z. B. eine E-Mail, eine SMS oder einen Aufruf eines HTTP-Endpunkts.

> **Note**
>
> Die Benachrichtigung, die in einer E-Mail oder einer SMS gesendet wird, enthält keine Ereignis-Tags.

Die folgende Tabelle zeigt die Nachrichtenattribute für RDS-Ereignisse, die an den Themen-Subscriber gesendet wurden.

Amazon-RDS-Ereignisattribut	Beschreibung
EventID	Kennung der RDS-Ereignisnachricht, z. B. RDS-EVENT-0006.
Ressource	Die ARN-ID für die Ressource, die das Ereignis aussendet, z. B. `arn:aws:rds:ap-southeast-2:123456789012:db:database-1` .

Die RDS-Tags liefern Daten über die Ressource, die vom Serviceereignis betroffen war. RDS fügt den aktuellen Status der Tags im Nachrichtentext hinzu, wenn die Benachrichtigung an SNS oder EventBridge gesendet wird.

Weitere Informationen zur Nachrichtenfilterung für SNS finden Sie unter Amazon-SNS-Nachrichtenfilterung im Entwicklerhandbuch zu Amazon Simple Notification Service.

Weitere Informationen zum Filtern von Ereignis-Tags für EventBridge finden Sie unter Comparison operators for use in event patterns in Amazon EventBridge im Amazon-EventBridge-Benutzerhandbuch.

Weitere Informationen zum Filtern von nutzlastbasierten Tags für SNS finden Sie unter Introducing payload-based message filtering for Amazon SNS.

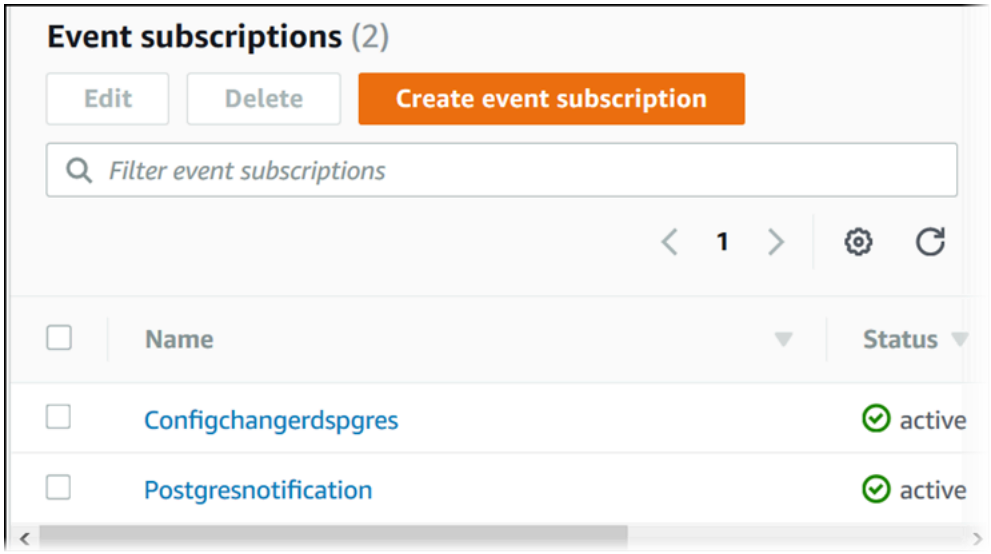
Auflisten von Abonnements für Amazon-RDS-Ereignisbenachrichtigungen

Sie können Ihre aktuellen Abonnements für Amazon-RDS-Ereignisbenachrichtigungen in einer Liste anzeigen.

Konsole

So zeigen Sie Ihre aktuellen Abonnements für Amazon-RDS-Ereignisbenachrichtigungen in einer Liste an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Ereignisabonnements aus. Im Bereich Ereignisabonnements werden all Ihre Abonnements für Ereignisbenachrichtigungen angezeigt.



AWS CLI

Verwenden Sie den AWS CLI-Befehl `describe-event-subscriptions`, um Ihre aktuellen Abonnements zu Amazon-RDS-Ereignisbenachrichtigungen auflisten zu lassen.

Example

Im folgenden Beispiel werden alle Ereignisabonnements beschrieben.

```
aws rds describe-event-subscriptions
```

Das folgende Beispiel beschreibt den Stack `myfirsteventsubscription`.

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

Rufen Sie die Amazon-RDS-API-Aktion `DescribeEventSubscriptions` auf, um Ihre aktuellen Abonnements zu Amazon-RDS-Ereignisbenachrichtigungen auflisten zu lassen.

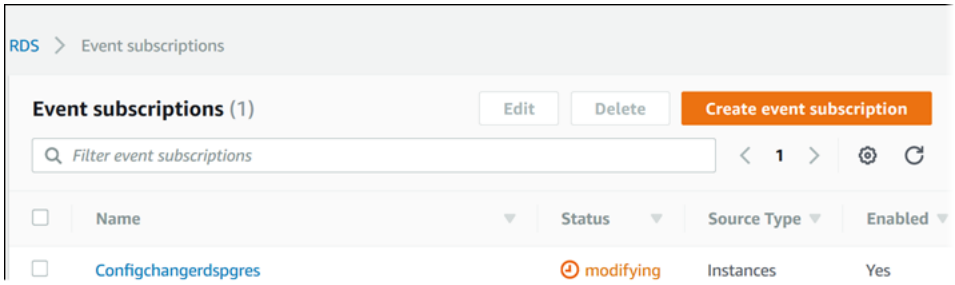
Ändern eines Abonnements für Amazon-RDS-Ereignisbenachrichtigungen

Nachdem Sie ein Abonnement erstellt haben, können Sie den Namen des Abonnements, die Quell-ID, Kategorien oder den Thema-ARN ändern.

Konsole

So ändern Sie ein Abonnement für Amazon-RDS-Ereignisbenachrichtigungen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Ereignisabonnements aus.
3. Wählen Sie im Bereich Ereignisabonnements das Abonnement, das Sie modifizieren möchten, und klicken Sie auf Bearbeiten.
4. Nehmen Sie Ihre Änderungen am Abonnement im Bereich Ziel oder Quelle vor.
5. Wählen Sie Bearbeiten aus. In der Amazon-RDS-Konsole wird die Änderung des Abonnements angezeigt.



AWS CLI

Verwenden Sie den AWS CLI-Befehl `modify-event-subscription`, um ein Abonnement für Amazon-RDS-Ereignisbenachrichtigungen zu ändern. Verwenden Sie den folgenden erforderlichen Parameter:

- `--subscription-name`

Example

Mit folgendem Code wird aktivier `myeventsubscription`.

Für Linux, macOS oder Unix:

```
aws rds modify-event-subscription \
    --subscription-name myeventsubscription \
    --enabled
```

Für Windows:

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

Um ein Amazon-RDS-Ereignis zu ändern, rufen Sie die Amazon-RDS-API-Operation auf `ModifyEventSubscription`. Verwenden Sie den folgenden erforderlichen Parameter:

- `SubscriptionName`

Hinzufügen einer Quell-ID zu einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen

Sie können einem vorhandenen Abonnement eine Quell-ID (die Amazon-RDS-Quelle, von der das Ereignis generiert wird) hinzufügen.

Konsole

Sie können Quell-IDs einfach über die Amazon-RDS-Konsole hinzufügen oder entfernen, indem Sie diese beim Ändern eines Abonnements aktivieren oder deaktivieren. Weitere Informationen finden Sie unter Ändern eines Abonnements für Amazon-RDS-Ereignisbenachrichtigungen.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `add-source-identifier-to-subscription`, um einen „Quellenidentifizierer" zu einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen hinzuzufügen. Nutzen Sie die folgenden erforderlichen Parameter:

- `--subscription-name`
- `--source-identifier`

Example

Im folgenden Beispiel wird die Quell-ID `mysqldb` zum Abonnement `myrdseventsubscription` hinzugefügt.

Für Linux, macOS oder Unix:

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Für Windows:

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Rufen Sie die Amazon-RDS-API auf, um eine Quell-ID zu einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen hinzuzufügen `AddSourceIdentifierToSubscription`. Nutzen Sie die folgenden erforderlichen Parameter:

- `SubscriptionName`
- `SourceIdentifier`

Entfernen einer Quell-ID aus einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen

Sie können eine Quell-ID (die Amazon-RDS-Quelle, von der das Ereignis generiert wird) aus einem Abonnement entfernen, wenn Sie keine weiteren Benachrichtigungen mehr über Ereignisse für diese Quelle erhalten möchten.

Konsole

Sie können Quell-IDs einfach über die Amazon-RDS-Konsole hinzufügen oder entfernen, indem Sie diese beim Ändern eines Abonnements aktivieren oder deaktivieren. Weitere Informationen finden Sie unter Ändern eines Abonnements für Amazon-RDS-Ereignisbenachrichtigungen.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `remove-source-identifier-from-subscription`, um einen „Source Quellenidentifizierer" aus einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen zu entfernen. Nutzen Sie die folgenden erforderlichen Parameter:

- `--subscription-name`
- `--source-identifier`

Example

Im folgenden Beispiel wird die Quell-ID `mysqldb` aus dem Abonnement `myrdseventsubscription` entfernt.

Für Linux, macOS oder Unix:

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Für Windows:

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Verwenden Sie den Amazon-RDS-API-Befehl `RemoveSourceIdentifierFromSubscription`, um einen „Source Identifier” aus einem Abonnement für Amazon-RDS-Ereignisbenachrichtigungen zu entfernen. Nutzen Sie die folgenden erforderlichen Parameter:

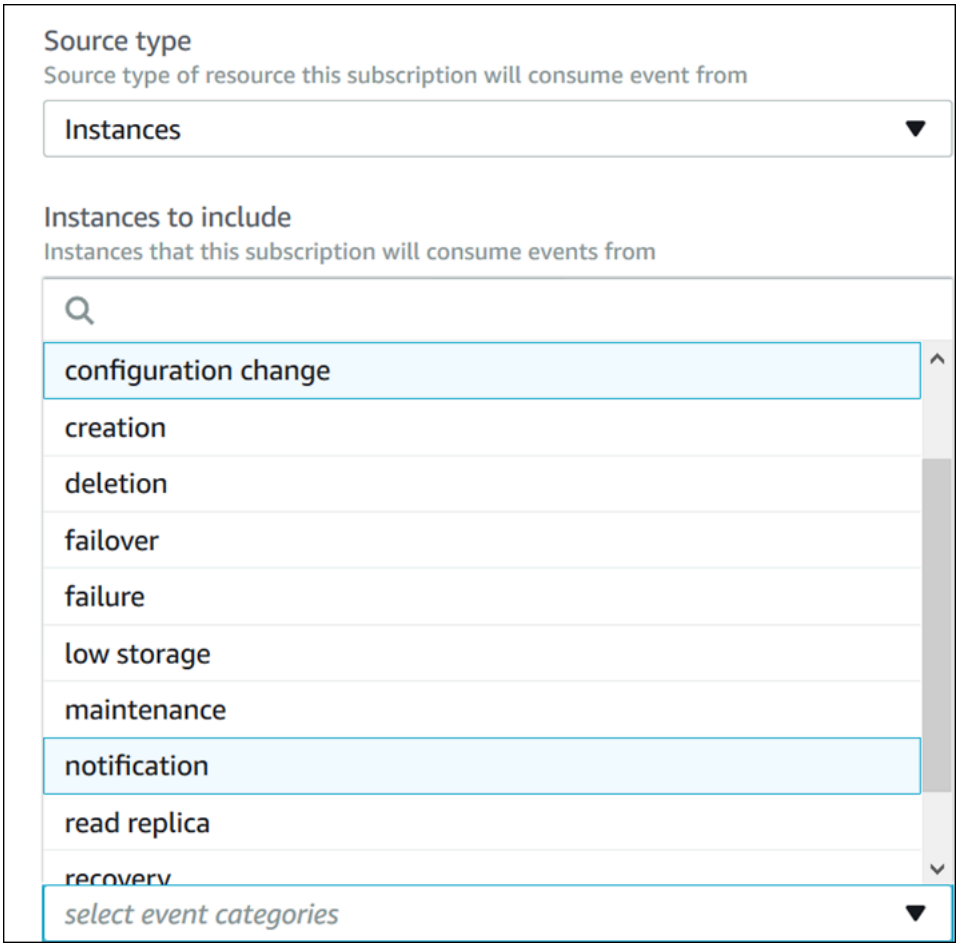
- `SubscriptionName`
- `SourceIdentifier`

Auflisten von Kategorien für Amazon-RDS-Ereignisbenachrichtigungen

Alle Ereignisse für einen Ressourcentyp sind in Kategorien gruppiert. Gehen Sie wie folgt vor, um die verfügbaren Kategorien in einer Liste anzuzeigen.

Konsole

Wenn Sie ein Abonnement für Ereignisbenachrichtigungen erstellen oder ändern, werden die Ereigniskategorien in der Amazon-RDS-Konsole angezeigt. Weitere Informationen finden Sie unter Ändern eines Abonnements für Amazon-RDS-Ereignisbenachrichtigungen.



AWS CLI

Verwenden Sie den AWS CLI-Befehl `describe-event-categories`, um die Amazon-RDS-Ereignisbenachrichtigungskategorien auflisten zu lassen. Dieser Befehl hat keine erforderlichen Parameter.

Example

```
aws rds describe-event-categories
```

API

Verwenden Sie den Amazon-RDS-API-Befehl `DescribeEventCategories`, um die Amazon-RDS-Ereignisbenachrichtigungskategorien auflisten zu lassen. Dieser Befehl hat keine erforderlichen Parameter.

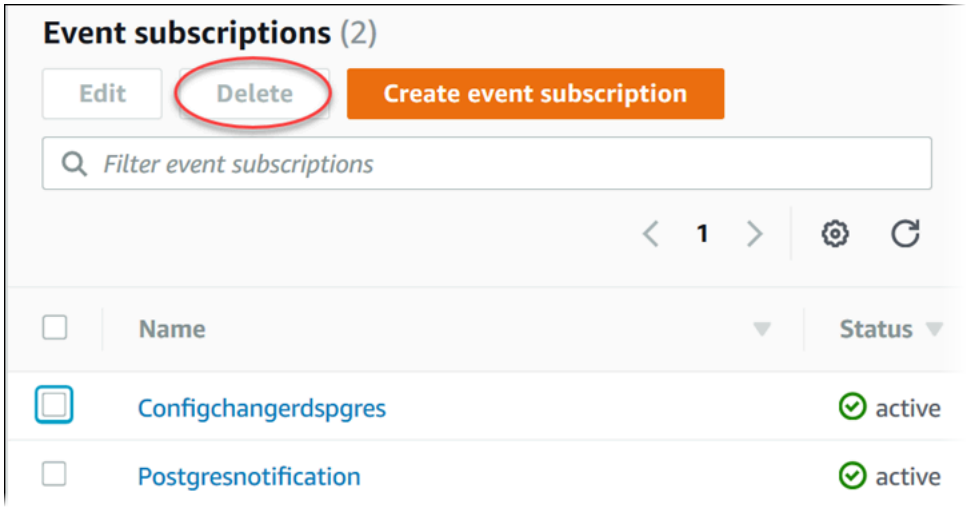
Löschen eines Abonnements für Amazon-RDS-Ereignisbenachrichtigungen

Sie können ein Abonnement löschen, wenn Sie es nicht mehr benötigen. Alle Abonnenten des Themas erhalten dann keine weiteren Ereignisbenachrichtigungen, die über dieses Abonnement ausgegeben wurden.

Konsole

So löschen Sie ein Abonnement für Amazon-RDS-Ereignisbenachrichtigungen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich DB Event Subscriptions (DB-Ereignisabonnements) aus.
3. Wählen Sie im Bereich My DB Event Subscriptions (Meine DB-Ereignisabonnements) das Abonnement aus, das Sie löschen möchten.
4. Wählen Sie Delete (Löschen).
5. In der Amazon-RDS-Konsole wird die Löschung des Abonnements angezeigt.



AWS CLI

Verwenden Sie den AWS CLI-Befehl `delete-event-subscription`, um ein Abonnement für Amazon-RDS-Ereignisbenachrichtigungen zu löschen. Verwenden Sie den folgenden erforderlichen Parameter:

- `--subscription-name`

Example

Im folgenden Beispiel wird das Abonnement gelösch `myrdssubscription`.

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

Verwenden Sie den RDS-API-Befehl `DeleteEventSubscription`, um ein Abonnement für Amazon-RDS-Ereignisbenachrichtigungen zu löschen. Verwenden Sie den folgenden erforderlichen Parameter:

- `SubscriptionName`

Erstellen einer Regel, die bei einem Amazon Aurora-Ereignis ausgelöst wird

Mit Amazon EventBridge können Sie AWS-Services automatisieren und auf Systemereignisse reagieren, z. B. Probleme mit der Anwendungsverfügbarkeit oder Ressourcenänderungen.

Themen

- Tutorial: Protokollieren von Statusänderungen an einer DB-Instance mit Amazon EventBridge

Tutorial: Protokollieren von Statusänderungen an einer DB-Instance mit Amazon EventBridge

Sie können eine AWS Lambda-Funktion erstellen, die die Statusänderungen einer -Instance protokolliert. Anschließend erstellen Sie eine Regel, die die Funktion ausführt, sobald eine Statusänderung einer vorhandenen RDS-DB-Instance stattfindet. Das Tutorial geht davon aus, dass Sie eine kleine laufende Test-Instance haben, die Sie vorübergehend herunterfahren können.

> **Important**
>
> Führen Sie dieses Tutorial nicht für eine laufende Produktions-DB-Instance durch.

Themen

- Schritt 1: Erstellen einer AWS Lambda-Funktion
- Schritt 2: Erstellen einer Regel
- Schritt 3: Testen der Regel

Schritt 1: Erstellen einer AWS Lambda-Funktion

Erstellen Sie eine Lambda-Funktion, um die Statusänderungsereignisse zu protokollieren. Sie geben diese Funktion beim Erstellen der Regel an.

So erstellen Sie eine Lambda-Funktion:

1. Öffnen Sie die AWS Lambda-Konsole unter https://console.aws.amazon.com/lambda/.
2. Wenn Sie noch nicht mit Lambda gearbeitet haben, wird Ihnen eine Willkommensseite angezeigt. Wählen Sie Get Started Now. Andernfalls, wählen Sie Create function (Funktion erstellen) aus.
3. Wählen Sie Von Grund auf neu schreiben aus.

4. Gehen Sie auf der Seite Create function (Funktion erstellen) wie folgt vor:

   a. Geben Sie einen Namen und eine Beschreibung für die Lambda-Funktion ein. Geben Sie der Funktion beispielsweise den Namen **`RDSInstanceStateChange`**.

   b. Wählen Sie in Runtime Node.js 16x aus.

   c. Wählen Sie für Architecture (Architektur) x86_64 aus.

   d. Führen Sie für Execution role (Ausführungsrolle) einen der folgenden Schritte aus:

      - Wählen Sie Create a new role with basic Lambda permissions (Eine neue Rolle mit den grundlegenden Lambda-Berechtigungen erstellen) aus.
      - Wählen Sie für Execution role (Ausführungsrolle) die Option Use an existing role (Vorhandene Rolle verwenden) aus. Wählen Sie die Rolle aus.

   e. Wählen Sie Funktion erstellen.

5. Machen Sie auf der Seite RDSInstanceStateChange Folgendes:

   a. In Code-Quelle wählen Sie index.js.

   b. Im Ausschnitt index.js löschen Sie den vorhandenen Code.

   c. Geben Sie den folgenden Code ein:

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

   d. Wählen Sie Deploy (Bereitstellen) aus.

Schritt 2: Erstellen einer Regel

Erstellen Sie eine Regel, damit die Lambda-Funktion ausgeführt wird, wenn Sie eine Amazon-RDS-Instance starten.

So erstellen Sie die EventBridge-Regel

1. Öffnen Sie die Amazon EventBridge-Konsole unter https://console.aws.amazon.com/events/.
2. Wählen Sie im Navigationsbereich Regeln aus.
3. Wählen Sie Regel erstellen aus.

4. Geben Sie einen Namen und eine Beschreibung für die Regel ein. Geben Sie z. B. ei **`RDSInstanceStateChangeRule`**.
5. Wählen Sie Rule with an event pattern 8Regel mit einem Ereignismuster) und dann Next (Weiter) aus.
6. Wählen Sie für Ereignisquelle AWS-Ereignisse oder EventBridge-Partnerereignisse.
7. Scrollen Sie nach unten zum Abschnitt Event pattern (Ereignismuster).
8. Wählen Sie für Ereignisquelle die Option AWS-Services aus.
9. Wählen Sie für AWS-Service die Option Relational Database Service (RDS) aus.
10. Für Ereignistyp wählen Sie RDS DB-Instance-Ereignis.
11. Übernehmen Sie das Standard-Ereignismuster. Wählen Sie anschließend Weiter.
12. Bei Zieltypen wählen Sie AWS-Service aus.
13. Für Select a target (Ein Ziel auswählen), wählen die Option Lambda function (Lambda-Funktion) aus.
14. Wählen Sie für Function (Funktion) die Lambda-Funktion aus, die Sie erstellt haben. Wählen Sie anschließend Weiter.
15. Wählen Sie in Configure tags (Tags konfigurieren) Next (Weiter) aus.
16. Überprüfen Sie die Schritte in Ihrer Regel. Wählen Sie dann Create rule (Regel erstellen) aus.

Schritt 3: Testen der Regel

Um Ihre Regel zu testen, fahren Sie eine RDS-DB-Instance herunter. Warten Sie einige Minuten, bis die Instance heruntergefahren wurde, und prüfen Sie dann, ob Ihre Lambda-Funktion aufgerufen wurde.

Testen der Regel durch Anhalten einer DB-Instance

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Stopp einer RDS-DB-Instance.
3. Öffnen Sie die Amazon EventBridge-Konsole unter https://console.aws.amazon.com/events/.
4. Wählen Sie im Navigationsbereich Rules (Regeln), den Namen der von Ihnen erstellten Regel aus.
5. Wählen Sie unter Regeldetails die Option Überwachung aus.

Sie werden zur Amazon CloudWatch-Konsole umgeleitet. Wenn Sie nicht weitergeleitet werden, klicken Sie auf Die Metriken in CloudWatch anzeigen.

6. In Alle Metriken wählen Sie den Namen der Regel aus, die Sie erstellt haben.

   Das Diagramm sollte darauf hinweisen, dass die Regel aufgerufen wurde.

7. Wählen Sie im Navigationsbereich Log groups (Protokollgruppen) aus.
8. Wählen Sie den Namen der Protokollgruppe für die Lambda-Funktion aus (/aws/lambda/***function-name***).
9. Wählen Sie den Namen des Protokoll-Streams aus, um die von der Funktion für die von Ihnen gestartete Instance bereitgestellten Daten anzuzeigen. Das empfangene Ergebnis sollte in etwa wie folgt aussehen:

```
{
    "version": "0",
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```

Weitere Beispiele für RDS-Ereignisse im JSON-Format finden Sie unter Überblick über Ereignisse für Aurora.

10. (Optional) Zum Abschluss können Sie die Amazon-RDS-Konsole öffnen und die von Ihnen gestoppte Instance starten.

Amazon-RDS-Ereigniskategorien und -Ereignismeldungen für Aurora

Amazon RDS generiert eine beträchtliche Anzahl von Ereignissen in Kategorien, die Sie über die Amazon RDS-Konsole oder die API abonnieren können. AWS CLI

Themen

- DB-Cluster-Ereignisse
- DB-Cluster-Snapshot-Ereignisse
- DB-Instance-Ereignisse
- DB-Parametergruppenereignisse
- DB-Sicherheitsgruppenereignisse
- DB-Shard-Gruppenereignisse
- RDS-Proxy-Ereignisse
- Blau/Grün-Bereitstellungsereignisse

DB-Cluster-Ereignisse

Die folgende Tabelle zeigt den Ereignistyp sowie eine Liste der Ereignisse für den Fall, dass ein DB-Cluster der Quelltyp ist.

> **Note**
>
> Im DB-Cluster-Ereignistyp ist keine Ereigniskategorie für Aurora Serverless vorhanden. Die Aurora Serverless-Ereignisse reichen von RDS-EVENT-0141 bis RDS-EVENT-0149.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0016	Die Anmeldeinformationen des Hauptbenutzers werden zurückgesetzt.	Keine
Konfigurationsänderung	RDS-EVENT-0179	Database Activity Streams wird auf Ihrem Datenbank-Cluster gestartet.	Weitere Informationen finden Sie unter Überwachung von Amazon Aurora

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
			mithilfe von Datenbank aktivitätsstreams.
Konfigurationsänderung	RDS-EVENT-0180	Database Activity Streams wird auf Ihrem Datenbank-Cluster angehalten.	Weitere Informationen finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbank aktivitätsstreams.
Erstellung	RDS-EVENT-0170	DB-Cluster erstellt.	Keine
Löschung	RDS-EVENT-0171	DB-Cluster gelöscht.	Keine
Failover	RDS-EVENT-0069	Das Cluster-Failover ist fehlgeschlagen. Überprüfen Sie den Zustand Ihrer Cluster-Instances und versuchen Sie es erneut.	Keine
Failover	RDS-EVENT-0070	Erneute Werbung für die vorherige Primarver einigung:*name*.	Keine
Failover	RDS-EVENT-0071	Failover zur DB-Instance abgeschlossen:*name*.	Keine
Failover	RDS-EVENT-0072	Derselbe AZ-Failover zur DB-Instance gestartet:. *name*	Keine
Failover	RDS-EVENT-0073	Cross-AZ-Failover zur DB-Instance gestartet:. *name*	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0083	Amazon RDS konnte keine Anmeldeinformationen für den Zugriff auf Ihren Amazon S3 S3-Bucket für Ihren DB-Cluster erstellen *name*. Dies liegt daran, dass die IAM-Rolle für die S3-Snapshot-Erfassung in Ihrem Konto nicht richtig konfiguriert ist oder der angegebene Amazon-S3 -Bucket nicht gefunden werden kann. Weitere Informationen finden Sie im Abschnitt zur Fehlerbeh ebung in der Amazon-RDS-Dokumentation.	Weitere Informationen finden Sie unter Physische Migration von MySQL mithilfe von Percona XtraBackup und Amazon S3 .
Ausfall	RDS-EVENT-0143	Der DB-Cluster konnte aus diesem *units* Grund nicht von *units* bis skaliert *reason* werden:.	Skalierung fehlgeschlagen für den Aurora Serverless-DB-Cluster.
Ausfall	RDS-EVENT-0354	Sie können den DB-Cluste r aufgrund inkompatibler Ressourcen nicht erstellen. *message*.	Das *message* beinhaltet Details zum Fehler.
Ausfall	RDS-EVENT-0355	Der DB-Cluster kann aufgrund unzureichender Ressourcenlimits nicht erstellt werden. *message*.	Das *message* beinhaltet Details zum Fehler.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0387	Fehler beim Partitionieren der DB-Instances im DB-Cluster *name* für das Patchen.	Die Betriebssystem-Upgrades für die DB-Instances im DB-Cluster sind fehlgeschlagen.
Globales Failover	RDS-EVENT-0181	Der globale Switchover zum DB-Cluster *name* in der Region *name* wurde gestartet.	Dieses Ereignis bezieht sich auf einen Umstellungsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet). Der Prozess kann verzögert werden, da andere Vorgänge auf dem DB-Cluster ausgeführt werden.
Globales Failover	RDS-EVENT-0182	Der alte primäre DB-Cluster *name* in Region *name* wurde erfolgreich heruntergefahren.	Dieses Ereignis bezieht sich auf einen Umstellungsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet). Die alte primäre Instance in der globalen Datenbank akzeptiert keine Schreibvorgänge. Alle Volumes sind synchronisiert.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Globales Failover	RDS-EVENT-0183	Es wird auf die Datensynchronisierung zwischen den globalen Clustermitgliedern gewartet. Der aktuelle Stand liegt hinter dem primären DB-Cluster zurück: *reason*.	Dieses Ereignis bezieht sich auf einen Umstellungsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet). Während der Synchronisierungsphase des globalen Datenbank-Failovers tritt eine Replikationsverzögerung auf.
Globales Failover	RDS-EVENT-0184	Der neue primäre DB-Cluster *name* in Region *name* wurde erfolgreich hochgestuft.	Dieses Ereignis bezieht sich auf einen Umstellungsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet). Die Volumentopologie der globalen Datenbank wird mit dem neuen primären Volume wiederhergestellt.
Globales Failover	RDS-EVENT-0185	Die globale Umstellung auf den DB-Cluster *name* in der Region *name* wurde abgeschlossen.	Dieses Ereignis bezieht sich auf einen Umstellungsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet). Die globale Datenbankumstellung wurde auf dem primären DB-Cluster abgeschlossen. Nach Abschluss des Failovers kann es lange dauern, bis Replikate online sind.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Globales Failover	RDS-EVENT-0186	Der globale Switchover zum DB-Cluster *name* in Region wurde abgebroch en*name*.	Dieses Ereignis bezieht sich auf einen Umstellun gsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet).
Globales Failover	RDS-EVENT-0187	Der globale Switchover zum DB-Cluster *name* in Region *name* ist fehlgesch lagen.	Dieses Ereignis bezieht sich auf einen Umstellun gsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet).
Globales Failover	RDS-EVENT-0238	Der globale Failover zum DB-Cluster *name* in der Region *name* wurde abgeschlossen.	Keine
Globales Failover	RDS-EVENT-0239	Das globale Failover zum DB-Cluster *name* in der Region *name* ist fehlgesch lagen.	Keine
Globales Failover	RDS-EVENT-0240	*name*Nach dem globalen Failover wurde mit der Neusynchronisierung von Mitgliedern des DB-Clusters *name* in Region begonnen.	Keine
Globales Failover	RDS-EVENT-0241	Die Neusynchronisierung von Mitgliedern des DB-Clusters *name* in Region wurde nach dem globalen Failover abgeschlossen. *name*	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Globales Failover	RDS-EVENT-0397	Aurora hat die Änderung des DNS-Namens abgeschlossen, in den der globale Writer-Endpunkt aufgelöst wird.	Keine
Globales Failover	RDS-EVENT-0423	Es wird auf die Datensynchronisierung mit dem Ziel-DB-Cluster gewartet. Der aktuelle Ziel-DB-Cluster hinkt dem primären DB-Cluster hinterher: `%s`.	Dieses Ereignis bezieht sich auf einen Umstellungsvorgang (früher als „verwaltetes geplantes Failover“ bezeichnet). Während der Synchronisierungsphase des globalen Datenbank-Failovers tritt eine Replikationsverzögerung auf.
Wartung	RDS-EVENT-0156	Der DB-Cluster verfügt über ein Upgrade der DB-Engine für kleinere Versionen.	Keine
Wartung	RDS-EVENT-0173	Die Version der Datenbank-Cluster-Engine wurde aktualisiert.	Das Patchen des DB-Clusters ist abgeschlossen.
Wartung	RDS-EREIGNIS-0174	Der Datenbank-Cluster befindet sich in einem Zustand, für den kein Upgrade durchgeführt werden kann.	Keine
Wartung	RDS-EVENT-0176	Die Hauptversion der Datenbank-Cluster-Engine wurde aktualisiert.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wartung	RDS-EREIGNIS-0177	Datenbank-Cluster-Upgrade ist in Bearbeitung.	Keine
Wartung	RDS-EVENT-0286	Das Versionsupgrade der Datenbank-Cluster-Engine wurde gestartet*version_number* . Der Cluster bleibt online.	Keine
Wartung	RDS-EVENT-0287	Es wurde festgestellt, dass ein Betriebssystem-Upgrade erforderlich ist.	Keine
Wartung	RDS-EVENT-0288	Das Upgrade des Cluster-Betriebssystems wird gestartet.	Keine
Wartung	RDS-EVENT-0289	Das Upgrade des Cluster-Betriebssystems wurde abgeschlossen.	Keine
Wartung	RDS-EVENT-0290	Der Datenbankcluster wurde gepatcht: Quellversion *version_number* =>*new_version_number* .	Keine
Wartung	RDS-EVENT-0363	Vorbereitung des Upgrades ist im Gange: *cluster_name*	Die Upgrade-Vorabprüfungen für den DB-Cluster wurden gestartet.
Wartung	RDS-EVENT-0388	Offline-Patches für DB-Instances in Partition *name*/*name*für DB-Cluster starten:. *name partition_n*	Betriebssystem-Upgrades für die DB-Instances im DB-Cluster werden gestartet.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wartung	RDS-EVENT-0389	Wir konnten Ihr DB-Cluster-Betriebssystem nicht aktualisieren. Sie können auf das nächste Wartungsfenster warten oder Ihr DB-Cluster-Betriebssystem manuell aktualisieren.	Keine
Wartung	RDS-EREIGNIS-0424	Auf dem DB-Cluster *name* wird eine Version ausgeführt*version*, die höher ist als die Ziel-Upgrade-Version *version* für den globalen Cluster. Es wird nicht empfohlen, einen sekundären Cluster auf einer höheren Version als dem globalen Cluster zu verwenden, da dies zu Problemen beim Failover oder Switchover führen kann. Erwägen Sie, Ihren globalen Cluster entsprechend zu aktualisieren.	Keine
Benachrichtigung	RDS-EVENT-0076	Fehler bei der Migration von *name* zu*name*. Grund:*reason*.	Migration zu einem Aurora-DB-Cluster fehlgeschlagen.
Benachrichtigung	RDS-EVENT-0077	Konnte nicht konvertieren*name*. *name*zu InnoDB. Grund:*reason*.	Während der Migration zu einem Aurora-DB-Cluster ist der Versuch fehlgeschlagen, ein Tabellenformular aus der Quelldatenbank in InnoDB zu konvertieren.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung	RDS-EVENT-0085	Das DB-Cluster konnte nicht aktualisiert werden*name*, da die Instance den Status *name* hat*name*. Beheben Sie das Problem oder löschen Sie die Instance und versuchen Sie es erneut.	Beim Versuch, den Aurora DB-Cluster zu patchen, ist ein Fehler aufgetreten. Überprüfen Sie Ihren Instance-Status, lösen Sie das Problem und versuchen Sie es erneut. Weitere Informationen finden Sie unter Warten eines Amazon Aurora-DB-Clusters.
Benachrichtigung	RDS-EVENT-0141	Aus diesem Grund wird der DB-Cluster von *units* bis *units* skaliert:*reason*.	Skalierung initiiert für den Aurora Serverless-DB-Cluster.
Benachrichtigung	RDS-EVENT-0142	Der DB-Cluster wurde von *units* bis *units* skaliert.	Skalierung abgeschlossen für den Aurora Serverless-DB-Cluster.
Benachrichtigung	RDS-EVENT-0144	Der DB-Cluster wird angehalten.	Für den DB-Cluster von Aurora Serverless wurde eine automatische Pause eingeleitet.
Benachrichtigung	RDS-EVENT-0145	Der Cluster wird angehalten.	Der DB-Cluster von Aurora Serverless wurde angehalten.
Benachrichtigung	RDS-EVENT-0146	Das Anhalten des DB-Clusters wurde abgebrochen.	Das Anhalten des DB-Clusters von Aurora Serverless wurde abgebrochen.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung	RDS-EVENT-0147	Der Betrieb des DB-Clusters wird fortgesetzt.	Für den DB-Cluster von Aurora Serverless wurde eine Fortsetzungsoperation eingeleitet.
Benachrichtigung	RDS-EVENT-0148	Der Betrieb des DB-Clusters wird fortgesetzt.	Die Fortsetzungsoperation für den DB-Cluster von Aurora Serverless ist abgeschlossen.
Benachrichtigung	RDS-EVENT-0149	Der DB-Cluster wurde von *units* bis skaliert*units*, aber die Skalierung verlief aus diesem Grund nicht reibungslos:. *reason*	Die nahtlose Skalierung des mit Durchsetzungsoption für den Aurora Serverless-DB-Cluster wurde abgeschlossen. Verbindungen wurden möglicherweise unterbrochen, wenn erforderlich.
Benachrichtigung	RDS-EVENT-0150	Der DB-Cluster wurde gestoppt.	Keine
Benachrichtigung	RDS-EVENT-0151	Der DB-Cluster wurde gestartet.	Keine
Benachrichtigung	RDS-EVENT-0152	Das Stoppen des DB-Clusters ist fehlgeschlagen.	Keine
Benachrichtigung	RDS-EVENT-0153	Der DB-Cluster wird gestartet, da er die maximal zulässige Anhaltezeit überschritten hat.	Keine
Benachrichtigung	RDS-EVENT-0172	Der Cluster wurde von *name* bis *name* umbenannt.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung	RDS-EVENT-0234	Exportaufgabe fehlgeschlagen.	Bei der Aufgabe zum Exportieren des DB-Clusters ist ein Fehler aufgetreten.
Benachrichtigung	RDS-EVENT-0235	Exportaufgabe abgebrochen.	Die Aufgabe zum Exportieren des DB-Clusters wurde abgebrochen.
Benachrichtigung	RDS-EVENT-0236	Exportaufgabe abgeschlossen.	Die Aufgabe zum Exportieren des DB-Clusters ist abgeschlossen.
Benachrichtigung	RDS-EVENT-0386	DB-Instances im DB-Cluster *name* wurden partitioniert:. *list_of_partitions* DB-Cluster ist online.	Die Betriebssystem-Upgrades für die DB-Instances im DB-Cluster waren erfolgreich.
Benachrichtigung	RDS-EVENT-0426	RDS kann den DB-Cluster aufgrund inaktiver oder lang andauernder Transaktionen nicht *name* als Replikationsquelle konfigurieren. Warten Sie, bis die Transaktionen abgeschlossen sind, oder stornieren Sie sie und versuchen Sie es erneut.	Keine

DB-Cluster-Snapshot-Ereignisse

In der folgenden Tabelle werden die Ereigniskategorie und die Ereignisse für den Quelltyp „DB-Cluster-Snapshot“ aufgeführt.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
backup	RDS-EVENT-0074	Ein manueller Cluster-Snapshot wird erstellt.	Keine
backup	RDS-EVENT-0075	Ein manueller Cluster-Snapshot wurde erstellt.	Keine
Benachrichtigung	RDS-EVENT-0162	Bei der Aufgabe zum Exportieren von Cluster-Snapshots ist ein Fehler aufgetreten.	Keine
Benachrichtigung	RDS-EVENT-0163	Die Aufgabe zum Exportieren von Cluster-Snapshots wurde abgebrochen.	Keine
Benachrichtigung	RDS-EVENT-0164	Die Aufgabe zum Exportieren von Cluster-Snapshots ist abgeschlossen.	Keine
backup	RDS-EVENT-0168	Erstellen eines automatisierten Cluster-Snapshots.	Keine
backup	RDS-EVENT-0169	Automatisierter Cluster-Snapshot erstellt.	Keine

DB-Instance-Ereignisse

In der folgenden Tabelle werden die Ereigniskategorie und die Ereignisse für den Quelltyp "DB-Instance" aufgeführt.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Verfügbarkeit	RDS-EVENT-0004	DB-Instance-Shutdown.	Keine
Verfügbarkeit	RDS-EVENT-0006	Die DB-Instance wurde neu gestartet.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Verfügbarkeit	RDS-EVENT-0022	Fehler beim Neustart von MySQL:. *message*	Während des Neustarts von Aurora MySQLRDS für MariaDB ist ein Fehler aufgetreten.
Verfügbarkeit	RDS-EREIGNIS-0419	Amazon RDS konnte nicht auf den KMS Verschlüsselungsschlüssel für die Datenbank-Instance zugreifen*name*. Diese Datenbank wird in den Zustand versetzt, dass nicht auf sie zugegriffen werden kann. Weitere Informationen finden Sie im Abschnitt zur Fehlerbehebung in der Amazon-RDS-Dokumentation.	Keine
Backtrack (Rückverfolgung)	RDS-EVENT-0131	Das eigentliche Rückverfolgungsfenster kann kleiner als das von Ihnen angegebene Zielfenster der Rückverfolgung sein. Erwägen Sie, die Anzahl von Stunden im Zielfenster der Rückverfolgung zu reduzieren.	Weitere Informationen zur Rückverfolgung finden Sie unter Rückverfolgen eines Aurora-DB-Clusters.
Backtrack (Rückverfolgung)	RDS-EVENT-0132	Das tatsächliche Rückverfolgungsfenster entspricht dem Zielfenster der Rückverfolgung.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0011	Es wurde aktualisiert, um DBParameter Group zu verwenden*name*.	Keine
Konfigurationsänderung	RDS-EVENT-0012	Die Änderung wird auf die Datenbank-Instance-Klasse angewendet.	Keine
Konfigurationsänderung	RDS-EVENT-0014	Die Änderung wurde auf die DB-Instance-Klasse angewendet.	Keine
Konfigurationsänderung	RDS-EVENT-0017	Die Änderung wurde auf den zugewiesenen Speicher angewendet.	Keine
Konfigurationsänderung	RDS-EVENT-0025	Die Änderung zur Konvertierung in eine Multi-AZ-DB-Instance wurde angewendet.	Keine
Konfigurationsänderung	RDS-EVENT-0029	Die Änderung zur Konvertierung in eine standardmäßige (Single-AZ-)DB-Instance wurde angewendet.	Keine
Konfigurationsänderung	RDS-EVENT-0033	Es gibt *number* Benutzer, die dem Master-Benutzernamen entsprechen. Es wird nur der Benutzer zurückgesetzt, der nicht an einen bestimmten Host gebunden ist.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0067	Ihr Passwort konnte nicht zurückgesetzt werden. Informationen zum Fehler:. *message*	Keine
Konfigurationsänderung	RDS-EVENT-0078	Das Überwachungsintervall wurde geändert auf*number*.	Die Konfiguration von „Enhanced Monitoring“ (Erweiterte Überwachung) wurde geändert.
Konfigurationsänderung	RDS-EVENT-0092	Die Aktualisierung der DB-Parametergruppe ist abgeschlossen.	Keine
Erstellung	RDS-EVENT-0005	Die DB-Instance wurde erstellt.	Keine
Löschung	RDS-EVENT-0003	Die DB-Instance wurde gelöscht.	Keine
Ausfall	RDS-EVENT-0035	Datenbankinstanz, in die eingefügt wurde*state*. *message*.	Einige Parameter der DB-Instance sind ungültig. Wenn beispielsweise die DB-Instance nicht gestartet werden konnte, da der Wert eines speicherbezogenen Parameters zu hoch für diese Instance-Klasse ist, sollten Sie den Speicherparameter ändern und die DB-Instance neu starten.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0036	Datenbankinstanz in *state*. *message*.	Die DB-Instance befindet sich in einem inkompatiblen Netzwerk. Einige der angegebenen Subnetze IDs sind ungültig oder existieren nicht.
Ausfall	RDS-EVENT-0079	Amazon RDS konnte keine Anmeldeinformationen für die erweiterte Überwachung erstellen und diese Funktion wurde deaktiviert. Dies liegt wahrscheinlich daran, dass es in Ihrem Konto rds-monitoring-role nicht vorhanden und nicht korrekt konfiguriert ist. Weitere Informationen finden Sie im Abschnitt zur Fehlerbehebung in der Amazon-RDS-Dokumentation.	Die erweiterte Überwachung kann nicht ohne die IAM-Rolle für erweiterte Überwachung aktiviert werden. Weitere Informationen zum Erstellen der IAM-Rolle finden Sie unter So erstellen Sie eine IAM-Rolle für „Enhanced Monitoring“ (Erweiterte Überwachung) in Amazon RDS.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0080	Amazon RDS konnte die erweiterte Überwachung für Ihre Instance nicht konfigurieren: *name* und diese Funktion wurde deaktiviert. Dies liegt wahrscheinlich daran, dass es in Ihrem Konto rds-monitoring-role nicht vorhanden und nicht richtig konfiguriert ist. Weitere Informationen finden Sie im Abschnitt zur Fehlerbehebung in der Amazon-RDS-Dokumentation.	Die erweiterte Überwachung wurde aufgrund eines Fehlers während der Konfigurationsänderung deaktiviert. Wahrscheinlich ist die IAM-Rolle für erweiterte Überwachung falsch konfiguriert. Weitere Informationen über das Erstellen der IAM-Rolle für erweiterte Überwachung finden Sie unter So erstellen Sie eine IAM-Rolle für „Enhanced Monitoring“ (Erweiterte Überwachung) in Amazon RDS.
Ausfall	RDS-EVENT-0082	Amazon RDS konnte keine Anmeldeinformationen für den Zugriff auf Ihren Amazon S3 S3-Bucket für Ihre DB-Instance erstellen *name*. Dies liegt daran, dass die IAM-Rolle für die S3-Snapshot-Erfassung in Ihrem Konto nicht richtig konfiguriert ist oder der angegebene Amazon-S3-Bucket nicht gefunden werden kann. Weitere Informationen finden Sie im Abschnitt zur Fehlerbehebung in der Amazon-RDS-Dokumentation.	Aurora konnte die Sicherungsdaten aus einem Amazon S3-Bucket nicht kopieren. Wahrscheinlich sind die Berechtigungen für Aurora zum Zugriff auf den Amazon S3-Bucket falsch konfiguriert. Weitere Informationen finden Sie unter Physische Migration von MySQL mithilfe von Percona XtraBackup und Amazon S3 .

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0353	Die DB-Instance kann aufgrund unzureichender Ressourcenlimits nicht erstellt werden. *message*.	Das *message* beinhaltet Details zum Fehler.
Ausfall	RDS-EREIGNIS-0418	Amazon RDS kann nicht auf den KMS Verschlüsselungsschlüssel für die Datenbank-Instance zugreifen*name*. Dies liegt wahrscheinlich daran, dass der Schlüssel deaktiviert ist oder Amazon RDS nicht darauf zugreifen kann. Wenn dies so weitergeht, wird die Datenbank in den Zustand versetzt, dass nicht auf sie zugegriffen werden kann. Weitere Informationen finden Sie im Abschnitt zur Fehlerbehebung in der Amazon-RDS-Dokumentation.	Keine
Ausfall	RDS-EVENT-0420	Amazon RDS kann jetzt erfolgreich auf den KMS Verschlüsselungsschlüssel für die Datenbank-Instance zugreifen*name*.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wenig Speicherplatz	RDS-EVENT-0007	Der zugewiesene Speicherplatz ist ausgeschöpft. Weisen Sie zusätzlichen Speicherplatz zu, um das Problem zu lösen.	Der Speicherplatz, der für die DB-Instance zugewiesen wurde, ist aufgebraucht. Sie sollten der DB-Instance weiteren Speicher zuordnen, um dieses Problem zu lösen. Weitere Informationen finden Sie unter RDS FAQ. Sie können den Speicherplatz für eine DB-Instance mit der Metrik Freier Speicherplatz überwachen.
Wenig Speicherplatz	RDS-EVENT-0089	Die freie Speicherkapazität für die DB-Instance: *name* ist im Verhältnis *percentage* zum bereitgestellten Speicher gering [Provisioned Storage:*size*, Free Storage:*size*]. Erhöhen Sie ggf. den bereitgestellten Speicher, um dieses Problem zu beheben.	Die DB-Instance hat mehr als 90% ihres zugewiesenen Speichers verbraucht. Sie können den Speicherplatz für eine DB-Instance mit der Metrik Freier Speicherplatz überwachen.
Wenig Speicherplatz	RDS-EVENT-0227	Der Speicherplatz Ihres Aurora-Clusters ist gefährlich knapp, es sind nur noch *amount* Terabyte übrig. Bitte ergreifen Sie Maßnahmen, um die Speicherlast in Ihrem Cluster zu reduzieren.	Das Aurora-Speicher-Subsystem hat wenig Speicherplatz.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wartung	RDS-EVENT-0026	Es werden Offline-Patches auf die DB-Instance angewendet.	Die Offline-Wartung der DB-Instance wird gerade durchgeführt. Die DB-Instance ist zurzeit nicht verfügbar.
Wartung	RDS-EVENT-0027	Es wurden Offline-Patches auf die DB-Instance angewendet.	Die Offline-Wartung der DB-Instance ist abgeschlossen. Die DB-Instance ist nun verfügbar.
Wartung	RDS-EVENT-0047	Die Datenbank-Instance wurde gepatcht.	Keine
Wartung	RDS-EREIGNIS-0155	Die DB-Instance verfügt über ein Upgrade der DB-Engine für kleinere Versionen.	Keine
Wartung	RDS-EVENT-0178	Datenbank-Instance-Upgrade ist in Bearbeitung.	Keine
Wartung	RDS-EREIGNIS-0422	RDS ersetzt den Host der DB-Instance *name* aufgrund einer ausstehenden Wartungsmaßnahme.	Keine
Benachrichtigung	RDS-EVENT-0044	*message*	Dies ist eine vom Betreiber ausgegebene Benachrichtigung. Weitere Informationen finden Sie in der Ereignismeldung.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung	RDS-EVENT-0048	Das Upgrade der Datenbank-Engine verzögert sich, da diese Instance über Lesereplikate verfügt, die zuerst aktualisiert werden müssen.	Patchen der DB-Instance wurde verzögert.
Benachrichtigung	RDS-EVENT-0087	Die DB-Instance wurde gestoppt.	Keine
Benachrichtigung	RDS-EVENT-0088	Die DB-Instance wurde gestartet.	Keine
Benachrichtigung	RDS-EREIGNIS-0365	Zeitzonendateien wurden aktualisiert. Starten Sie Ihre RDS-Instance neu, damit die Änderungen übernommen werden.	Keine
Benachrichtigung, Serverless	RDS-EREIGNIS-0370	Es wurde eine Pause für die DB-Instance initiiert.	Ein neuer Versuch, eine inaktive DB-Instance von Aurora Serverless v2 anzuhalten, wurde gestartet.
Benachrichtigung, Serverless	RDS-EREIGNIS-0371	Die Pause wurde für den DB-Cluster abgebrochen.	Der Versuch, eine inaktive DB-Instance von Aurora Serverless v2 anzuhalten, war aufgrund des Workloads nicht erfolgreich.
Benachrichtigung, Serverless	RDS-EREIGNIS-0372	Die DB-Instance wurde erfolgreich angehalten.	Die DB-Instance von Aurora Serverless v2 wurde angehalten.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung, Serverless	RDS-EREIGNIS-0373	Die Fortsetzung der DB-Instance wurde initiiert.	Die DB-Instance von Aurora Serverless v2 wurde aufgrund eines neuen Workloads oder aufgrund von Verwaltungs- oder Wartungsaktivitäten fortgesetzt.
Benachrichtigung, Serverless	RDS-EREIGNIS-0374	Die DB-Instance wurde erfolgreich fortgesetzt.	Die DB-Instance von Aurora Serverless v2 wurde fortgesetzt.
Benachrichtigung	RDS-EREIGNIS-0385	Die Cluster-Topologie wurde aktualisiert.	Es liegen DNS-Änderungen am DB-Cluster für die DB-Instance vor. Dies gilt auch für Fälle, in denen neue DB-Instances hinzugefügt oder gelöscht werden oder ein Failover auftritt.
Benachrichtigung, globale Datenbank	RDS-EREIGNIS-0390	Der Versuch, Schreibvorgänge für den DB-Cluster *cluster_id* in Region zu blockieren, war erfolgreich. *region_id*	Aurora begann, Schreibvorgänge auf der Speicherebene zu blockieren, um die Umstellung oder das Failover einer globalen Aurora-Datenbank vorzubereiten.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung, globale Datenbank	RDS-EVENT-0391	Beim Versuch, Schreibvorgänge für den DB-Cluster *cluster_id* in der Region zu blockieren, wurde das Zeitlimit überschritten. *region_id*	Aurora konnte keine Schreibvorgänge auf der Speicherebene blockieren, um die Umstellung oder das Failover einer globalen Aurora-Datenbank vorzubereiten. Die Umstellung oder das Failover wird fortgesetzt, aber Sie müssen möglicherweise kürzlich geschriebene Daten aus dem Snapshot des ursprünglichen primären Clusters wiederherstellen.
Read Replica	RDS-EVENT-0045	Die Replikation wurde gestoppt.	Diese Meldung wird angezeigt, wenn während der Replikation ein Fehler auftritt. Weitere Informationen zur Bestimmung des Fehlertyps finden Sie unter Troubleshooting a MySQL read replica problem.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Read Replica	RDS-EVENT-0046	Die Replikation für das Lesereplikat wurde fortgesetzt.	Diese Meldung wird beim ersten Erstellen eines Lesereplikats sowie als Überwachungsmeldung zur Bestätigung der ordnungsgemäßen Replikationsausführung angezeigt. Falls diese Meldung auf die Benachrichtigung `RDS-EVENT-0045` folgt, wurde die Replikation (nach einem Fehler oder nachdem sie gestoppt wurde) fortgesetzt.
Read Replica	RDS-EVENT-0057	Das Replikationsstreaming wurde beendet.	Keine
Wiederherstellung	RDS-EVENT-0020	Wiederherstellung der DB-Instance wurde gestartet. Die Wiederherstellungsdauer variiert je nach zu wiederherstellender Datenmenge.	Keine
Wiederherstellung	RDS-EVENT-0021	Wiederherstellung der DB-Instance ist abgeschlossen.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wiederherstellung	RDS-EVENT-0023	Auftretende Snapshot-Anfrage:. *message*	Ein manuelles Backup wurde angefordert, jedoch erstellt Amazon RDS gerade einen DB-Snapshot. Senden Sie die Anforderung erneut, nachdem Amazon RDS den DB-Snapshot abgeschlossen hat.
Wiederherstellung	RDS-EVENT-0052	Die Wiederherstellung der Multi-AZ-Instance wurde gestartet.	Die Wiederherstellungsdauer variiert je nach zu wiederherstellender Datenmenge.
Wiederherstellung	RDS-EVENT-0053	Die Wiederherstellung der Multi-AZ-Instance wurde abgeschlossen. Failover oder Aktivierung steht noch aus.	Diese Meldung weist darauf hin, dass Amazon RDS Ihre DB-Instance darauf vorbereitet hat, bei Bedarf ein Failover zur sekundären Instance zu initiieren.
Wiederherstellung	RDS-EVENT-0361	Die Wiederherstellung der Standby-DB-Instance wurde gestartet.	Die Standby-DB-Instance wird während des Wiederherstellungsprozesses neu erstellt. Die Datenbankleistung wird während des Wiederherstellungsprozesses beeinträchtigt.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wiederherstellung	RDS-EREIGNIS-0362	Die Wiederherstellung der Standby-DB-Instance wurde abgeschlossen.	Die Standby-DB-Instance wird während des Wiederherstellungsprozesses neu erstellt. Die Datenbankleistung wird während des Wiederherstellungsprozesses beeinträchtigt.
Wiederherstellung	RDS-EVENT-0019	Von der DB-Instance auf wiederhergestellt. *name* *name*	Die DB-Instance wurde aus einem point-in-time Backup wiederhergestellt.
Ausführen von Sicherheits-Patches	RDS-EVENT-0230	Für Ihre DB-Instance ist ein System-Update verfügbar. Informationen zum Anwenden von Updates finden Sie unter „Warten einer DB-Instance“ im RDS-Benutzerhandbuch.	Ein neues Unterversions-Patch des Betriebssystems ist für Ihre DB-Instance verfügbar. Informationen zum Anwenden von Aktualisierungen finden Sie unter Betriebssystemupdates für Aurora-DB-Cluster.
Wartung	RDS-EVENT-0425	Amazon RDS kann das Betriebssystem-Upgrade nicht durchführen, da in den angegebenen Subnetzen keine IP-Adressen verfügbar sind. Wählen Sie Subnetze mit verfügbaren IP-Adressen aus und versuchen Sie es erneut.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wartung	RDS-EREIGNIS-0429	Amazon RDS kann das Betriebssystem-Upgrade nicht durchführen, da für den *type* Instance-Typ in der *zone* Availability Zone nicht genügend Kapazität verfügbar ist	Keine
Wartung	RDS-EVENT-0501	Das Serverzertifikat der Amazon RDS-DB-Instance muss im Rahmen einer ausstehenden Wartungsaktion rotiert werden.	Das Serverzertifikat der DB-Instance muss im Rahmen einer ausstehenden Wartungsaktion rotiert werden. Amazon RDS startet Ihre Datenbank während dieser Wartung neu, um die Zertifikatsrotation abzuschließen. Um diese Wartung zu planen, gehen Sie zur Registerkarte Wartung und Backups und wählen Sie „Jetzt anwenden“ oder „Für das nächste Wartungsfenster planen“. Wenn die Änderung nicht geplant ist, wendet Amazon RDS sie automatisch an dem in Ihrer Wartungsaktion angegebenen Datum für die auto Anwendung in Ihrem Wartungsfenster an.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wartung	RDS-EVENT-052	Amazon RDS RDS hat für das nächste Wartungsfenster eine Rotation der Serverzertifikate für die DB-Instance der DB-Instance geplant. Diese Wartung erfordert einen Neustart der Datenbank.	Keine

DB-Parametergruppenereignisse

In der folgenden Tabelle werden die Ereigniskategorie und die Ereignisse für den Quelltyp "DB-Parametergruppe" aufgeführt.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0037	Der Parameter wurde *name* auf die Methode „*value*Mit Anwenden“ aktualisiert*method*.	Keine

DB-Sicherheitsgruppenereignisse

In der folgenden Tabelle werden die Ereigniskategorie und die Ereignisse für den Quelltyp "DB-Sicherheitsgruppe" aufgeführt.

> **Note**
>
> DB-Sicherheitsgruppen sind Ressourcen für EC2 -Classic. EC2-Classic wurde am 15. August 2022 eingestellt. Wenn Sie noch nicht von EC2 -Classic zu einer VPC migriert haben, empfehlen wir Ihnen, so bald wie möglich zu migrieren. Weitere Informationen finden Sie unter Migration von EC2 -Classic zu einer VPC im EC2 Amazon-Benutzerhandbuch und im Blog EC2-Classic Networking is Retiring — So bereiten Sie sich vor.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0038	Die Änderung wurde auf die Sicherheitsgruppe angewendet.	Keine
Ausfall	RDS-EVENT-0039	Autorisierung widerrufen als. *user*	Die Sicherheitsgruppe, deren Eigentümer *user* ist, existiert nicht. Die Autorisierung für die Sicherheitsgruppe wurde widerrufen, weil sie ungültig ist.

DB-Shard-Gruppenereignisse

In der folgenden Tabelle werden die Ereigniskategorie und die Ereignisse für den Quelltyp „DB-Shard-Gruppe“ aufgeführt.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0324	Datenladeauftrag fehlgeschlagen.	Der Datenladeauftrag ist aus dem in der Fehlermeldung angegebenen Grund fehlgeschlagen.
Wartung	RDS-EREIGNIS-0302	Ihre Aktion ist erforderlich, um einen ausstehenden Split-Shard-Job *job_ID* für die Subcluster-ID in der DB-Shard-Gruppe abzuschließen. *number name* Rufen Sie SQL auf, um den Vorgang abzuschließen:	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
		`SELECT rds_auror a.limitless_finali ze_split_ shard( job_ID);`	
Wartung	RDS-EVENT-0303	Finalize für den Split-Sha rd-Job *job_ID* wurde für die Subcluster-ID in der DB-Shard-Gruppe gestartet . *number name*	Keine
Wartung	RDS-EVENT-0304	Der Split-Shard-Auftrag für die *job_ID* Subcluste r-ID in der DB-Shard-Gruppe wurde erfolgreich abgeschlossen. *number name* Ein neuer Shard mit der Subcluster-ID *number* wurde der DB-Shard-Gruppe hinzugefügt. *name*	Keine
Wartung	RDS-EVENT-0305	Der Split-Shard-Job *job_ID* ist für die Subcluster-ID in der DB-Shard-Gruppe fehlgesch lagen. *number name*	Keine
Wartung	RDS-EVENT-0366	Der Split-Shard-Job für die Subcluster-ID in der DB-Shard-Gruppe *job_ID* wurde gestartet. *number name*	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Wartung	RDS-EVENT-0367	Der Job zum Hinzufüge n eines Routers *job_ID* wurde in der DB-Shard-Gruppe gestartet. *name*	Keine
Wartung	RDS-EVENT-0368	Der Auftrag zum Hinzufüge n eines Routers wurde erfolgreich abgeschlossen. *job_ID* Ein neuer Router mit Subcluster-ID *number* wurde der DB-Shard-Gruppe hinzugefügt. *name*	Keine
Wartung	RDS-EVENT-0369	Der Auftrag zum Hinzufüge n eines Routers ist in der *job_ID* DB-Shard-Gruppe fehlgeschlagen. *name*	Keine
Wartung	RDS-EVENT-0394	Der Auftrag zum Hinzufüge n eines Routers *job_ID* wurde in der DB-Shard-Gruppe abgebrochen. *name*	Keine
Wartung	RDS-EVENT-0395	Der Split-Shard-Job *job_ID* wurde in der DB-Shard-Gruppe abgebroch en. *name*	Keine
Benachric htigung	RDS-EVENT-0321	Die Infrastruktur für den Datenladeauftrag wird initialisiert.	Keine
Benachric htigung	RDS-EREIG NIS-0322	Datenladeauftrag ist in Bearbeitung.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung	RDS-EREIGNIS-0323	Datenladeauftrag wurde erfolgreich abgeschlossen.	Keine
Benachrichtigung	RDS-EREIGNIS-0325	Der Datenladeauftrag wird auf Anforderung des Kunden abgebrochen.	Keine
Benachrichtigung	RDS-EREIGNIS-0326	Der Datenladeauftrag wurde abgebrochen.	Keine

RDS-Proxy-Ereignisse

Die folgende Tabelle zeigt die Ereigniskategorie und eine Liste von Ereignissen für den Fall, dass ein RDS-Proxy der Quelltyp ist.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0204	RDS-modifizierter DB-Proxy*name*.	Keine
Konfigurationsänderung	RDS-EVENT-0207	RDS hat den Endpunkt des DB-Proxys geändert*name*.	Keine
Konfigurationsänderung	RDS-EVENT-0213	RDS hat das Hinzufügen der DB-Instance erkannt und sie automatisch der Zielgruppe des DB-Proxys hinzugefügt*name*.	Keine
Konfigurationsänderung	RDS-EVENT-0214	RDS hat das Löschen der DB-Instance erkannt *name* und sie automatisch aus	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
		name der Zielgruppe des DB-Proxys entfernt*name*.	
Konfigurationsänderung	RDS-EVENT-0215	RDS hat das Löschen des DB-Clusters erkannt *name* und ihn automatisch aus *name* der Zielgruppe des DB-Proxys entfernt*name*.	Keine
Erstellung	RDS-EVENT-0203	RDS hat einen DB-Proxy erstellt*name*.	Keine
Erstellung	RDS-EVENT-0206	RDS hat einen Endpunkt *name* für den DB-Proxy erstellt*name*.	Keine
Löschung	RDS-EVENT-0205	RDS hat den DB-Proxy gelöscht*name*.	Keine
Löschung	RDS-EVENT-0208	RDS hat den Endpunkt *name* für den DB-Proxy gelöscht*name*.	Keine
Ausfall	RDS-EVENT-0243	RDS konnte keine Kapazität für den Proxy bereitstellen, *name* da in Ihren Subnetzen nicht genügend IP-Adressen verfügbar sind:*name*. Stellen Sie sicher, dass Ihre Subnetze die Mindestanzahl unbenutzter IP-Adressen aufweisen, wie in der RDS-Proxy-Dokumentation empfohlen.	Informationen zur Bestimmung der empfohlenen Anzahl für Ihre Instance-Klasse finden Sie unter Planen der Kapazität von IP-Adressen.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0275	RDS hat einige Verbindungen zum DB-Proxy gedrosselt. *name* Die Anzahl der gleichzeitigen Verbindungsanforderungen vom Client zum Proxy hat den Grenzwert überschritten.	Keine

Blau/Grün-Bereitstellungsereignisse

Die folgende Tabelle zeigt die Ereigniskategorie und eine Liste der Ereignisse, wenn es sich bei einer blue/green Bereitstellung um den Quelltyp handelt.

Weitere Informationen zu blue/green Bereitstellungen finden Sie unterVerwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

Kategorie	Amazon-RDS-Ereignis-ID	Fehlermeldung	Hinweise
Erstellung	RDS-EVENT-0244	Die Blau/Grün-Bereitstellungsaufgaben sind abgeschlossen. Sie können weitere Änderungen an den Datenbanken der grünen Umgebung vornehmen oder die Bereitstellung umstellen.	Keine
Ausfall	RDS-EVENT-0245	Die Erstellung der blue/green Bereitstellung ist fehlgeschlagen, weil*reason*.	Keine
Löschung	RDS-EVENT-0246	Die Blau/Grün-Bereitstellung wurde gelöscht.	Keine

Kategorie	Amazon-RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung	RDS-EVENT-0247	Der Switchover von *blue* zu *green* wurde gestartet.	Keine
Benachrichtigung	RDS-EVENT-0248	Der Switchover wurde bei der Bereitstellung abgeschlossen. blue/green	Keine
Ausfall	RDS-EVENT-0249	Der Switchover wurde bei der Bereitstellung abgebrochen. blue/green	Keine
Benachrichtigung	RDS-EVENT-0259	Switchover vom zum Start-DB-Cluster. *blue* *green*	Keine
Benachrichtigung	RDS-EVENT-0260	Der Switchover vom Replica-DB-Cluster auf abgeschlossen. *blue* *green* Umbenannt in *blue* und in. *blue-old* *green* *blue*	Keine
Ausfall	RDS-EVENT-0261	Der Switchover vom *blue* auf *green* wurde aufgrund von abgebrochen. *reason*	Keine
Benachrichtigung	RDS-EVENT-0311	Die Sequenzsynchronisierung für den Switchover des DB-Clusters auf wurde eingeleitet. *blue* *green* Die Umstellung kann bei der Verwendung von Sequenzen zu längeren Ausfallzeiten führen.	Keine

Kategorie	Amazon-RDS-Ereignis-ID	Fehlermeldung	Hinweise
Benachrichtigung	RDS-EVENT-0312	Die Sequenzsynchronisierung für den Switchover des DB-Clusters *blue* auf *green* ist abgeschlossen.	Keine
Ausfall	RDS-EVENT-0314	Die Sequenzsynchronisierung für den Switchover des DB-Clusters *blue* auf *green* wurde abgebrochen, da die Sequenzen nicht synchronisiert werden konnten.	Keine
Benachrichtigung	RDS-EREIGNIS-0409	*message*	Keine

Überwachen von Amazon Aurora-Protokolldateien

Jede RDS-Datenbank-Engine generiert Protokolle, auf die Sie für die Überwachung und Fehlerbehebung zugreifen können. Der Typ der Protokolle hängt von Ihrer Datenbank-Engine ab.

Sie können auf Datenbankprotokolle für DB-Instance mithilfe der AWS-Managementkonsole, der AWS Command Line Interface (AWS CLI) oder der Amazon-RDS-API zugreifen. Sie können Transaktionsprotokolle nicht anzeigen, ansehen oder herunterladen.

> **Note**
>
> In einigen Fällen enthalten Protokolle versteckte Daten. Daher kann es sein, dass das AWS-Managementkonsole den Inhalt in einer Protokolldatei anzeigt, aber die Protokolldatei ist möglicherweise leer, wenn Sie sie herunterladen.

Themen

- Anzeigen und Auflisten von Datenbank-Protokolldateien
- Herunterladen einer Datenbank-Protokolldatei
- Überwachen einer Datenbank-Protokolldatei
- Veröffentlichen von Datenbankprotokollen in Amazon CloudWatch Logs
- Lesen der Protokolldateiinhalte mit REST
- Aurora-MySQL-Datenbank-Protokolldateien
- Datenbank-Protokolldateien von Aurora PostgreSQL

Anzeigen und Auflisten von Datenbank-Protokolldateien

Sie können Datenbank-Protokolldateien für Ihre Amazon-Aurora-DB-Engine mithilfe der AWS-Managementkonsole anzeigen. Sie können auflisten, welche Protokolldateien zum Herunterladen bzw. Überwachen verfügbar sind, indem Sie die AWS CLI oder die Amazon-RDS-API verwenden.

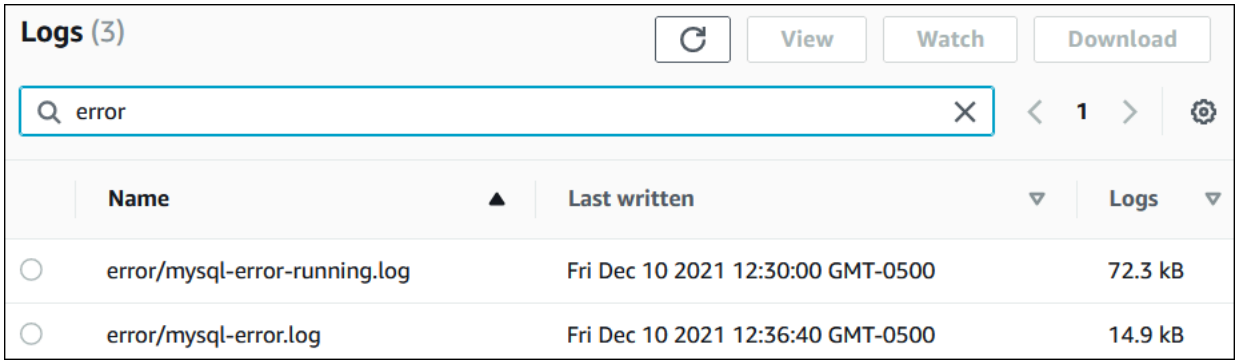
> **Note**
>
> Sie können die Protokolldateien für Aurora Serverless v1-DB-Cluster in der RDS-Konsole nicht anzeigen. Sie können sie jedoch in der Amazon-CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/ ansehen.

Konsole

So zeigen Sie eine Datenbank-Protokolldatei an

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Namen der DB-Instance, welche die anzuzeigende Protokolldatei enthält.
4. Wählen Sie die Registerkarte Logs & events (Protokolle und Ereignisse).
5. Scrollen Sie nach unten bis zum Abschnitt Protokolle.
6. (Optional) Geben Sie einen Suchbegriff ein, um Ihre Ergebnisse zu filtern.

   Im folgenden Beispiel werden Protokolle aufgeführt, die nach dem Text **`error`** gefiltert wurden.



7. Wählen Sie das gewünschte Protokoll und dann View (Anzeigen) aus.

AWS CLI

Um die verfügbaren Datenbank-Protokolldateien für eine DB-Instance aufzulisten, verwenden Sie den AWS CLI-Befehl `describe-db-log-files`.

Das folgende Beispiel gibt eine Liste von Protokolldateien für eine DB-Instance namens zurüc `my-db-instance`.

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS-API

Um die verfügbaren Datenbank-Protokolldateien für eine DB-Instance aufzulisten, verwenden Sie die Amazon-RDS-API-Aktion `DescribeDBLogFiles`.

Herunterladen einer Datenbank-Protokolldatei

Sie können die AWS-Managementkonsole, AWS CLI oder API zum Herunterladen einer Datenbank-Protokolldatei verwenden.

Konsole

So laden Sie eine Datenbank-Protokolldatei herunter

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Namen der DB-Instance, welche die anzuzeigende Protokolldatei enthält.
4. Wählen Sie die Registerkarte Logs & events (Protokolle und Ereignisse).
5. Scrollen Sie nach unten bis zum Abschnitt Protokolle.
6. Klicken Sie im Bereich Protokolle auf die Schaltfläche neben dem gewünschten Protokoll und wählen Sie Herunterladen.
7. Öffnen Sie das Kontextmenü (rechte Maustaste) für den bereitgestellten Link und wählen Sie Save Link As (Link speichern unter) aus. Geben Sie den Speicherort für die Protokolldatei ein und klicken Sie dann auf Speichern.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `download-db-log-file-portion`, um eine Datenbank-Protokolldatei herunterzuladen. Standardmäßig lädt dieser Befehl nur den neuesten Teil einer

Protokolldatei herunter. Sie können jedoch eine ganze Datei herunterladen, indem Sie den Parameter angebe `--starting-token 0`.

Das folgende Beispiel zeigt, wie man den Inhalt einer Protokolldatei namens log/ERROR.4 herunterlädt und in einer lokalen Datei namens errorlog.txt speichert.

Example

Für Linux, macOS oder Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

Für Windows:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS-API

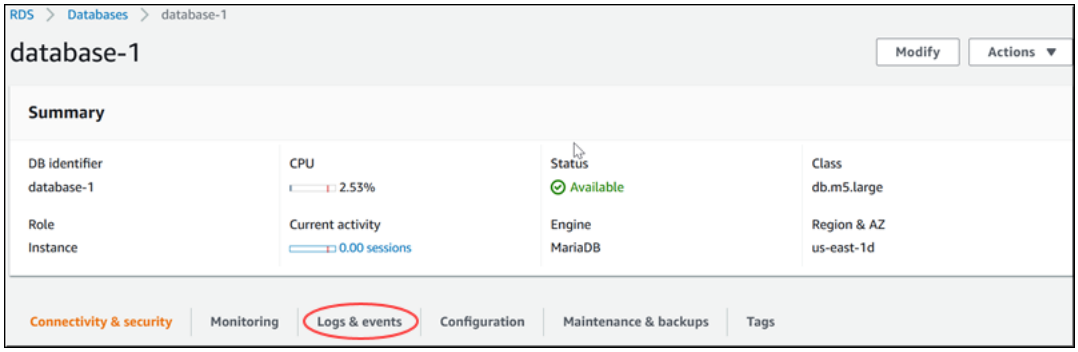
Verwenden Sie die Amazon-RDS-API-Aktion `DownloadDBLogFilePortion`, um eine Datenbank-Protokolldatei herunterzuladen.

Überwachen einer Datenbank-Protokolldatei

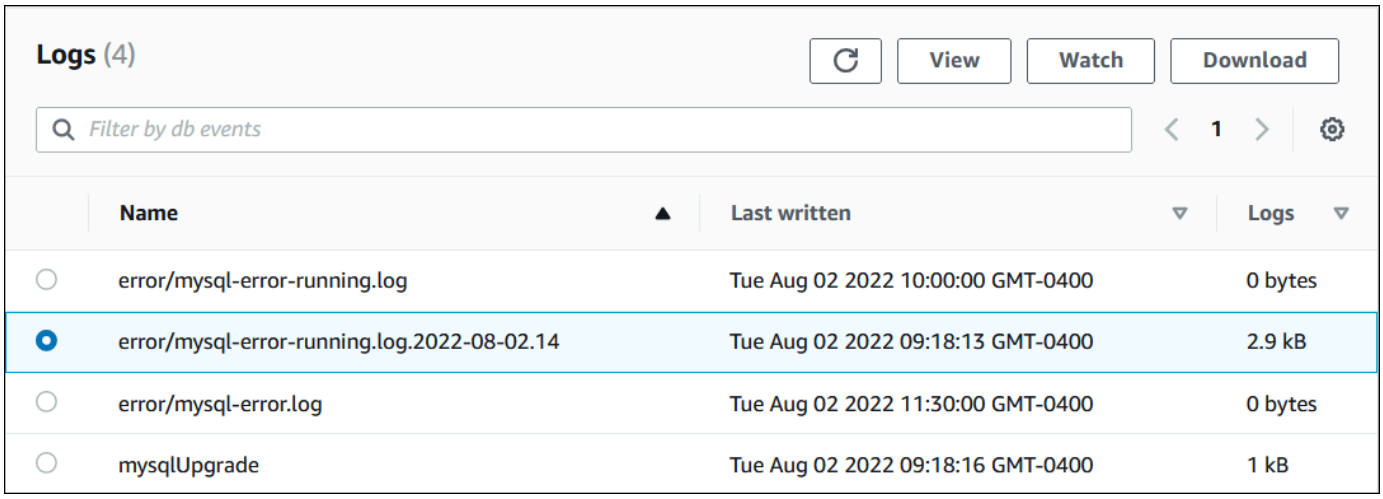
Das Überwachen einer Datenbank-Protokolldatei entspricht dem Abrufen der Datei auf einem UNIX- oder Linux-System. Sie können eine Protokolldatei mithilfe der AWS-Managementkonsole überwachen. RDS aktualisiert das Ende des Protokolls alle 5 Sekunden.

So überwachen Sie eine Datenbank-Protokolldatei

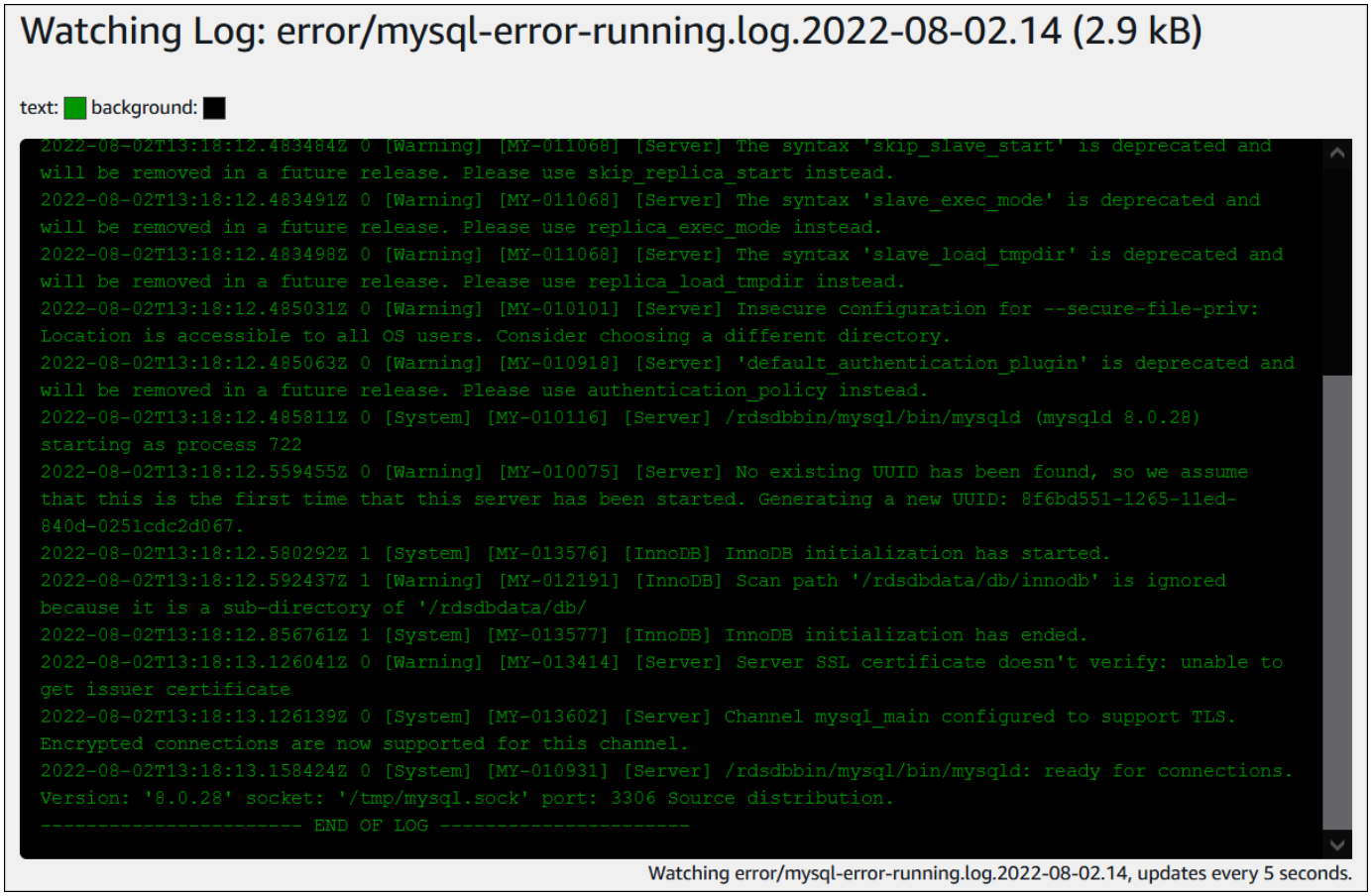
1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Namen der DB-Instance, welche die anzuzeigende Protokolldatei enthält.
4. Wählen Sie die Registerkarte Logs & events (Protokolle und Ereignisse).



5. Wählen Sie im Abschnitt Protokolle eine Protokolldatei und wählen Sie dann Watch (Beobachten).



RDS zeigt das Ende des Protokolls an, wie im folgenden MySQL-Beispiel.



Veröffentlichen von Datenbankprotokollen in Amazon CloudWatch Logs

In einer On-Premises-Datenbank befinden sich die Datenbankprotokolle im Dateisystem. Amazon RDS bietet keinen Host-Zugriff auf die Datenbankprotokolle im Dateisystem Ihres DB-Clusters. Aus diesem Grund ermöglicht Amazon RDS das Exportieren von Datenbankprotokollen in Amazon CloudWatch Logs. Mit CloudWatch Logs können Sie Echtzeitanalysen der Protokolldaten durchführen. Sie können die Daten auch in einem Speicher mit hoher Beständigkeit speichern und mit dem CloudWatch-Logs-Agenten verwalten.

Themen

- Überblick über die RDS-Integration mit CloudWatch Logs
- Entscheiden, welche Protokolle in CloudWatch Logs veröffentlicht werden
- Angeben der Protokolle, die in CloudWatch Logs veröffentlicht werden sollen
- Suchen und Filtern Ihrer Protokolle in CloudWatch Logs

Überblick über die RDS-Integration mit CloudWatch Logs

In CloudWatch Logs ist ein Protokollstream eine Abfolge von Protokollereignissen, die dieselbe Quelle nutzen. Jede separate Quelle für Protokolle in CloudWatch Logs bildet einen separaten Protokollstream. Eine Protokollgruppe ist eine Gruppe von Protokollstreams, die dieselben Einstellungen für die Aufbewahrung, Überwachung und Zugriffskontrolle besitzen.

Amazon Aurora streamt kontinuierlich die Protokolldatensätze Ihres DB-Clusters in eine Protokollgruppe. Sie haben z. B. eine Protokollgruppe `/aws/rds/cluster/`*`cluster_name`*`/`*`log_type`* für jeden Protokolltyp, den Sie veröffentlichen. Diese Protokollgruppe befindet sich in derselben AWS-Region wie die Datenbank-Instance, die das Protokoll erzeugt.

AWS bewahrt Protokolldaten, die in CloudWatch Logs veröffentlicht wurden, auf unbegrenzte Dauer auf, wenn keine Aufbewahrungsfrist festgelegt wird. Weitere Informationen finden Sie unter Ändern der Aufbewahrungszeit von Protokolldaten in CloudWatch Logs.

Entscheiden, welche Protokolle in CloudWatch Logs veröffentlicht werden

Jede RDS-Datenbank-Engine unterstützt einen eigene Gruppe von Protokollen. Lesen Sie die folgenden Themen, um mehr über die Optionen für Ihre Datenbank-Engine zu erfahren:

- the section called “Veröffentlichen von Aurora MySQL-Protokollen in CloudWatch Logs”
- the section called “Veröffentlichen von Aurora-PostgreSQL-Protokollen in CloudWatch Logs”

Angeben der Protokolle, die in CloudWatch Logs veröffentlicht werden sollen

Sie geben an, welche Protokolle in der Konsole veröffentlicht werden sollen. Stellen Sie sicher, dass eine servicegebundene Rolle in AWS Identity and Access Management (IAM) vorhanden ist. Weitere Informationen zu serviceverknüpften Rollen finden Sie unter Verwenden von serviceverknüpften Rollen für Amazon Aurora.

So geben Sie die Protokolle an, die veröffentlicht werden sollen

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Führen Sie eine der folgenden Aufgaben aus:
   - Wählen Sie Create database (Datenbank erstellen) aus.

- Wählen Sie eine Datenbank in der Liste und dann Modify (Ändern) aus.

4. Wählen Sie in Logs exports (Protokollexporte) die Protokolle aus, die veröffentlicht werden sollen.

   Das folgende Beispiel spezifiziert das Auditprotokoll, die Fehlerprotokolle, das allgemeine Protokoll, das Instance-Protokoll, das Fehlerprotokoll für die IAM-Datenbankauthentifizierung und das Protokoll für langsame Abfragen für einen DB-Cluster von Aurora MySQL .

Suchen und Filtern Ihrer Protokolle in CloudWatch Logs

Sie können mithilfe der Konsole von CloudWatch Logs nach den Protokolleinträgen suchen, die ein bestimmtes Kriterium erfüllen. Sie können auf die Protokolle entweder über die RDS-Konsole, die Sie zur CloudWatch-Logs-Konsole führt, oder direkt über die CloudWatch-Logs-Konsole zugreifen.

So suchen Sie Ihre RDS-Protokolle mithilfe der RDS-Konsole

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie ein DB-Cluster oder eine DB-Instance aus.
4. Wählen Sie Konfiguration.
5. Wählen Sie unter Published logs (Veröffentlichte Protokolle) das Datenbankprotokoll aus, das Sie anzeigen möchten.

So suchen Sie Ihre RDS-Protokolle mithilfe der CloudWatch-Logs-Konsole

1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie im Navigationsbereich Log groups (Protokollgruppen) aus.
3. Geben Sie im Filterfeld **`/aws/rds`** ein.
4. Wählen Sie für Log Groups den Namen der Protokollgruppe mit dem Protokoll-Stream aus, nach dem gesucht werden soll.
5. Wählen Sie für Log Streams den Namen des zu suchenden Protokoll-Streams.
6. Geben Sie unter Protokollereignisse die zu verwendende Filtersyntax ein.

Weitere Informationen finden Sie unter Suchen und Filtern von Protokolldaten im Benutzerhandbuch von Amazon CloudWatch Logs. Ein Blog-Tutorial zur Überwachung von RDS-Protokollen finden Sie

unter Erstellen Sie proaktive Datenbanküberwachung für Amazon RDS mit Amazon CloudWatch Logs, AWS Lambda und Amazon SNS.

Lesen der Protokolldateiinhalte mit REST

Amazon RDS bietet einen REST-Endpunkt, der den Zugriff auf die Protokolldateien der DB-Instance ermöglicht. Dies ist nützlich, wenn Sie eine Anwendung schreiben müssen, um Amazon-RDS-Protokolldateiinhalte zu streamen.

Die Syntax lautet wie folgt:

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
Content-type: application/json
host: rds.region.amazonaws.com
```

Die folgenden Parameter sind erforderlich:

- *DBInstanceIdentifier*: der Name der DB-Instance, welche die Protokolldatei enthält, die Sie herunterladen möchten.
- *LogFileName*: der Name der Protokolldatei, die heruntergeladen werden soll.

Die Antwort enthält die Inhalte der angeforderten Protokolldatei als Stream zurück.

Im folgenden Beispiel wird die Protokolldatei namens log/ERROR.6 für die DB-Instance namens sample-sql in der Region us-west-2 heruntergeladen.

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

Wenn Sie eine nicht vorhandene DB-Instance angeben, besteht die Antwort aus dem folgenden Fehler:

- `DBInstanceNotFound`: *`DBInstanceIdentifier`* bezieht sich nicht auf eine vorhandene DB-Instance. (HTTP-Statuscode: 404)

Aurora-MySQL-Datenbank-Protokolldateien

Sie können die Aurora-MySQL-Protokolle direkt über die Amazon-RDS-Konsole, Amazon RDS API, AWS CLI oder AWS-SDKs überwachen. Sie können auf MySQL-Protokolle auch direkt zugreifen, indem Sie die Protokolle in eine Datenbank-Tabelle in der Hauptdatenbank weiterleiten und diese Tabelle abfragen. Mit dem Dienstprogramm "mysqlbinlog" können Sie ein binäres Protokoll herunterladen.

Weitere Informationen zum Anzeigen und Herunterladen von dateibasierten Datenbankprotokollen finden Sie unter Überwachen von Amazon Aurora-Protokolldateien.

Themen

- Überblick über Aurora-MySQL-Datenbankprotokolle
- Senden der Protokollausgabe von Aurora MySQL an Tabellen
- Konfigurieren der Aurora MySQL-Binärprotokollierung für Single-AZ-Datenbanken
- Zugriff auf MySQL-Binärprotokolle

Überblick über Aurora-MySQL-Datenbankprotokolle

Sie können die folgenden Arten von Aurora-MySQL-Protokolldateien überwachen:

- Fehler-log
- Slow-Query-Protokoll
- Allgemeines Protokoll
- Prüfungsprotokoll
- Instance-Protokoll
- Fehlerprotokoll zur IAM-Datenbankauthentifizierung

Das Aurora-MySQL-Fehlerprotokoll wird standardmäßig generiert. Sie können die langsamen Abfrage- und allgemeinen Protokolle generieren, indem Sie Parameter in Ihrer DB-Parametergruppe festlegen.

Themen

- Aurora-MySQL-Fehlerprotokolle
- Aurora-MySQL-Protokolle für langsame Abfragen und allgemeine Protokolle

- Aurora-MySQL-Prüfprotokoll
- Instance-Protokoll von Aurora MySQL
- Protokollrotation und -aufbewahrung für Aurora MySQL
- Veröffentlichen von Aurora MySQL-Protokollen in Amazon CloudWatch Logs

Aurora-MySQL-Fehlerprotokolle

Aurora MySQL schreibt Fehler in die `mysql-error.log`-Datei. An den Namen jeder Protokolldatei wird die Stunde ihrer Erstellung (in UTC) angefügt. Die Protokolldateien verfügen auch über einen Zeitstempel, anhand dessen Sie feststellen können, wann die Protokolleinträge geschrieben wurden.

Aurora MySQL schreibt das Fehlerprotokoll nur beim Startup, Herunterfahren und beim Auftreten von Fehlern. Eine DB-Instance kann Stunden oder Tage lang laufen, ohne dass neue Einträge in das Fehlerprotokoll geschrieben werden. Wenn Sie keine neuen Einträge sehen, sind im Server keine Fehler aufgetreten, die zu einem Eintrag in das Protokoll führen würden.

Konstruktionsbedingt werden die Fehlerprotokolle gefiltert, sodass nur unerwartete Ereignisse wie Fehler angezeigt werden. Die Fehlerprotokolle enthalten jedoch auch einige zusätzliche Datenbankinformationen, z. B. den Abfragefortschritt, die nicht angezeigt werden. Daher kann die Größe der Fehlerprotokolle auch ohne tatsächliche Fehler aufgrund laufender Datenbankaktivitäten zunehmen. Und auch wenn für die Fehlerprotokolle in der AWS-Managementkonsole ggf. eine bestimmte Größe in Byte oder Kilobyte angezeigt wird, enthalten die Protokolle möglicherweise 0 Byte, wenn Sie sie herunterladen.

Aurora MySQL schreibt `mysql-error.log` alle 5 Minuten auf die Festplatte. Es fügt den Inhalt des Protokolls `mysql-error-running.log` an.

Aurora MySQL rotiert die Datei `mysql-error-running.log` stündlich.

> **Note**
>
> Der Aufbewahrungszeitraum für das Protokoll ist zwischen Amazon RDS und unterschiedlich Aurora.

Aurora-MySQL-Protokolle für langsame Abfragen und allgemeine Protokolle

Das Slow-Query-Protokoll von Aurora MySQL und das allgemeine Protokoll können in eine Datei oder in eine Datenbanktabelle geschrieben werden. Legen Sie dazu die Parameter in Ihrer DB-

Parametergruppe fest. Weitere Informationen zum Erstellen und Ändern einer DB-Parametergruppe finden Sie unter Parametergruppen für Amazon Aurora. Sie müssen diese Parameter festlegen, bevor Sie das Slow-Query-Protokoll oder das allgemeine Protokoll in der Amazon-RDS-Konsole bzw. mithilfe von Amazon-RDS-API, Amazon-RDS-CLI oder AWS SDKs sehen können.

Sie können Aurora-MySQL-Protokolle mithilfe der Parameter in dieser Liste kontrollieren:

- `slow_query_log` Um das Slow-Query-Protokoll zu erstellen, auf 1 setzen. Der Standardwert ist 0.
- `general_log` Um das allgemeine Protokoll zu erstellen, auf 1 setzen. Der Standardwert ist 0.
- `long_query_time`: Damit vermieden wird, dass schnell ausgeführte Abfragen im Slow-Query-Protokoll aufgenommen werden, legen Sie die kürzeste Laufzeit für eine zu protokollierende Abfrage in Sekunden fest. Der Standardwert liegt bei 10 Sekunden, der Mindestwert bei 0. Wenn log_output = FILE, können Sie einen Gleitkommawert angeben, der die Mikrosekundenauflösung festlegt. Wenn log_output = TABLE, können Sie einen Ganzzahlwert angeben, der die Sekundenauflösung festlegt. Nur Abfragen, deren Laufzeit den `long_query_time`-Wert übersteigt, werden im Protokoll aufgenommen. Wenn Sie beispielsweise `long_query_time` auf 0,1 setzen, verhindert dies Einträge von allen Abfragen, die weniger als 100 Millisekunden lang ausgeführt werden.
- `log_queries_not_using_indexes`: Um alle Abfragen, die keinen Index für das Slow-Query-Protokoll verwenden im Protokoll aufzunehmen, auf 1 setzen. Abfragen, die keinen Index verwenden, werden protokolliert, auch wenn ihre Laufzeit niedriger als der Wert des Parameters `long_query_time` ist. Der Standardwert ist 0.
- `log_output` *`option`*: Sie können eine der folgenden Optionen für den `log_output`-Parameter festlegen.
  - TABLE – schreibt allgemeine Abfragen in die `mysql.general_log`-Tabelle und langsame Abfragen in die `mysql.slow_log`-Tabelle.
  - FILE– schreibt Protokolle allgemeiner und langsamer Abfragen in das Dateisystem.
  - NONE– Die Protokollierung ist deaktiviert.

  Bei Aurora-MySQL-Version 2 und 3 ist der Standardwert für `log_output` `FILE`.

Für die Anzeige von langsamen Abfragedaten in Amazon CloudWatch Logs müssen die folgenden Bedingungen erfüllt sein:

- CloudWatch Logs muss so konfiguriert sein, dass es langsame Abfrageprotokolle einschließt.
- `slow_query_log` muss aktiviert sein.

- muss `log_output` auf `FILE` festgelegt sein.
- Die Abfrage muss länger dauern als die für `long_query_time` konfigurierte Zeit.

Weitere Informationen zu den Slow-Query- und allgemeinen Protokollen finden Sie in den folgenden Themen in der MySQL-Dokumentation:

- Das Slow-Query-Protokoll
- Das allgemeine Abfrageprotokoll

Aurora-MySQL-Prüfprotokoll

Die Auditprotokollierung für Aurora MySQL heißt Advanced Auditing. Zum Aktivieren von Advanced Auditing legen Sie bestimmte DB-Cluster-Parameter fest. Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.

Instance-Protokoll von Aurora MySQL

Aurora erstellt eine separate Protokolldatei für DB-Instances, bei denen die Auto-Pause-Funktion aktiviert ist. In der Datei instance.log werden alle Gründe aufgezeichnet, warum diese DB-Instances nicht wie erwartet pausiert werden konnten. Weitere Informationen zum Verhalten von Instance-Protokolldateien und zur Auto-Pause-Funktion von Aurora finden Sie unter Monitoring Aurora Serverless v2 pause and resume activity.

Protokollrotation und -aufbewahrung für Aurora MySQL

Wenn die Protokollierung aktiviert ist, rotiert oder löscht Amazon Aurora Protokolldateien in regelmäßigen Intervallen. Dies ist eine Vorsichtsmaßnahme, um möglichst zu vermeiden, dass eine umfangreiche Protokolldatei die Datenbanknutzung blockiert oder die Leistung beeinträchtigt. Aurora MySQL behandelt Rotation und Löschen wie folgt:

- Die Dateigrößen der Aurora-MySQL-Fehlerprotokolle sind auf maximal 15 Prozent des zugewiesenen Speicherplatzes für eine DB-Instance beschränkt. Um diesen Schwellenwert einzuhalten, werden die Protokolle automatisch stündlich gedreht. Aurora MySQL entfernt Protokolle nach 30 Tagen oder wenn 15 % des Festplattenspeichers belegt sind. Wenn die kombinierte Größe der Protokolle nach dem Löschen von alten Protokolldateien den Schwellwert überschreitet, werden die ältesten Protokolldateien gelöscht, bis die Größe den Schwellwert nicht mehr überschreitet.

- Aurora MySQL entfernt die Audit-, allgemeinen, langsamen Abfrage-Protokolle entweder nach 24 Stunden oder wenn 15 % des Speichers verbraucht wurden.
- Wenn die `FILE`-Protokollierung aktiviert ist, werden allgemeine Protokolldateien und Protokolldateien für langsame Abfragen stündlich geprüft und Protokolldateien, die älter als 24 Stunden sind, werden gelöscht. In einigen Fällen kann die Größe der verbleibenden kombinierten Protokolldatei nach dem Löschen die Schwelle von 15 % des lokalen Speicherplatzes für eine DB-Instance überschreiten. In diesen Fällen werden die ältesten Protokolldateien gelöscht, bis die Größe den Schwellenwert nicht mehr überschreitet.
- Wenn die `TABLE`-Protokollierung aktiviert ist, werden Protokolltabellen nicht rotiert oder gelöscht. Protokolltabellen werden abgeschnitten, wenn die Größe aller kombinierten Protokolle zu groß ist. Sie können die Ereigniskategorie `low storage` abonnieren, um Benachrichtigungen zu erhalten, wenn Protokolltabellen manuell rotiert oder gelöscht werden sollen, um Speicherplatz freizugeben. Weitere Informationen finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

  Sie können die `mysql.general_log`-Tabelle manuell rotieren, indem Sie die Prozedur `mysql.rds_rotate_general_log` aufrufen. Sie können die `mysql.slow_log`-Tabelle rotieren, wenn Sie die Prozedur `mysql.rds_rotate_slow_log` aufrufen.

  Wenn Sie Protokolldateien manuell rotieren, wird die aktuelle Protokolltabelle in eine Sicherungsprotokolltabelle kopiert und die Einträge in der aktuellen Protokolltabelle werden entfernt. Sofern bereits eine Sicherungsprotokolltabelle vorhanden ist, wird diese gelöscht, bevor die aktuelle Protokolltabelle ins Backup kopiert wird. Sie können die Sicherungsprotokolltabelle abfragen, wenn dies nötig ist. Die Backup-Protokolltabelle für die `mysql.general_log`-Tabelle ist als `mysql.general_log_backup` benannt. Die Backup-Protokolltabelle für die `mysql.slow_log`-Tabelle ist als `mysql.slow_log_backup` benannt.
- Die Aurora-MySQL-Überwachungsprotokolle werden rotiert, wenn die Dateigröße 100 MB erreicht, und nach 24 Stunden entfernt.
- Amazon RDS rotiert Fehlerprotokolldateien für die IAM-Datenbankauthentifizierung, die größer als 10 MB sind. Amazon RDS entfernt Fehlerprotokolldateien für die IAM-Datenbankauthentifizierung, die älter als fünf Tage oder größer als 100 MB sind.

Um mit den Protokollen über die Amazon-RDS-Konsole, Amazon-RDS-API, Amazon-RDS-CLI, oder AWS SDKs zu arbeiten, setzen Sie den Parameter `log_output` auf FILE. Diese Protokolldateien werden wie das Aurora-MySQL-Fehlerprotokoll stündlich rotiert. Die Protokolldateien, die während der vorherigen 24 Stunden angelegt wurden, werden aufbewahrt. Beachten Sie, dass der Aufbewahrungszeitraum bei Amazon RDS und Aurora jeweils unterschiedlich ist.

Veröffentlichen von Aurora MySQL-Protokollen in Amazon CloudWatch Logs

Sie können die Aurora MySQL-DB-Cluster so konfigurieren, dass Protokolldaten in einer Protokollgruppe in Amazon CloudWatch Logs veröffentlicht werden. Mit CloudWatch Logs können Sie Echtzeitanalysen der Protokolldaten durchführen, und mit CloudWatch können Sie Alarme und Metriken erstellen. Sie können CloudWatch Logs verwenden, um Ihre Protokolldatensätze in einem Speicher mit hoher Beständigkeit abzulegen. Weitere Informationen finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.

Senden der Protokollausgabe von Aurora MySQL an Tabellen

Sie können die allgemeinen und Slow-Query-Protokolle an Tabellen in der DB-Instance weiterleiten, indem Sie eine DB-Parametergruppe erstellen und den `log_output`-Serverparameter auf `TABLE` setzen. Allgemeine Abfragen werden anschließend in der `mysql.general_log`-Tabelle und Slow-Queries in der`mysql.slow_log`-Tabelle protokolliert. Sie können die Tabellen abfragen, um auf Protokollinformationen zuzugreifen. Durch Aktivieren dieser Protokollierung wird die Datenmenge erhöht, die in die Datenbank geschrieben wird, was die Performance beeinträchtigen kann.

Das allgemeine Protokoll und das Slow-Query-Protokoll sind standardmäßig deaktiviert. Um die Protokollierung in Tabellen zu aktivieren, müssen Sie auch die Serverparameter `general_log` und `slow_query_log` auf `1` setzen.

Protokolltabellen wachsen stetig, bis die entsprechenden Protokollierungsaktivitäten ausgeschaltet werden, indem der entsprechende Parameter auf gesetzt wir `0`. Mit der Zeit sammelt sich häufig eine große Datenmenge an und belegt einen beträchtlichen Anteil Ihres zugeteilten Speicherplatzes. Amazon Aurora erlaubt Ihnen nicht, die Protokolltabellen zu kürzen, aber Sie können ihre Inhalte verschieben. Beim Rotieren einer Tabelle wird deren Inhalt in einer Sicherungstabelle gespeichert, und anschließend wird eine neue leere Protokolldatei angelegt. Sie können Protokolltabellen mithilfe der folgenden Befehlszeilenprozeduren manuell rotieren, wobei die Eingabeaufforderung mit bezeichnet is `PROMPT>`:

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

Um alte Daten komplett zu entfernen und den Speicherplatz zurückzugewinnen, rufen Sie die entsprechende Prozedur zweimal nacheinander auf.

Konfigurieren der Aurora MySQL-Binärprotokollierung für Single-AZ-Datenbanken

Das Binärprotokoll ist eine Reihe von Protokolldateien, die Informationen zu Datenänderungen enthalten, die an einer Aurora-MySQL-Server-Instance vorgenommen wurden. Das Binärprotokoll enthält Informationen wie die folgenden:

- Ereignisse, die Datenbankänderungen wie Tabellenerstellungen oder Zeilenänderungen beschreiben
- Informationen über die Dauer jeder Anweisung, durch die Daten aktualisiert wurden
- Ereignisse für Anweisungen, durch die Daten aktualisieren werden hätten können, aber nicht wurden

Das binäre Protokoll zeichnet Anweisungen auf, die während der Replikation gesendet werden. Es ist auch für einige Wiederherstellungsvorgänge erforderlich. Weitere Informationen finden Sie unter The Binary Log in der MySQL-Dokumentation.

Binäre Protokolle sind nur von der primären DB-Instance aus zugänglich, nicht von den Replicas.

MySQL in Amazon Aurora unterstützt die binären Protokollformate row-based, statement-based und mixed. Wir empfehlen gemischt, sofern Sie kein spezifisches Format des Binärprotokolls benötigen. Einzelheiten zu den verschiedenen Aurora-MySQL-Binärprotokollformaten finden Sie in der MySQL-Dokumentation unter Binärprotokollierungsformate.

Zur Verwendung der Replikation ist das binäre Protokollierungsformat wichtig, da es den Datensatz der Datenänderungen bestimmt, der in der Quelle aufgezeichnet und an die Replikationsziele gesendet wird. Weitere Informationen über Vor- und Nachteile verschiedener binärer Protokollierungsformate finden Sie unter Vorteile und Nachteile einer auf Anweisungen und einer auf Zeilen basierenden Replikation in der MySQL-Dokumentation.

> **Important**
>
> Mit MySQL 8.0.34 hat MySQL den Parameter `binlog_format` als veraltet eingestuft. In späteren MySQL-Versionen plant MySQL, den Parameter zu entfernen und nur die zeilenbasierte Replikation zu unterstützen. Daher empfehlen wir, für neue MySQL-Replikationseinrichtungen die zeilenbasierte Protokollierung zu verwenden. Weitere Informationen finden Sie unter binlog_format in der MySQL-Dokumentation.

Die MySQL-Versionen 8.0 und 8.4 akzeptieren den Parameter `binlog_format`. Bei Verwendung dieses Parameters gibt MySQL eine Warnmeldung zur Veralterung aus. In einer zukünftigen Hauptversion wird MySQL den Parameter `binlog_format` entfernen.
Die anweisungsbasierte Replikation kann zu Inkonsistenzen zwischen dem Quell-DB--Cluster und einem Lese-Replikat führen. Weitere Informationen finden Sie unter Determination of Safe and Unsafe Statements in Binary Logging in der MySQL-Dokumentation.
Durch die Aktivierung der binären Protokollierung wird die Anzahl der Write-Disk-I/O-Operationen für den DB-Cluster erhöht. Sie können die IOPS-Nutzung mit der CloudWatch-Metrik `VolumeWriteIOPs` überwachen.

Stellen Sie das MySQL-binäres-Protokollierungsformat wie folgt ein:

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie die zu ändernde DB-Cluster-Parametergruppe aus, die dem DB-Cluster zugeordnet ist.

   Eine Standard-Parametergruppe kann nicht modifiziert werden. Erstellen Sie eine neue Parametergruppe und ordnen Sie diese dem DB--Cluster zu, wenn der DB--Cluster eine Standardparametergruppe verwendet.

   Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.
4. Wählen Sie unter Aktionen die Option Bearbeiten aus.
5. Legen Sie den Parameter `binlog_format` auf das binäre Protokollierungsformat Ihrer Wahl fest (`ROW`, `STATEMENT` oder `MIXED`). Sie können auch den Wert `OFF` verwenden, um die Binärprotokollierung zu deaktivieren.

   > **Note**
   >
   > Die Einstellung von `binlog_format` auf `OFF` in der DB-Cluster-Parametergruppe deaktiviert die Sitzungsvariable `log_bin`. Dadurch wird die binäre Protokollierung auf dem DB-Cluster von Aurora MySQL deaktiviert, wodurch wiederum die Sitzungsvariable `binlog_format` auf den Standardwert `ROW` in der Datenbank zurückgesetzt wird.

6. Wählen Sie Save changes (Änderungen speichern), um die Aktualisierungen in dieser DB-Cluster-Parametergruppe zu speichern.

Nachdem Sie diese Schritte ausgeführt haben, müssen Sie die Writer-Instance im DB-Cluster neu starten, damit Ihre Änderungen übernommen werden. Wenn Sie in Aurora MySQL Version 2.09 und niedriger die Writer-Instance neu starten, werden auch alle Reader-Instances im DB-Cluster neu gestartet. In Aurora MySQL Version 2.10 und höher müssen Sie alle Reader-Instances manuell neu starten. Weitere Informationen finden Sie unter Neustart eines Amazon Aurora DB-Clusters oder einer Amazon Aurora DB-Instance.

> **Important**
>
> Das Ändern einer DB-Cluster-Parametergruppe wirkt sich auf alle DB-Cluster aus, die diese Parametergruppe verwenden. Wenn Sie unterschiedliche binäre Protokollierungsformate für verschiedene Aurora-MySQL-DB-Cluster in einer AWS-Region angeben möchten, müssen die DB-Cluster unterschiedliche DB-Cluster-Parametergruppen verwenden. Diese Parametergruppen identifizieren unterschiedliche Protokollierungsformate. Weisen Sie jedem DB-Cluster die entsprechende DB-Cluster-Parametergruppe zu. Weitere Informationen zu Aurora MySQL-Parametern finden Sie unter Aurora MySQL Konfigurationsparameter.

Zugriff auf MySQL-Binärprotokolle

Sie können das Dienstprogramm mysqlbinlog verwenden, um Binärprotokolle aus RDS-for-MySQL-DB-Instances herunterzuladen oder zu streamen. Das Binärprotokoll wird auf den lokalen Computer heruntergeladen, von wo aus Sie Aktionen, wie die Wiedergabe eines Protokolls mithilfe des Hilfsprogramms mysql ausführen können. Weitere Informationen über die Verwendung des Dienstprogramms mysqlbinlog finden Sie unter Verwenden von mysqlbinlog zum Sichern binärer Protokolldateien in der MySQL-Dokumentation.

Verwenden Sie zum Ausführen des Dienstprogramms mysqlbinlog mit einer Amazon-RDS-Instance die folgenden Optionen:

- `--read-from-remote-server` – Erforderlich.
- `--host` – der DNS-Name vom Endpunkt der Instance.
- `--port` – der von der Instance verwendete Port.
- `--user` – ein MySQL-Benutzer, dem die Berechtigung `REPLICATION SLAVE` erteilt wurde.

- `--password` – das Passwort für den MySQL-Benutzer oder lassen Sie einen Passwortwert aus, damit das Dienstprogramm zur Eingabe eines Passworts auffordert.
- `--raw` – Laden Sie die Datei im Binärformat herunter.
- `--result-file` – die lokale Datei, die den raw-Output empfangen soll.
- `--stop-never` – Streamen Sie die binären Protokolldateien.
- `--verbose` – Wenn Sie das Binlog-Format `ROW` verwenden, schließen Sie diese Option ein, um die Zeilenereignisse als Pseudo-SQL-Anweisungen anzuzeigen. Weitere Informationen zur Option `--verbose` finden Sie unter mysqlbinlog row event display in der MySQL-Dokumentation.
- Geben Sie die Namen einer oder mehrerer Binärprotokolldateien an. Verwenden Sie den SQL-Befehl `SHOW BINARY LOGS`, um eine Liste der verfügbaren Protokolle abzurufen.

Weitere Informationen über mysqlbinlog-Optionen finden Sie unter mysqlbinlog – Hilfsprogramm für die Verarbeitung binärer Protokolldateien in der MySQL-Dokumentation.

Die folgenden Beispiele veranschaulichen die Verwendung des Dienstprogramms mysqlbinlog.

Für Linux, macOS oder Unix:

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

Für Windows:

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
```

```
    --result-file=/tmp/ ^
    binlog.00098
```

Binärprotokolle müssen auf der DB-Instance verfügbar bleiben, damit das Dienstprogramm „mysqlbinlog“ auf sie zugreifen kann. Verwenden Sie die gespeicherte Prozedur mysql.rds_set_configuration und geben Sie einen Zeitraum mit ausreichend Zeit für den Download der Protokolle an, um die Verfügbarkeit sicherzustellen. Wenn diese Konfiguration nicht festgelegt ist, löscht Amazon RDS die Binärprotokolle so schnell wie möglich, was zu Lücken in den Binärprotokollen führt, die das Dienstprogramm „mysqlbinlog“ abruft.

Das folgende Beispiel setzt den Aufbewahrungszeitraum auf 1 Tag.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

Verwenden Sie die gespeicherte Prozedur mysql.rds_show_configuration, um die aktuelle Einstellung anzeigen zu lassen.

```
call mysql.rds_show_configuration;
```

Datenbank-Protokolldateien von Aurora PostgreSQL

Sie können die folgenden Arten von Aurora-PostgreSQL-Protokolldateien überwachen:

- PostgreSQL-Protokoll
- Instance-Protokoll
- Fehlerprotokoll zur IAM-Datenbankauthentifizierung

> **Note**
>
> Um Fehlerprotokolle für die IAM-Datenbankauthentifizierung zu aktivieren, müssen Sie zunächst die IAM-Datenbankauthentifizierung für Ihre DB-Cluster von Aurora PostgreSQL aktivieren. Weitere Informationen zur Aktivierung der IAM-Datenbankauthentifizierung finden Sie unter Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung.

Aurora PostgreSQL protokolliert Datenbankaktivitäten in der PostgreSQL-Standardprotokolldatei. Bei einer On-Premises PostgreSQL-DB-Instance werden diese Nachrichten in `log/postgresql.log` lokal gespeichert. Für einen DB-Cluster von Aurora PostgreSQL ist die Protokolldatei auf dem Aurora-Cluster verfügbar. Auf diese Protokolle kann auch über die AWS-Managementkonsole zugegriffen werden, wo Sie sie einsehen oder herunterladen können. Die Standardprotokollierungsebene erfasst Anmeldefehler, schwerwiegende Serverfehler, Deadlocks und Abfragefehler.

Weitere Informationen zum Anzeigen, Herunterladen und Überwachen von dateibasierten Datenbankprotokollen finden Sie unter Überwachen von Amazon Aurora-Protokolldateien. Weitere Informationen zu PostgreSQL-Protokollen finden Sie unter Arbeiten mit Amazon RDS und Aurora-PostgreSQL-Protokollen: Teil 1 und Arbeiten mit Amazon RDS und Aurora-PostgreSQL-Protokollen: Teil 2.

Zusätzlich zu den in diesem Thema behandelten PostgreSQL-Standardprotokollen unterstützt Aurora PostgreSQL auch die PostgreSQL-Audit-Erweiterung (`pgAudit`). Die meisten regulierten Branchen und Regierungsbehörden müssen ein Auditprotokoll oder einen Audit-Trail für die Änderungen von Daten führen, um die gesetzlichen Bestimmungen zu erfüllen. Weitere Informationen zur Installation und Verwendung von pgAudit finden Sie unter Verwenden von pgAudit zur Protokollierung der Datenbankaktivität.

Aurora erstellt eine separate Protokolldatei für DB-Instances, bei denen die Auto-Pause-Funktion aktiviert ist. In der Datei instance.log werden alle Gründe aufgezeichnet, warum diese DB-Instances nicht wie erwartet pausiert werden konnten. Weitere Informationen zum Verhalten von Instance-Protokolldateien und zur Auto-Pause-Funktion von Aurora finden Sie unter Monitoring Aurora Serverless v2 pause and resume activity.

Themen

- Parameter für die Protokollierung in Aurora PostgreSQL
- Aktivieren der Abfrageprotokollierung für Ihren DB-Cluster von Aurora PostgreSQL

Parameter für die Protokollierung in Aurora PostgreSQL

Sie können das Protokollierungsverhalten für Ihren DB-Cluster von Aurora PostgreSQL anpassen, indem Sie verschiedene Parameter ändern. In der folgenden Tabelle finden Sie unter anderem die Parameter, die sich darauf auswirken, wie lange die Protokolle gespeichert werden, wann das Protokoll rotiert werden soll und ob das Protokoll im CSV-Format (Comma-Separated Value) ausgegeben werden soll. Außerdem ist abgesehen von anderen Einstellungen die Textausgabe angegeben, die an STDERR gesendet wurde. Wenn Sie die Einstellungen für die modifizierbaren Parameter ändern möchten, verwenden Sie eine benutzerdefinierte DB-Cluster -Parametergruppe für Ihren DB-Cluster von Aurora PostgreSQL. Weitere Informationen finden Sie unter Parametergruppen für Amazon Aurora.

Parameter	Standard	Beschreibung
log_destination	stderr	Legt das Ausgabeformat für das Protokoll fest. Die Standardeinstellung ist `stderr`, aber Sie können auch das CSV-Format angeben, indem Sie der Einstellung `csvlog` hinzufügen. Weitere Informationen finden Sie unter Festlegen des Protokollziels (`stderr`, `csvlog`).
log_filename	postgresql.log.%Y-%m-%d-%H%M	Gibt das Muster für den Namen der Protokolldatei an. Zusätzlich zur Standardeinstellung unterstützt dieser Parameter `postgresql.log.%Y-%m-%d` und `postgresql.log.%Y-%m-%d-%H` für das Dateiname

Parameter	Standard	Beschreibung
		nmuster. Für Aurora PostgreSQL-Version 17.4 und höher können Sie diesen Parameter nicht ändern.
log_line_prefix	%t:%r:%u@%d:[%p]:	Definiert das Präfix für jede Protokollzeile, die in `stderr` geschrieben wird, um die Uhrzeit (%t), den Remote-Host (%r), den Benutzer (%u), die Datenbank (%d) und die Prozess-ID (%p) anzugeben.
log_rotation_age	60	Minuten, nach denen die Protokolldatei automatisch rotiert wird. Sie können diesen Wert im Bereich von 1 Minute bis 1440 Minuten ändern. Weitere Informationen finden Sie unter Festlegen der Rotation der Protokolldatei.
log_rotation_size	–	Die Größe (kB), bei der das Protokoll automatisch rotiert wird. Sie können diesen Wert im Bereich von 50 000 bis 1 000 000 Kilobyte ändern. Weitere Informationen hierzu finden Sie unter Festlegen der Rotation der Protokolldatei.
rds.log_retention_period	4320	PostgreSQL-Protokolle, die älter als die angegebene Anzahl von Minuten sind, werden gelöscht. Mit dem Standardwert von 4.320 Minuten werden Protokolldateien nach 3 Tagen gelöscht. Weitere Informationen finden Sie unter Festlegen des Aufbewahrungszeitraums für Protokolle.

Anwendungsprobleme können Sie identifizieren, indem Sie im Protokoll nach Abfragefehlern, Anmeldefehlern, Deadlocks und schwerwiegenden Serverfehlern suchen. Angenommen, Sie haben eine Legacy-Anwendung von Oracle in Aurora PostgreSQL konvertiert, wobei jedoch nicht alle Abfragen ordnungsgemäß umgewandelt wurden. Diese falsch formatierten Abfragen generieren

Fehlermeldungen in den Protokollen, mit denen Sie Probleme identifizieren können. Weitere Informationen zur Protokollierung von Abfragen finden Sie unter Aktivieren der Abfrageprotokollierung für Ihren DB-Cluster von Aurora PostgreSQL.

In den folgenden Themen finden Sie Informationen darüber, wie Sie verschiedene Parameter festlegen, die die grundlegenden Details Ihrer PostgreSQL-Protokolle steuern.

Themen

- Festlegen des Aufbewahrungszeitraums für Protokolle
- Festlegen der Rotation der Protokolldatei
- Festlegen des Protokollziels (stderr, csvlog)
- Grundlagen des Parameters log_line_prefix

Festlegen des Aufbewahrungszeitraums für Protokolle

Der `rds.log_retention_period`-Parameter gibt an, wie lange Ihr DB-Cluster von Aurora PostgreSQL die entsprechenden Protokolldateien aufbewahrt. Die Standardeinstellung ist 3 Tage (4 320 Minuten). Sie können diese Einstellung jedoch auf einen beliebigen Wert zwischen 1 Tag (1 440 Minuten) und 7 Tagen (10 080 Minuten) festlegen. Stellen Sie sicher, dass Ihr DB-Cluster von Aurora PostgreSQL über ausreichend Speicherplatz verfügt, um die Protokolldateien für diesen Zeitraum zu speichern.

Wir empfehlen Ihnen, Ihre Protokolle routinemäßig in Amazon CloudWatch Logs veröffentlichen zu lassen, damit Sie Systemdaten lange nach dem Entfernen der Protokolle aus Ihrem DB-Cluster von Aurora PostgreSQL anzeigen und analysieren können. Weitere Informationen finden Sie unter Veröffentlichen von Aurora-PostgreSQL-Protokollen in Amazon CloudWatch Logs. Nachdem Sie die CloudWatch-Veröffentlichung eingerichtet haben, löscht Aurora ein Protokoll erst, wenn es in CloudWatch Logs veröffentlicht wurde.

Amazon Aurora komprimiert ältere PostgreSQL Protokolle, wenn der Speicher für die DB-Instance einen Schwellenwert erreicht. Aurora komprimiert die Dateien mit dem gzip-Komprimierungsprogramm. Weitere Informationen finden Sie auf der gzip-Website.

Wenn der Speicher für die DB-Instance niedrig ist und alle verfügbaren Protokolle komprimiert sind, erhalten Sie eine Warnung wie die folgende:

```
Warning: local storage for PostgreSQL log files is critically low for
```

```
this Aurora PostgreSQL instance, and could lead to a database outage.
```

Wenn nicht genügend Speicher vorhanden ist, löscht Aurora möglicherweise komprimierte PostgreSQL-Protokolle vor Ablauf des angegebenen Aufbewahrungszeitraums. In diesem Fall wird eine Meldung ähnlich der folgenden angezeigt:

```
The oldest PostgreSQL log files were deleted due to local storage constraints.
```

Festlegen der Rotation der Protokolldatei

Neue Protokolldateien werden von Aurora standardmäßig jede Stunde erstellt. Das Timing wird vom Parameter `log_rotation_age` kontrolliert. Dieser Parameter hat einen Standardwert von 60 (Minuten). Sie können ihn jedoch auf jeden beliebigen Wert zwischen 1 Minute und 24 Stunden (1 440 Minuten) festlegen. Wenn die Rotation ansteht, wird eine neue eindeutige Protokolldatei erstellt. Die Datei wird nach dem Muster benannt, das durch den Parameter `log_filename` angegeben wird.

Protokolldateien können auch entsprechend ihrer Größe gedreht werden, wie im Parameter `log_rotation_size` angegeben. Dieser Parameter gibt an, dass das Protokoll rotiert werden soll, wenn es die angegebene Größe (in Kilobyte) erreicht. Der Standardwert für `log_rotation_size` ist 100 000 KB (Kilobyte) für einen DB-Cluster von Aurora PostgreSQL. Sie können diesen Parameter jedoch auf einen beliebigen Wert zwischen 50 000 und 1 000 000 Kilobyte festlegen.

Die Protokolldateinamen basieren auf dem Dateinamenmuster des Parameters `log_filename`. Die verfügbaren Einstellungen für diesen Parameter lauten wie folgt:

- `postgresql.log.%Y-%m-%d` – Standardformat für den Namen der Protokolldatei. Nimmt das Jahr, den Monat und das Datum in den Namen der Protokolldatei auf.
- `postgresql.log.%Y-%m-%d-%H` – Nimmt die Stunde in das Format des Protokolldateinamens auf.
- `postgresql.log.%Y-%m-%d-%H%M` – Nimmt die Stunde und die Minuten in das Format des Protokolldateinamens auf.

Wenn Sie den `log_rotation_age`-Parameter auf weniger als 60 Minuten festlegen, stellen Sie für den `log_filename`-Parameter das Minutenformat ein.

Weitere Informationen finden Sie unter `log_rotation_age` und `log_rotation_size` in der PostgreSQL-Dokumentation.

Festlegen des Protokollziels (**`stderr`**, **`csvlog`**)

Standardmäßig generiert Aurora PostgreSQL Protokolle im Standardfehlerformat (stderr). Dieses Format ist die Standardeinstellung des Parameters `log_destination`. Jede Nachricht erhält ein Präfix nach dem im `log_line_prefix`-Parameter angegebenen Muster. Weitere Informationen finden Sie unter Grundlagen des Parameters log_line_prefix.

Aurora PostgreSQL kann die Protokolle auch im `csvlog`-Format generieren. Das `csvlog`-Format ist nützlich, um die Protokolldaten als CSV-Daten zu analysieren. Angenommen, Sie verwenden die Erweiterung `log_fdw`, um mit Ihren Protokollen als Fremdtabellen zu arbeiten. Die Fremdtabelle, die für `stderr`-Protokolldateien erstellt wurde, enthält eine einzelne Spalte mit Protokollereignisdaten. Durch Hinzufügen von `csvlog` zum `log_destination`-Parameter erhalten Sie die Protokolldatei im CSV-Format mit Abgrenzungen für die verschiedenen Spalten der Fremdtabelle. Sie können Ihre Protokolle jetzt einfacher sortieren und analysieren.

Wenn Sie `csvlog` für diesen Parameter angeben, beachten Sie, dass sowohl `stderr`- als auch `csvlog`-Dateien generiert werden. Achten Sie auf den von den Protokollen verbrauchten Speicher und berücksichtigen Sie dabei die `rds.log_retention_period`und andere Einstellungen, die sich auf den Protokollspeicher und Turnover auswirken. Wenn Sie `stderr` und `csvlog` verwenden, verdoppelt sich der von den Protokollen verbrauchte Speicher.

Wenn Sie `csvlog` zu `log_destination` hinzufügen und zu `stderr` allein zurückkehren möchten, müssen Sie den Parameter zurücksetzen. Rufen Sie dazu die Amazon-RDS-Konsole auf und öffnen Sie die benutzerdefinierte DB-Cluster-Parametergruppe für Ihre Instance. Wählen Sie den `log_destination`-Parameter, die Option Edit parameter (Parameter bearbeiten) und dann Reset (Zurücksetzen) aus.

Weitere Informationen zum Konfigurieren der Protokollierung finden Sie unter Arbeiten mit Amazon-RDS- und Aurora-PostgreSQL-Protokollen: Teil 1.

Grundlagen des Parameters log_line_prefix

Das Protokollformat `stderr` wird jeder Protokollnachricht mit den Details vorangestellt, die durch den Parameter `log_line_prefix` angegeben werden. Der Standardwert ist:

```
%t:%r:%u@%d:[%p]:t
```

Ab Aurora PostgreSQL-Version 16 können Sie auch Folgendes auswählen:

```
%m:%r:%u@%d:[%p]:%l:%e:%s:%v:%x:%c:%q%a
```

Jeder Protokolleintrag, der an stderr gesendet wird, enthält die folgenden Informationen, die auf dem ausgewählten Wert basieren:

- `%t` – Zeitpunkt des Protokolleintrags ohne Millisekunden
- `%m` – Zeitpunkt des Protokolleintrags mit Millisekunden
- `%r` – Adresse des Remote-Hosts
- `%u@%d` – Benutzername und Datenbankname
- `[%p]` – Prozess-ID, falls verfügbar
- `%l` – Nummer der Protokollzeile pro Sitzung
- `%e` – SQL-Fehlercode
- `%s` – Start-Zeitstempel des Prozesses
- `%v` – ID der virtuellen Transaktion
- `%x` – Transaktions-ID
- `%c` – Sitzungs-ID
- `%q` – Nicht für Sitzungen vorgesehene Begrenzung
- `%a` – Anwendungsname

Aktivieren der Abfrageprotokollierung für Ihren DB-Cluster von Aurora PostgreSQL

Sie können detailliertere Informationen über Ihre Datenbankaktivitäten sammeln, einschließlich Abfragen, Abfragen, die auf Sperren warten, Prüfpunkte und viele andere Details, indem Sie einige der in der folgenden Tabelle aufgeführten Parameter festlegen. Dieses Thema konzentriert sich auf das Protokollieren von Abfragen.

Parameter	Standard	Beschreibung
log_connections	–	Protokolliert jede erfolgreiche Verbindung. Informationen zur Verwendung dieses Parameters mit `log_disconnections` zum Erkennen von Verbindungsproblemen finden Sie unter Verwalten von Aurora PostgreSQL Verbindungsabwanderung mit Pooling .

Parameter	Standard	Beschreibung
log_disconnections	–	Protokolliert das Ende jeder Sitzung und ihre Dauer. Informationen zur Verwendung dieses Parameters mit `log_connections` zum Erkennen von Verbindungsproblemen finden Sie unter Verwalten von Aurora PostgreSQL Verbindungsabwanderung mit Pooling .
log_checkpoints	–	Gilt nicht für Aurora PostgreSQL
log_lock_waits	–	Protokolliert lange Sperrenwartezeiten. Dieser Parameter ist standardmäßig nicht festgelegt.
log_min_duration_sample	–	(ms) Legt die Mindestausführungszeit fest, jenseits der stichprobenartig Anweisungen protokolliert werden. Die Stichprobengröße wird mit dem `log_statement_sample_rate` -Parameter festgelegt.
log_min_duration_statement	–	Jede SQL-Anweisung, die mindestens für die angegebene Zeit oder länger ausgeführt wird, wird protokolliert. Dieser Parameter ist standardmäßig nicht festgelegt. Die Aktivierung dieses Parameters kann Sie dabei unterstützen, nicht optimierte Abfragen zu finden.

Parameter	Standard	Beschreibung
log_statement	–	Legt den Typ der protokollierten Anweisungen fest. Standardmäßig ist dieser Parameter nicht festgelegt, aber Sie können ihn in `all`, `ddl` oder `mod` ändern, um die Typen von SQL-Anweisungen anzugeben, die protokolliert werden sollen. Wenn Sie etwas anderes als `none` für diesen Parameter angeben, sollten Sie auch zusätzliche Maßnahmen ergreifen, um die Offenlegung von Passwörtern in den Protokolldateien zu verhindern. Weitere Informationen finden Sie unter Reduzieren des Risikos der Offenlegung von Passwörtern bei der Verwendung der Abfrageprotokollierung.
log_statement_sample_rate	–	Der Prozentsatz der Anweisungen, die die in `log_min_duration_sample` angegebene Zeit bei der Protokollierung überschreiten. Diese Angabe wird als Gleitkommawert zwischen 0,0 und 1,0 ausgedrückt.
log_statement_stats	–	Schreibt kumulative Leistungsstatistiken in das Serverprotokoll.

Verwendung der Protokollierung, um Abfragen mit langsamer Ausführung zu suchen

Sie können SQL-Anweisungen und Abfragen protokollieren, um Abfragen zu finden, die langsam ausgeführt werden. Sie aktivieren diese Funktion, indem Sie die Einstellungen der Parameter `log_statement` und `log_min_duration` wie in diesem Abschnitt beschrieben ändern. Bevor Sie die Abfrageprotokollierung für Ihren DB-Cluster von Aurora PostgreSQL aktivieren, sollten Sie sich der möglichen Offenlegung von Passwörtern in den Protokollen bewusst sein und wissen, wie Sie die Risiken minimieren können. Weitere Informationen finden Sie unter Reduzieren des Risikos der Offenlegung von Passwörtern bei der Verwendung der Abfrageprotokollierung.

Nachstehend finden Sie Referenzinformationen zu den Parametern `log_statement` und `log_min_duration`.

log_statement

Dieser Parameter gibt den Typ der SQL-Anweisungen an, die an das Protokoll gesendet werden sollen. Der Standardwert ist `none`. Wenn Sie diesen Parameter in `all`, `ddl` oder `mod` ändern, stellen Sie sicher, dass Sie die empfohlenen Maßnahmen ergreifen, um das Risiko einer Offenlegung von Passwörtern in den Protokollen zu reduzieren. Weitere Informationen finden Sie unter Reduzieren des Risikos der Offenlegung von Passwörtern bei der Verwendung der Abfrageprotokollierung.

all

Protokolliert alle Anweisungen. Diese Einstellung wird für Debugging-Zwecke empfohlen.

ddl

Protokolliert alle Data Definition Language (DDL)-Anweisungen wie CREATE, ALTER, DROP usw.

mod

Protokolliert alle DDL-Anweisungen und Data Manipulation Language (DML)-Anweisungen wie INSERT, UPDATE und DELETE, die die Daten modifizieren.

Keine

Es werden keine SQL-Anweisungen protokolliert. Wir empfehlen diese Einstellung, um das Risiko zu vermeiden, dass Passwörter in den Protokollen offengelegt werden.

log_min_duration_statement

Jede SQL-Anweisung, die mindestens für die angegebene Zeit oder länger ausgeführt wird, wird protokolliert. Dieser Parameter ist standardmäßig nicht festgelegt. Die Aktivierung dieses Parameters kann Sie dabei unterstützen, nicht optimierte Abfragen zu finden.

–1–2147483647

Die Laufzeit in Millisekunden (ms), in der eine Anweisung protokolliert wird.

So richten Sie die Abfrageprotokollierung ein

Bei diesen Schritten wird davon ausgegangen, dass Ihr DB-Custer von Aurora PostgreSQL eine benutzerdefinierte DB-Cluster-Parametergruppe verwendet.

1. Stellen Sie den Parameter `log_statement` auf `all` ein. Im folgenden Beispiel werden die Informationen gezeigt, die bei dieser Parametereinstellung in die Datei `postgresql.log` geschrieben werden.

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
---------------------- END OF LOG ----------------------
```

2. Legen Sie den Parameter `log_min_duration_statement` fest. Im folgenden Beispiel werden die Informationen gezeigt, die in die Datei `postgresql.log` geschrieben werden, wenn der Parameter auf 1 festgelegt wird.

   Abfragen, die die im `log_min_duration_statement`-Parameter angegebene Dauer überschreiten, werden protokolliert. Es folgt ein Beispiel. Sie können die Protokolldatei für Ihren DB-Cluster von Aurora PostgreSQL in der Amazon-RDS-Konsole anzeigen.

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
 table comments;
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
 167.754 ms
2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
```

```
2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
 (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
 total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
 estimate=131028 kB
---------------------- END OF LOG ----------------------
```

Reduzieren des Risikos der Offenlegung von Passwörtern bei der Verwendung der Abfrageprotokollierung

Wir empfehlen, für `log_statement` die Einstellung `none` beizubehalten, um zu vermeiden, dass Passwörter offengelegt werden. Wenn Sie `log_statement` auf `all`, `ddl` oder `mod` festlegen, sollten Sie einen oder mehrere der folgenden Schritte auszuführen.

- Verschlüsseln Sie vertrauliche Informationen für den Client. Weitere Informationen finden Sie unter Verschlüsselungsoptionen der PostgreSQL-Dokumentation. Verwenden Sie die Optionen `ENCRYPTED` und `UNENCRYPTED` der `CREATE`- und `ALTER`-Anweisungen. Weitere Informationen finden Sie im Abschnitt CREATE USER der PostgreSQL-Dokumentation.
- Richten Sie für Ihren DB-Cluster von Aurora PostgreSQL die PostgreSQL Auditing (PGAudit)-Erweiterung ein und verwenden Sie sie. Diese Erweiterung redigiert sensible Informationen in CREATE- und ALTER-Anweisungen, die an das Protokoll gesendet werden. Weitere Informationen finden Sie unter Verwenden von pgAudit zur Protokollierung der Datenbankaktivität.
- Beschränken Sie den Zugriff auf die CloudWatch-Protokolle.
- Verwenden Sie stärkere Authentifizierungsmechanismen wie IAM.

Überwachung von Amazon Aurora-API-Aufrufen in AWS CloudTrail

AWS CloudTrail ist ein AWS-Service, mit dem Sie Ihr AWS-Konto überprüfen können. AWS CloudTrail wird für Ihr AWS-Konto aktiviert, wenn Sie es erstellen. Weitere Informationen über CloudTrail finden Sie im AWS CloudTrail-Leitfaden.

Themen

- Integration von CloudTrail in Amazon Aurora
- Amazon Aurora-Protokolldateieinträge

Integration von CloudTrail in Amazon Aurora

Alle Amazon Aurora-Aktionen werden von CloudTrail protokolliert. CloudTrail bietet eine Aufzeichnung der von einem Benutzer, einer Rolle oder einem AWS-Service in Amazon Aurora durchgeführten Aktionen.

CloudTrail-Ereignisse

CloudTrail erfasst API-Aufrufe für Amazon Aurora als Ereignisse. Ein Ereignis stellt eine einzelne Anfrage aus einer beliebigen Quelle dar und enthält unter anderem Informationen über die angeforderte Aktion, das Datum und die Uhrzeit der Aktion sowie über die Anfrageparameter. Zu Ereignissen gehören Aufrufe von der Amazon RDS-Konsole und von Code-Aufrufen der Amazon-RDS-API-Operationen.

Amazon Aurora-Aktivitäten werden in einem CloudTrail-Ereignis im Event history (Ereignisverlauf) aufgezeichnet. Sie können die CloudTrail-Konsole verwenden, um API-Aktivitäten und -Ereignisse der letzten 90 Tage in einer AWS-Region anzuzeigen. Weitere Informationen finden Sie unter Anzeigen von Ereignissen mit dem CloudTrail-API-Ereignisverlauf.

CloudTrail-Trails

Zur kontinuierlichen Aufzeichnung von Ereignissen in Ihrem AWS-Konto, einschließlich Ereignissen für Amazon Aurora, erstellen Sie einen Trail. Ein Trail ist eine Konfiguration, die die Zustellung von Ereignissen an einen angegebenen Amazon-S3-Bucket ermöglicht. CloudTrail übermittelt Protokolldateien in der Regel innerhalb von 15 Minuten nach einer Kontoaktivität.

> **Note**
>
> Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse in der CloudTrail-Konsole trotzdem in Ereignisverlauf anzeigen.

Sie können zwei Arten von Trails für ein AWS-Konto erstellen: einen Trail, der für alle Regionen gilt, oder einen Trail für eine Region. Wenn Sie ein Trail in der Konsole anlegen, gilt dieser für alle Regionen.

Darüber hinaus können Sie andere AWS-Services konfigurieren, um die in den CloudTrail-Protokollen erfassten Ereignisdaten weiter zu analysieren und entsprechend zu agieren. Weitere Informationen finden Sie unter:

- Übersicht zum Erstellen eines Trails
- In CloudTrail unterstützte Services und Integrationen
- Konfigurieren von Amazon SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail-Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail-Protokolldateien aus mehreren Konten

Amazon Aurora-Protokolldateieinträge

CloudTrail-Protokolldateien können einen oder mehrere Einträge enthalten. CloudTrail-Protokolleinträge sind kein geordnetes Stacktrace der öffentlichen API-Aufrufe und erscheinen daher nicht in einer bestimmten Reihenfolge.

Das folgende Beispiel zeigt einen CloudTrail-Protokolleintrag, der die Aktion `CreateDBInstance` demonstriert.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
```

```
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
```

```
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
```

```
        "cACertificateIdentifier": "rds-ca-2015",
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


Wie im vorangegangenen Beispiel im Element `userIdentity` gezeigt, enthält jeder Ereignis- oder Protokolleintrag Informationen darüber, wer die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Gibt an, ob die Anforderung mit Root- oder IAM-Benutzer-Anmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.

- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde

Weitere Informationen zu `userIdentity` finden Sie unter CloudTrail-userIdentity-Element. Weitere Informationen zu `CreateDBInstance` und anderen Amazon Aurora-Aktionen finden Sie in der Amazon RDS API Reference (Amazon-RDS-API-Referenz).

Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams

Mithilfe von Datenbankaktivitätsstreams können Sie nahezu in Echtzeit Datenbankaktivitätsstreams überwachen.

Themen

- Übersicht über Datenbankaktivitätsstreams
- Netzwerkvoraussetzungen für Datenbankaktivitäts-Streams bei Aurora MySQL
- Starten eines Datenbankaktivitäts-Streams
- Abrufen des Status eines Datenbank-Aktivitätsstreams
- Stoppen eines Datenbankaktivitäts-Streams
- Überwachen von Datenbankaktivitäts-Streams
- IAM-Richtlinien-Beispiele für Datenbankaktivitäts-Streams

Übersicht über Datenbankaktivitätsstreams

Als Amazon-AuroraDatenbankadministrator müssen Sie Ihre Datenbank schützen und Compliance- und regulatorische Anforderungen erfüllen. Eine Strategie besteht darin, Datenbankaktivitätsstreams in Ihre Überwachungstools zu integrieren. Auf diese Weise überwachen und legen Sie Alarme für Prüfungsaktivitäten in Ihrem Amazon-Aurora-Cluster fest.

Sicherheitsbedrohungen sind sowohl extern als auch intern. Zum Schutz vor internen Bedrohungen können Sie den Administratorzugriff auf Datenströme durch die Konfiguration der Funktion „Datenbankaktivitätsstreams“ steuern. DBAs haben keinen Zugriff auf die Erfassung, Übertragung, Speicherung und Verarbeitung der Streams.

Inhalt

- Funktionsweise von Datenbankaktivitätsstreams
- Asynchroner und synchroner Modus für Datenbankaktivitätsstreams
- Anforderungen und Einschränkungen für Datenbankaktivitätsstreams
- Verfügbarkeit von Regionen und Versionen
- Unterstützte DB-Instance-Klassen für Datenbankaktivitätsstreams

Funktionsweise von Datenbankaktivitätsstreams

In Amazon Aurora starten Sie einen Datenbankaktivitäts-Stream auf Cluster-Ebene. Für alle DB-Instances in Ihrem Cluster sind Datenbankaktivitätsstreams aktiviert.

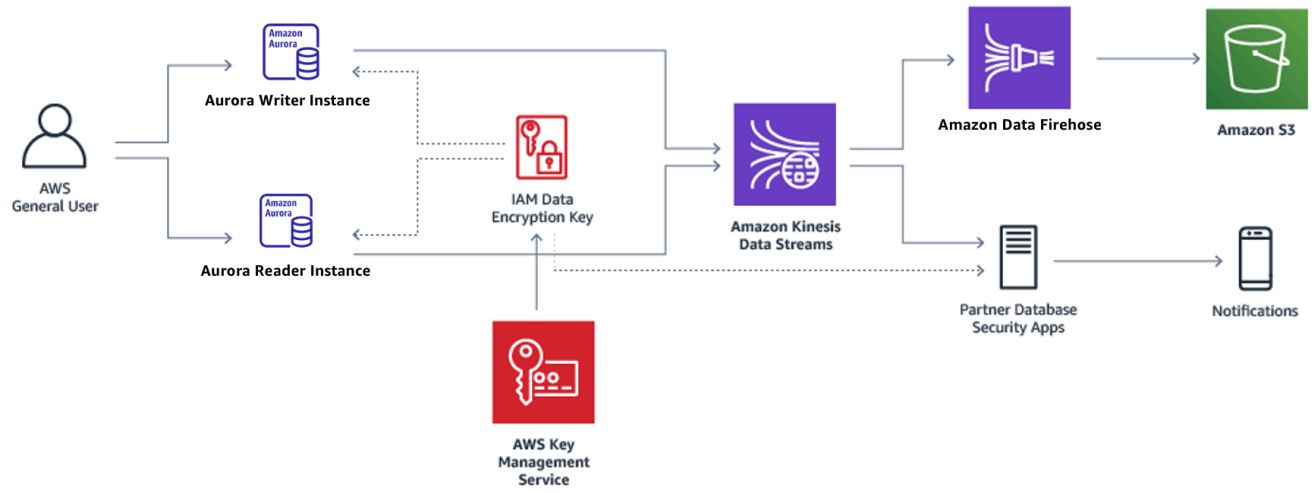
Ihr Aurora-DB-Cluster sendet Aktivitäten nahezu in Echtzeit per Push in einen Amazon Kinesis Data Stream. Der Kinesis Stream wird automatisch erstellt. In Kinesis können Sie AWS-Services wie Amazon Data Firehose und AWS Lambda konfigurieren, um den Stream zu verbrauchen und die Daten zu speichern.

> **Important**
>
> Die Verwendung der Datenbankaktivitätsstreams-Funktion in Amazon Aurora ist kostenlos, Amazon Kinesis erhebt jedoch Gebühren für einen Datenstrom. Weitere Informationen finden Sie unter Amazon Kinesis Data Streams – Preise.

Wenn Sie eine globale Aurora-Datenbank verwenden, starten Sie einen Datenbankaktivitäts-Stream separat auf jedem DB-Cluster. Jeder Cluster liefert Auditdaten innerhalb seiner eigenen AWS-Region an seinen eigenen Kinesis-Stream. Die Aktivitätsstreams funktionieren während eines Failovers nicht anders. Sie prüfen Ihre globale Datenbank weiterhin wie gewohnt.

Sie können Anwendungen für das Compliance-Management für den Verbrauch von Datenbankaktivitäts-Streams konfigurieren. Für Aurora PostgreSQL umfassen die Compliance-Anwendungen IBM Security Guardium und Impervas SecureSphere Database Audit and Protection. Solche Anwendungen können den Datenstrom verwenden, um Warnungen zu generieren und Ihre Aurora-DB-Cluster- zu prüfen.

In der folgenden Abbildung wird ein Aurora-DB-Cluster dargestellt, der mit Amazon Data Firehose konfiguriert ist.



Asynchroner und synchroner Modus für Datenbankaktivitätsstreams

Sie können festlegen, ob Datenbankaktivitätsereignisse in der Datenbanksitzung in einem der folgenden Modi behandelt werden sollen:

- Asynchroner Modus – Wenn eine Datenbanksitzung ein Aktivitätsstream-Ereignis generiert, kehrt die Sitzung sofort zu normalen Aktivitäten zurück. Im Hintergrund wird das Aktivitäts-Stream-Ereignis zu einem dauerhaften Datensatz gemacht. Wenn bei der Aufgabe im Hintergrund ein Fehler auftritt, wird ein RDS-Ereignis gesendet. Dieses Ereignis gibt Anfang und Ende der Zeitfenster an, in denen möglicherweise Datensätze des Aktivitäts-Stream-Ereignisses verloren gegangen sind.

  Der asynchrone Modus beschleunigt die Datenbankleistung, verschlechtert jedoch die Genauigkeit des Aktivitäts-Streams.

  > **Note**
  >
  > Der asynchrone Modus ist sowohl für Aurora PostgreSQL als auch Aurora MySQL verfügbar.

- Synchroner Modus – Wenn eine Datenbanksitzung im synchronen Modus ein Aktivitäts-Stream-Ereignis generiert, blockiert die Sitzung so lange andere Aktivitäten, bis das Ereignis dauerhaft gemacht wird. Falls es aus irgendeinem Grund nicht möglich ist, das Ereignis dauerhaft zu machen, kehrt die Datenbanksitzung zu normalen Aktivitäten zurück. Es wird allerdings ein RDS-

Ereignis gesendet, das darauf hinweist, dass Aktivitäts-Stream-Datensätze möglicherweise für einige Zeit verloren gehen. Wenn sich das System wieder in fehlerfreiem Zustand befindet, wird ein zweites RDS-Ereignis gesendet.

Der synchrone Modus fördert die Genauigkeit des Aktivitäts-Streams, verschlechtert jedoch die Datenbankleistung.

> **Note**
>
> Der synchrone Modus ist für Aurora PostgreSQL verfügbar. Sie können den synchronen Modus nicht mit Aurora MySQL verwenden.

Anforderungen und Einschränkungen für Datenbankaktivitätsstreams

In Aurora gelten für Datenbankaktivitätsstreams die folgenden Anforderungen und Einschränkungen:

- Amazon Kinesis ist für Datenaktivitätsstreams erforderlich.
- AWS Key Management Service (AWS KMS) ist für Datenbankaktivitäts-Streams erforderlich, da sie immer verschlüsselt sind.
- Das Anwenden zusätzlicher Verschlüsselung auf Ihren Amazon Kinesis Data Stream ist nicht mit Datenbankaktivitäts-Streams kompatibel, die bereits mit Ihrem AWS KMS-Schlüssel verschlüsselt sind.
- Starten Sie Ihren DatenbankaktivitätsStream auf DB-Cluster-Ebene. Wenn Sie Ihrem Cluster eine DB-Instance hinzufügen, müssen Sie keinen Aktivitätsstream auf der Instance starten: Sie wird automatisch überwacht.
- In einer globalen Aurora-Datenbank müssen Sie einen Datenbankaktivitäts-Stream separat auf jedem DB-Cluster starten. Jeder Cluster liefert Auditdaten innerhalb seiner eigenen AWS-Region an seinen eigenen Kinesis-Stream.
- Stellen Sie in Aurora PostgreSQL sicher, dass der Datenbankaktivitäts-Stream vor einem Hauptversions-Upgrade gestoppt wird. Sie können den Datenbank-Aktivitätsstream starten, nachdem das Upgrade abgeschlossen ist.

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Aurora-Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur

Verfügbarkeit von Versionen und Regionen mit Aurora und Datenbank-Aktivitätsstreams finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Datenbankaktivitäts-Streams.

Unterstützte DB-Instance-Klassen für Datenbankaktivitätsstreams

Für Aurora MySQL können Sie Datenbankaktivitätsstreams mit den folgenden DB-Instance-Klassen verwenden:

- db.r8g.*large
- db.r7g.*large
- db.r7i.*large
- db.r6g.*large
- db.r6i.*large
- db.r5.*large
- db.x2g.*

Für Aurora PostgreSQL können Sie Datenbankaktivitätsstreams mit den folgenden DB-Instance-Klassen verwenden:

- db.r8g.*large
- db.r7i.*large
- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r6id.*large
- db.r5.*large
- db.r4.*large
- db.x2g.*

Netzwerkvoraussetzungen für Datenbankaktivitäts-Streams bei Aurora MySQL

Im folgenden Abschnitt erfahren Sie, wie Sie Ihre Virtual Private Cloud (VPC) für die Verwendung mit Datenbankaktivitäts-Streams konfigurieren.

> **Note**
>
> Die Netzwerkvoraussetzungen für Aurora MySQL gelten für die folgenden Engine-Versionen:
>
> - Aurora MySQL Version 2 bis 2.11.3
> - Aurora MySQL-Version 2.12.0
> - Aurora MySQL Version 3 bis 3.04.2

Themen

- Voraussetzungen fürAWS KMS-Endpunkte
- Voraussetzungen für die öffentliche Verfügbarkeit
- Voraussetzungen für die private Verfügbarkeit

Voraussetzungen fürAWS KMS-Endpunkte

Instances in einem Aurora MySQL Cluster, die Aktivitätsströme verwenden, müssen aufAWS KMS-Endpunkte. Stellen Sie sicher, dass diese Anforderung erfüllt ist, bevor Sie Datenbankaktivitäts-Streams für Ihren Aurora MySQL-Cluster aktivieren. Wenn der Aurora-Cluster öffentlich zugänglich ist, ist diese Anforderung automatisch erfüllt.

> **Important**
>
> Wenn der Aurora MySQL-DB-Cluster nicht auf den AWS KMS-Endpunkt zugreifen kann, funktioniert der Aktivitäts-Stream nicht mehr. In diesem Fall werden Sie von Aurora über dieses Problem mithilfe von RDS-Ereignissen benachrichtigt.

Voraussetzungen für die öffentliche Verfügbarkeit

Damit ein Aurora DB-Cluster öffentlich ist, muss es die folgenden Anforderungen erfüllen:

- Öffentlicher ZugriffistJaimAWS-ManagementkonsoleSeite „Cluster Details“.
- Der DB-Cluster befindet sich in einem öffentlichen Amazon VPC-Subnetz. Weitere Informationen über öffentlich zugängliche DB-Instances finden Sie unter Arbeiten mit einer DB-Cluster in einer VPC. Weitere Informationen zu öffentlichen Amazon-VPC-Subnetzen finden Sie unter Ihre VPC und Subnetze.

Voraussetzungen für die private Verfügbarkeit

Wenn sich Ihr Aurora-DB-Cluster in einem öffentlichen VPC-Subnetz befindet und nicht öffentlich zugänglich ist, ist er privat. Um Ihren Cluster privat zu halten und ihn mit Datenbankaktivitätsströmen zu verwenden, haben Sie folgende Möglichkeiten:

- Konfigurieren Sie Network Address Translation (NAT) in Ihrer VPC. Weitere Informationen finden Sie unter NAT-Gateways.
- Erstellen Sie einen AWS KMS-Endpunkt in Ihrer VPC. Diese Option wird empfohlen, weil sie einfacher zu konfigurieren ist.

So erstellen Sie einen AWS KMS-Endpunkt in Ihrer VPC

1. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie im Navigationsbereich Endpunkte aus.
3. Klicken Sie auf Endpunkt erstellen.

   DieErstellen eines Endpunktswird angezeigt.
4. Gehen Sie wie folgt vor:
   - Wählen Sie in der Kategorie Dienste die Option AWS Dienste.
   - Wählen Sie unter Service Name (Dienstname) com.amazonaws.***region***.kms aus, wobei *region* (Region) die AWS-Region ist, in der sich Ihr Cluster befindet.
   - Für VPC wählen Sie die VPC, in der sich Ihr Cluster befindet.
5. Klicken Sie auf Endpunkt erstellen.

Weitere Informationen über das Konfigurieren von VPC-Endpunkten finden Sie unter VPC-Endpunkte.

Starten eines Datenbankaktivitäts-Streams

Starten Sie einen Aktivitätsstream auf Cluster-Ebene, um die Datenbankaktivität für alle Instances Ihres Aurora-DB-Clusters zu überwachen,. Alle DB-Instances, die Sie dem Cluster hinzufügen, werden ebenfalls automatisch überwacht. Wenn Sie eine globale Aurora-Datenbank verwenden, starten Sie einen Datenbankaktivitäts-Stream separat auf jedem DB-Cluster. Jeder Cluster liefert Auditdaten innerhalb seiner eigenen AWS-Region an seinen eigenen Kinesis-Stream.

Wenn Sie einen Aktivitätsstream starten, generiert jedes Datenbankaktivitätsereignis, das Sie in der Audit-Richtlinie konfiguriert haben, ein Ereignis im Aktivitätsstream. Zugriffsereignisse werden von SQL-Befehlen wie `CONNECT` und `SELECT` generiert. Änderungsereignisse werden von SQL-Befehlen wie `CREATE` und `INSERT` generiert.

Console

Starten eines Datenbankaktivitäts-Streams

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den DB-Cluster, für den/die Sie einen Aktivitätsstream starten möchten.
4. Wählen Sie für Actions (Aktionen) die Option Start activity stream (Aktivitäts-Stream starten) aus.

   Das Fenster Start database activity stream: *name* erscheint, wobei *name* Ihr DB-Cluster ist.
5. Geben Sie die folgenden Einstellungen ein:

   - Für AWS KMS key wählen Sie einen Schlüssel aus der Liste der AWS KMS keys.

     > **Note**
     >
     > Wenn Ihr Aurora-MySQL-Cluster nicht auf KMS-Schlüssel zugreifen kann, folgen Sie den Anweisungen unter Netzwerkvoraussetzungen für Datenbankaktivitäts-Streams bei Aurora MySQL, um diesen Zugriff zunächst zu aktivieren.

     Aurora verwendet den KMS-Schlüssel zur Verschlüsselung des Schlüssels, der wiederum die Datenbankaktivitäten verschlüsselt. Wählen Sie einen anderen KMS-Schlüssel als den Standardschlüssel. Weitere Informationen zu Verschlüsselungsschlüsseln und AWS KMS finden Sie unter Was ist AWS Key Management Service? im AWS Key Management Service-Entwicklerhandbuch.
   - Wählen Sie für Database activity stream mode (Datenbankaktivitäts-Stream-Modus) die Option Asynchronous (Asynchron) oder Synchronous (Synchron) aus.

> **Note**
>
> Diese Auswahl gilt nur für Aurora PostgreSQL. Bei Aurora MySQL können Sie nur den asynchronen Modus verwenden.

- Wählen Sie Sofort aus.

  Wenn Sie Sofort auswählen, wird der DB-Cluster sofort neu gestartet. Wenn Sie Während des nächsten Wartungsfensters auswählen, wird der DB-Cluster nicht sofort neu gestartet. In diesem Fall wird der Datenbankaktivitäts-Stream erst im nächsten Wartungsfenster gestartet.

6. Wählen Sie Start database activity stream (Datenbank-Aktivitätsstream starten) aus.

   Der Status für den DB-Cluster zeigt an, dass der Aktivitätsstream gestartet wird.

> **Note**
>
> Wenn Sie den Fehler `You can't start a database activity stream in this configuration` erhalten, überprüfen Sie Unterstützte DB-Instance-Klassen für Datenbankaktivitätsstreams, um festzustellen, ob Ihr DB-Cluster eine unterstützte Instance-Klasse verwendet.

AWS CLI

Um Datenbankaktivitätsstreams für einen DB-Cluster zu starten, konfigurieren Sie den DB-Cluster mit dem AWS CLI-Befehl start-activity-stream.

- `--resource-arn` *`arn`* – Gibt den Amazon-Ressourcennamen (ARN) des DB-Clusters an.
- `--mode` *`sync-or-async`* – Gibt entweder den synchronen (`sync`) oder den asynchronen (`async`) Modus an. Bei Aurora PostgreSQL können Sie einen der beiden Werte wählen. Geben Sie bei Aurora MySQL `async` an.
- `--kms-key-id` *`key`* – Gibt die KMS-Schlüssel-ID für die Verschlüsselung von Nachrichten im Datenbankaktivitäts-Stream an. Der AWS KMS-Schlüsselbezeichner ist der Schlüssel-ARN, die Schlüssel-ID, der Alias-ARN oder der Alias-Name für den AWS KMS key.

Das folgende Beispiel startet einen Datenbankaktivitätsstream für einen DB-Cluster im asynchronen Modus.

Für Linux, macOS oder Unix:

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-cluster-arn \
    --apply-immediately
```

Für Windows:

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-cluster-arn ^
    --apply-immediately
```

Amazon RDS API

Um Datenbankaktivitäts-Streams für einen DB-Cluster zu starten, konfigurieren Sie den Cluster mit der Operation StartActivityStream.

Rufen Sie die Aktion mit den folgenden Parametern auf:

- `Region`
- `KmsKeyId`
- `ResourceArn`
- `Mode`

> **Note**
>
> Wenn Sie eine Fehlermeldung erhalten, die besagt, dass Sie mit der aktuellen Version Ihrer Aurora-PostgreSQL-Datenbank keinen Datenbankaktivitäts-Stream starten können, wenden Sie den neuesten Patch für Aurora PostgreSQL an, bevor Sie einen solchen Stream starten. Informationen zum Aktualisieren Ihrer Aurora-PostgreSQL-Datenbank finden Sie unter Upgrade von Amazon-Aurora-DB-Clustern.

Im Folgenden sind die Patch-Mindestversionen aufgeführt, die erforderlich sind, um Datenbankaktivitäts-Streams mit Aurora PostgreSQL zu starten.

- 3.4.15 (11.9.15), 11.21.10
- 12.9.15, 12.15.9, 12.16.10, 12.17.7, 12.18.5, 12.19.4, 12.20.3, 12.22.3
- 13.9.12, 13.11.9, 13.12.10, 13.13.7, 13.14.5, 13.15.4, 13.16.3, 13.18.3
- 14.6.12, 14.8.9, 14.9.10, 14.10.7, 14.11.5, 14.12.4, 14.13.3, 14.15.3
- 15.3.9, 15.4.10, 15.5.7, 15.6.5, 15.7.4, 15.8.3, 15.10.3
- 16.1.7, 16.2.5, 16.3.4, 16.4.3, 16.6.3

Abrufen des Status eines Datenbank-Aktivitätsstreams

Sie können den Status eines Aktivitätsstreams über die Konsole oder AWS CLI abrufen.

Konsole

Abrufen des Status eines Datenbank-Aktivitäts-Streams

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken und dann den Link des DB-Clusters aus.
3. Wählen Sie die Registerkarte Konfiguration aus und prüfen Sie die Statusangabe für Datenbank-Aktivitätsstream.

AWS CLI

Sie können die Aktivitätsstream-Konfiguration für einen DB-Cluster als Antwort auf eine CLI-Anforderung describe-db-clusters abrufen.

Das folgende Beispiel beschreibt *my-cluster*.

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

Das folgende Beispiel zeigt eine JSON-Antwort. Die folgenden Felder werden angezeigt:

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`

- `ActivityStreamStatus`
- `ActivityStreamMode`
-

Diese Felder sind für Aurora PostgreSQL und Aurora MySQL identisch, außer dass `ActivityStreamMode` für Aurora MySQL immer `async` ist, während es für Aurora PostgreSQL `sync` oder `async` sein kann.

```
{
    "DBClusters": [
        {
      "DBClusterIdentifier": "my-cluster",
            ...
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-
A6TSYXITZCZXJHIRVFUBZ5LTWY",
            "ActivityStreamStatus": "starting",
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
            "ActivityStreamMode": "async",
            "DbClusterResourceId": "cluster-ABCD123456"
            ...
        }
    ]
}
```

RDS-API

Sie können die Aktivitätsstream-Konfiguration für einen DB-Cluster als Antwort auf einen DescribeDBClusters -Vorgang abrufen.

Stoppen eines Datenbankaktivitäts-Streams

Sie können den Status eines Aktivitäts-Streams über die Konsole oder AWS CLI abrufen.

Wenn Sie Ihren DB-Cluster löschen, wird der Aktivitätsstream gestoppt und der zugrunde liegende Amazon-Kinesis-Stream wird automatisch gelöscht.

Konsole

So deaktivieren Sie einen Aktivitäts-Stream:

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie das DB-Cluster aus, für das Sie den Datenbankaktivitäts-Stream stoppen möchten.
4. Wählen Sie für Actions (Aktionen) die Option Stop activity stream (Aktivitäts-Stream stoppen) aus. Das Fenster Database Activity Stream (Datenbankaktivitäts-Stream) wird aufgerufen.

    a. Wählen Sie Sofort aus.

    Wenn Sie Sofort auswählen, wird der DB-Cluster sofort neu gestartet. Wenn Sie Während des nächsten Wartungsfensters auswählen, wird der DB-Cluster nicht sofort neu gestartet. In diesem Fall wird der Datenbankaktivitäts-Stream erst im nächsten Wartungsfenster gestoppt.

    b. Klicken Sie auf Weiter.

AWS CLI

Zum Stoppen von Datenbankaktivitäts-Streams für ein DB-Cluster konfigurieren Sie das DB-Cluster mit dem AWS CLI-Befehl stop-activity-stream. Bestimmen Sie die AWS-Region für das DB-Cluster mit dem Parameter `--region`. Der Parameter `--apply-immediately` ist optional.

Für Linux, macOS oder Unix:

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

Für Windows:

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

RDS-API

Um Datenbankaktivitäts-Streams für den DB-Cluster zu stoppen, konfigurieren Sie den Cluster mit der Operation StopActivityStream. Bestimmen Sie die AWS-Region für das DB-Cluster mit dem Parameter `Region`. Der Parameter `ApplyImmediately` ist optional.

Überwachen von Datenbankaktivitäts-Streams

Datenbankaktivitäts-Streams überwachen und melden Aktivitäten. Der Aktivitäts-Stream wird erfasst und an Amazon Kinesis übertragen. Von Kinesis aus können Sie den Aktivitäts-Stream überwachen, oder andere Dienste und Anwendungen können den Aktivitäts-Stream zur weiteren Analyse nutzen. Sie können den zugrunde liegenden Namen des Kinesis-Streams mit dem AWS CLI-Befehl `describe-db-clusters` oder der RDS-API-Operation `DescribeDBClusters` finden.

Aurora verwaltet den Kinesis Stream wie folgt:

- Aurora erzeugt den Kinesis Stream automatisch mit einem Aufbewahrungszeitraum von 24 Stunden.
- Aurora skaliert den Kinesis-Stream bei Bedarf.
- Wenn Sie den Datenbankaktivitäts-Stream stoppen oder den DB-Cluster löschen, löscht Aurora den Kinesis-Stream.

Die folgenden Kategorien von Aktivitäten werden überwacht und in das Prüfprotokoll des Aktivitäts-Streams aufgenommen:

- SQL-Befehle – Alle SQL-Befehle werden geprüft, ebenso vorbereitete Anweisungen, integrierte Funktionen und Funktionen in PL/SQL. Aufrufe von gespeicherten Prozeduren werden überprüft. Alle SQL-Anweisungen, die in gespeicherten Prozeduren oder Funktionen ausgegeben werden, werden ebenfalls überprüft.
- Sonstige Datenbankinformationen – Die überwachte Aktivität umfasst die vollständige SQL-Anweisung, die Zeilenzahl der betroffenen Zeilen aus DML-Befehlen, Objekte, auf die zugegriffen wurde, und den eindeutigen Datenbanknamen. Für Aurora PostgreSQL überwachen Datenbankaktivitäts-Streams auch die Bindevariablen und die Parameter der gespeicherten Prozedur.

> **Important**
>
> Der vollständige SQL-Text jeder Anweisung ist im Prüfprotokoll des Aktivitäts-Streams sichtbar, inklusive aller sensiblen Daten. Datenbankbenutzerkennwörter werden jedoch redigiert, wenn Aurora sie wie in der folgenden SQL-Anweisung aus dem Kontext ermitteln kann.

```
ALTER ROLE role-name WITH password
```

- Verbindungsinformationen – Die überwachte Aktivität umfasst Sitzungs- und Netzwerkinformationen, die Server-Prozess-ID und Beendigungscodes.

Wenn ein Aktivitätsstream während der Überwachung Ihrer DB-Instance fehlschlägt, werden Sie über RDS-Ereignisse benachrichtigt.

In den folgenden Abschnitten können Sie auf Datenbankaktivitäts-Streams zugreifen, diese prüfen und verarbeiten.

Themen

- Zugriff auf einen Aktivitäts-Stream von Amazon Kinesis aus
- Prüfungsprotokoll-Inhalte und Beispiele für Datenbankaktivitäts-Streams
- JSON-Array databaseActivityEventList für Datenbankaktivitäts-Streams
- Verarbeitung eines Datenbankaktivitäts-Streams mit dem AWS SDK

Zugriff auf einen Aktivitäts-Stream von Amazon Kinesis aus

Wenn Sie einen Aktivitäts-Stream für einen DB-Cluster aktivieren, wird ein Kinesis-Stream für Sie erstellt. Von Kinesis aus können Sie die Datenbankaktivität in Echtzeit überwachen. Zur weiteren Analyse der Datenbankaktivität können Sie Ihren Kinesis-Stream mit Consumer-Anwendungen verbinden. Sie können den Stream auch mit Compliance-Management-Anwendungen wie Security Guardium von IBM oder SecureSphere Database Audit and Protection von Imperva oder verbinden.

Sie können entweder über die RDS- oder Kinesis-Konsole auf Ihren Kinesis-Stream zugreifen.

So greifen Sie über die RDS-Konsole auf einen Aktivitätsstream zu

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie den/die DB-Cluster aus, auf der Sie einen Aktivitätsstream gestartet haben.
4. Wählen Sie Konfiguration.
5. Wählen Sie unter Database activity stream (Datenbank-Aktivitätsstream) den Link unter Kinesis stream (Kinesis-Stream) aus.

6. Wählen Sie in der Kinesis-Konsole Monitoring (Überwachung) aus, um mit der Überwachung der Datenbankaktivität zu beginnen.

So greifen Sie über die Kinesis-Konsole auf einen Aktivitätsstream von Kinesis zu

1. Öffnen Sie die Kinesis-Konsole unter.https://console.aws.amazon.com/kinesis.
2. Wählen Sie Ihren Aktivitäts-Stream aus der Liste der Kinesis-Streams aus.

   Der Name eines Aktivitäts-Streams besteht aus dem Präfix `aws-rds-das-cluster-` gefolgt von der Ressourcen-ID des DB-Clusters. Im Folgenden wird ein Beispiel gezeigt.

   ```
   aws-rds-das-cluster-NHVOV4PCLWHGF52NP
   ```

   Um die Amazon-RDS-Konsole zum Ermitteln der Ressourcen-ID für den DB-Cluster zu verwenden, wählen Sie Ihren DB-Cluster aus der Liste der Datenbanken aus und wählen dann die Registerkarte Konfiguration aus.

   Um den vollständigen Kinesis-Stream-Namen für einen Aktivitäts-Stream mit AWS CLI zu finden, verwenden Sie eine describe-db-clusters -CLI-Anfrage und notieren den Wert von `ActivityStreamKinesisStreamName` in der Antwort.
3. Wählen Sie Monitoring (Überwachung) aus, um mit der Überwachung der Datenbankaktivität zu beginnen.

Weitere Informationen zur Verwendung von Amazon Kinesis finden Sie unter Was sind Amazon Kinesis Data Streams?.

Prüfungsprotokoll-Inhalte und Beispiele für Datenbankaktivitäts-Streams

Überwachte Ereignisse werden im Datenbankaktivitäts-Stream als JSON-Zeichenfolgen dargestellt. Die Struktur besteht aus einem JSON-Objekt mit einem `DatabaseActivityMonitoringRecord`, der wiederum ein Array von Aktivitätsereignissen `databaseActivityEventList` enthält.

> **Note**
>
> Für Datenbankaktivitäts-Streams enthält das JSON-Array `paramList` keine Nullwerte aus Anwendungen im Ruhezustand.

Themen

- Prüfungsprotokollbeispiele für Aktivitäts-Streams
- JSON-Objekt DatabaseActivityMonitoringRecords
- databaseActivityEvents-JSON-Objekt

Prüfungsprotokollbeispiele für Aktivitäts-Streams

Im Folgenden sehen Sie Beispiele für entschlüsselte JSON-Prüfprotokolle von Aktivitätsereignisdatensätzen.

Example Aktivitätsereignisdatensatz einer Aurora-PostgreSQL-CONNECT SQL-Anweisung

Im Folgenden sehen Sie einen Aktivitätsereignisdatensatz einer Anmeldung unter Verwendung einer `CONNECT`-SQL-Anweisung (`command`) durch einen psql-Client (`clientApplication`).


```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-10-30 00:39:49.940668+00",
          "logTime": "2019-10-30 00:39:49.990579+00",
          "statementId": 1,
          "substatementId": 1,
          "objectType": null,
          "command": "CONNECT",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "49804",
          "sessionId": "5ce5f7f0.474b",
          "rowCount": null,
          "commandText": null,
          "paramList": [],
          "pid": 18251,
```

```
          "clientApplication": "psql",
          "exitCode": null,
          "class": "MISC",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aktivitätsereignisdatensatz einer Aurora MySQL-CONNECT SQL-Anweisung

Im Folgenden sehen Sie einen Aktivitätsereignisdatensatz einer Anmeldung unter Verwendung einer CONNECT-SQL-Anweisung (`command`) durch einen mysql-Client (`clientApplication`).

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:13.267214+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"rdsadmin",
      "databaseName":"",
      "remoteHost":"localhost",
      "remotePort":"11053",
      "command":"CONNECT",
      "commandText":"",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"",
      "statementId":0,
      "substatementId":1,
```

```
      "exitCode":"0",
      "sessionId":"725121",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:13.267207+00",
      "endTime":"2020-05-22 18:07:13.267213+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Example Aktivitätsereignisdatensatz einer Aurora PostgreSQL CREATE TABLE-Anweisung

Im Folgenden sehen Sie ein Beispiel eines `CREATE TABLE`-Ereignisses für Aurora PostgreSQL.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:36:54.403455+00",
          "logTime": "2019-05-24 00:36:54.494235+00",
          "statementId": 2,
          "substatementId": 1,
          "objectType": null,
          "command": "CREATE TABLE",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
```

```
          "rowCount": null,
          "commandText": "create table my_table (id serial primary key, name
 varchar(32));",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "DDL",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aktivitätsereignisdatensatz einer Aurora-MySQL-CREATE TABLE-Anweisung

Das folgende Beispiel zeigt eine `CREATE TABLE`-Anweisung für Aurora MySQL. Die Operation wird als zwei separate Ereignisdatensätze dargestellt. Das eine Ereignis verfügt über einen Wert `"class":"MAIN"`. Das andere über einen Wert `"class":"AUX"`. Die Nachrichten können in beliebiger Reihenfolge eintreffen. Das `logTime`-Feld des `MAIN`-Ereignisses ist immer früher als die `logTime`-Felder der entsprechenden `AUX`-Ereignisse.

Im folgenden Beispiel wird das Ereignis mit einem `class`-Wert von `MAIN` gezeigt.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.250221+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
```

```
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"CREATE TABLE test1 (id INT)",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.250222+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Im folgenden Beispiel wird das entsprechende Ereignis mit einem `class`-Wert von AUX gezeigt.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
```

```
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Example Aktivitätsereignisdatensatz einer Aurora PostgreSQL SELECT-Anweisung

Das folgende Beispiel zeigt ein SELECT-Ereignis .

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:39:49.920564+00",
          "logTime": "2019-05-24 00:39:49.940668+00",
          "statementId": 6,
          "substatementId": 1,
```

```
          "objectType": "TABLE",
          "command": "SELECT",
          "objectName": "public.my_table",
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": 10,
          "commandText": "select * from my_table;",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "READ",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "TABLE",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "SELECT",
            "commandText": "select * from [testDB].[dbo].[TestTable]",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
```

```
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:24:59.9422268+00",
            "netProtocol": null,
            "objectName": "TestTable",
            "objectType": "TABLE",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 62,
            "startTime": null,
            "statementId": "0x03baed90412f564fad640ebe51f89b99",
            "substatementId": 1,
            "transactionId": "4532935",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 2,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
                "target_server_principal_name": "",
                "schema_name": "dbo",
                "is_column_permission": true,
                "object_id": 581577110,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
```

```
                "server_principal_sid":
 "0x0105000000000005150000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

Example Aktivitätsereignisdatensatz einer Aurora MySQL-SELECT-Anweisung

Das folgende Beispiel zeigt ein `SELECT`-Ereignis.

Im folgenden Beispiel wird das Ereignis mit einem `class`-Wert von `MAIN` gezeigt.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986467+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"SELECT * FROM test1 WHERE id < 28",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"726571",
      "rowCount":2,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
```

```
      "endTime":"2020-05-22 18:29:57.986467+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Im folgenden Beispiel wird das entsprechende Ereignis mit einem `class`-Wert von AUX gezeigt.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986399+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"READ",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"726571",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986399+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
```

```
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

JSON-Objekt DatabaseActivityMonitoringRecords

Die Datenbank-Aktivitätsereignisdatensätze befinden sich in einem JSON-Objekt, das die folgenden Informationen enthält.

JSON-Feld	Datentyp	Beschreibung
`type`	string	Der Typ des JSON-Datensatzes. Der Wert ist `DatabaseActivityMonitoringRecords` .
`version`	string	Die Version der Datenbank-Aktivitätsüberwachungsdatensätze. Die Version der generierten Datenbank-Aktivitätsdatensätze hängt von der Engine-Version des DB-Clusters ab: • Datenbank-Aktivitätsdatensätze der Version 1.1 werden für Aurora PostgreSQL-DB-Cluster generiert, auf denen die Engine-Versionen 10.10 und höhere Nebenversionen sowie die Engine-Versionen 11.5 und höher ausgeführt werden. • Datenbank-Aktivitätsdatensätze der Version 1.0 werden für Aurora PostgreSQL-DB-Cluster generiert, auf denen die Engine-Versionen 10.7 und 11.4 ausgeführt werden.

JSON-Feld	Datentyp	Beschreibung
		Alle folgenden Felder befinden sich sowohl in Version 1.0 als auch in Version 1.1, sofern nicht ausdrücklich angegeben.
databaseActivityEvents	Zeichenfolge	Ein JSON-Objekt, das die Aktivitätsereignisse enthält.
Schlüssel	Zeichenfolge	Ein Verschlüsselungsschlüssel, den Sie zum Entschlüsseln des JSON-Array databaseActivityEventList verwenden

databaseActivityEvents-JSON-Objekt

Das `databaseActivityEvents`-JSON-Objekt enthält die folgenden Informationen.

Felder der obersten Ebene im JSON-Datensatz

Jedes Ereignis im Prüfprotokoll wird in einen Datensatz im JSON-Format verpackt. Dieser Datensatz enthält die folgenden Felder.

type

Dieses Feld hat immer den Wert `DatabaseActivityMonitoringRecords`.

Version

Dieses Feld stellt die Version des Datenprotokolls oder des Vertrags für die Datenbankaktivität dar. Es definiert, welche Felder verfügbar sind.

Version 1.0 stellt die Unterstützung der ursprünglichen Datenaktivitäts-Streams für die Aurora PostgreSQL-Versionen 10.7 und 11.4 dar. Version 1.1 stellt die Unterstützung der Datenaktivitäts-Streams für die Aurora PostgreSQL-Versionen ab 10.10 und ab Aurora PostgreSQL-Version 11.5 dar. Version 1.1 enthält die zusätzlichen Felder `errorMessage` und `startTime`. Version 1.2 stellt die Unterstützung der Datenaktivitäts-Streams für Aurora MySQL 2.08 und höher dar. Version 1.2 enthält die zusätzlichen Felder `endTime` und `transactionId`.

databaseActivityEvents

Eine verschlüsselte Zeichenfolge, die ein oder mehrere Aktivitätsereignisse darstellt. Sie wird als Base64-Byte-Array dargestellt. Wenn Sie die Zeichenfolge entschlüsseln, ist das Ergebnis ein Datensatz im JSON-Format mit Feldern, wie in den Beispielen in diesem Abschnitt gezeigt.

Schlüssel

Der verschlüsselte Datenschlüssel, der zum Verschlüsseln der `databaseActivityEvents`-Zeichenfolge verwendet wird. Dies ist dieselbe AWS KMS key, die Sie beim Starten des Datenbankaktivitätsstroms angegeben haben.

Im folgenden Beispiel wird das Format dieses Datensatzes gezeigt.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

Führen Sie die folgenden Schritte aus, um den Inhalt des `databaseActivityEvents`-Feldes zu entschlüsseln:

1. Entschlüsseln Sie den Wert im JSON-Feld `key` mit dem KMS-Schlüssel, den Sie beim Starten des Datenbankaktivitätsstroms angegeben haben. Dadurch wird der Datenverschlüsselungsschlüssel im Klartext zurückgegeben.
2. Base64-dekodieren Sie den Wert im `databaseActivityEvents`-JSON-Feld, um den Verschlüsselungstext der Prüfungsnutzlast im Binärformat zu erhalten.
3. Entschlüsseln Sie den binären Verschlüsselungstext mit dem Datenverschlüsselungsschlüssel, den Sie im ersten Schritt dekodiert haben.
4. Dekomprimieren Sie die entschlüsselte Nutzlast.
   - Die verschlüsselte Nutzlast befindet sich im `databaseActivityEvents`-Feld.
   - Das `databaseActivityEventList`-Feld enthält ein Array von Prüfdatensätzen. Die `type`-Felder im Array können `record` oder sein `heartbeat`.

Der Prüfprotokoll-Aktivitätsereignisdatensatz ist ein JSON-Objekt mit folgenden Informationen.

JSON-Feld	Datentyp	Beschreibung
`type`	string	Der Typ des JSON-Datensatzes. Der Wert ist `DatabaseActivityMonitoringRecord` .
`clusterId`	string	Die Ressourcen-ID des DB-Clusters. Sie entspricht dem DB-Clusterattribut `DbClusterResourceId` .
`instanceId`	string	Die Ressourcen-ID der DB-Instance. Sie dem DB-Instance-Attribut `DbiResourceId` .
JSON-Array databaseActivityEventList	string	Ein Array von Aktivitätsprüfdatensätzen oder Heartbeat-Nachrichten.

JSON-Array databaseActivityEventList für Datenbankaktivitäts-Streams

Die Prüfprotokollnutzlast ist ein verschlüsseltes JSON-Array `databaseActivityEventList`. In der folgenden Tabelle sind die Felder für jedes Aktivitätsereignis im entschlüsselten Array `DatabaseActivityEventList` eines Prüfprotokolls alphabetisch aufgelistet. Die Felder unterscheiden sich je nachdem, ob Sie Aurora PostgreSQL oder Aurora MySQL verwenden. Näheres entnehmen Sie bitte der Tabelle, die für Ihre Datenbank-Engine gilt.

> **Important**
>
> Die Ereignisstruktur kann sich ändern. Aurora könnte in Zukunft neue Felder zu Aktivitätsereignissen hinzufügen. Stellen Sie bei Anwendungen, welche die JSON-Daten analysieren, sicher, dass Ihr Code unbekannte Feldnamen ignorieren oder entsprechende Aktionen durchführen kann.

databaseActivityEventList-Felder für Aurora PostgreSQL

Im Folgenden sind `databaseActivityEventList`-Felder für Aurora PostgreSQL aufgeführt.

Feld	Datentyp	Beschreibung
`class`	string	Die Aktivitätsereignisklasse. Gültige Werte für Aurora PostgreSQL sind die folgenden: • `ALL` • `CONNECT` – Ein Verbindungs- oder Verbindungstrennungsereignis. • `DDL` – eine DDL-Anweisung, die nicht in der Liste der Anweisungen für die Klasse `ROLE` enthalten ist. • `FUNCTION` – ein Funktionsaufruf oder ein `DO`-Block. • `MISC` – ein sonstiger Befehl wie z. B. `DISCARD`, `FETCH`, `CHECKPOINT` oder `VACUUM`. • `NONE` • `READ` – eine Anweisung `SELECT` oder `COPY`, wenn es sich bei der Quelle um eine Relation oder Abfrage handelt. • `ROLE` – eine Anweisung in Zusammenhang mit Rollen und Berechtigungen wie z. B. `GRANT`, `REVOKE` und `CREATE/ALTER/DROP ROLE`. • `WRITE` – eine Anweisung `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, oder `COPY`, wenn das Ziel eine Relation ist.
`clientApplication`	string	Die Anwendung, die der Client laut Meldung für die Verbindung verwendet hat. Der Client muss diese Informationen nicht angeben, der Wert kann daher Null sein.
`command`	string	Der Name des SQL-Befehls ohne Befehlsdetails.
`commandText`	string	Die vom Benutzer übergebene eigentliche SQL-Anweisung. Bei Aurora PostgreSQL ist der Wert identisch mit der ursprünglichen SQL-Anweisung. Dieses Feld wird für alle Arten von Datensätzen verwendet, mit Ausnahme von Verbindungs- oder Verbindungstrennungsdatensätzen, bei denen der Wert Null ist.

Feld	Datentyp	Beschreibung
		Important Der vollständige SQL-Text jeder Anweisung ist im Prüfprotokoll des Aktivitäts-Streams sichtbar, inklusive aller sensiblen Daten. Datenbankbenutzerkennwörter werden jedoch redigiert, wenn Aurora sie wie in der folgenden SQL-Anweisung aus dem Kontext ermitteln kann. `ALTER ROLE role-name WITH password`
`databaseName`	string	Die Datenbank, zu der der Benutzer eine Verbindung hergestellt hat.
`dbProtocol`	string	Das Datenbankprotokoll, z. B. `Postgres 3.0`.
`dbUserName`	string	Der Datenbankbenutzer, mit dem sich der Client authentifiziert hat.

Feld	Datentyp	Beschreibung
`errorMessage` (nur Datenbank-Aktivitätsdatensätze der Version 1.1)	string	Wenn ein Fehler aufgetreten ist, wird dieses Feld mit der Fehlermeldung gefüllt, die vom DB-Server generiert worden wäre. Der `errorMessage` -Wert ist null für normale Anweisungen, die nicht zu einem Fehler geführt haben. Ein Fehler wird als jede Aktivität definiert, die ein vom Client sichtbares PostgreSQL-Fehlerprotokollereignis mit einem Schweregrad von `ERROR` oder höher erzeugen würde. Weitere Informationen finden Sie unter PostgreSQL-Nachrichtenschweregrade. Beispielsweise erzeugen Syntaxfehler und Abfrageabbrüche eine Fehlermeldung. Interne PostgreSQL-Serverfehler wie Hintergrund-Checkpointer-Prozessfehler erzeugen keine Fehlermeldung. Datensätze für solche Ereignisse werden jedoch weiterhin ausgegeben, unabhängig von der Einstellung des Schweregrads des Protokolls. Dadurch wird verhindert, dass Angreifer die Protokollierung deaktivieren, um eine Erkennung zu vermeiden. Siehe auch das Feld `exitCode`.
`exitCode`	int	Ein Wert, der für einen Sitzungsbeendigungs-Datensatz verwendet wird. Bei einer sauberen Beendigung ist hier der Beendigungscode enthalten. In manchen Fehlersituationen kann nicht immer ein Beendigungscode erhalten werden. Beispiele: `exit()` von PostgreSQL oder Ausführung eines Befehls wie `kill -9` durch einen Operator. Wenn ein Fehler aufgetreten ist, zeigt das `exitCode`-Feld den SQL-Fehlercode `SQLSTATE` an, wie in PostgreSQL-Fehlercodes aufgeführt. Siehe auch das Feld `errorMessage` .

Feld	Datentyp	Beschreibung
`logTime`	string	Ein Zeitstempel wie im Prüfcodepfad aufgezeichnet. Dies stellt die Endzeit der SQL-Anweisungsausführung dar. Siehe auch das Feld `startTime` .
`netProtocol`	string	Das Netzwerkkommunikationsprotokoll.
`objectName`	string	Der Name des Datenbankobjekts, wenn die SQL-Anweisung für eines ausgeführt wird. Dieses Feld wird nur verwendet, wenn die SQL-Anweisung für ein Datenbankobjekt ausgeführt wird. Falls die SQL-Anweisung nicht für ein Objekt ausgeführt wird, lautet dieser Wert Null.
`objectType`	string	Der Datenbankobjekttyp wie z. B. Tabelle, Index, Ansicht usw. Dieses Feld wird nur verwendet, wenn die SQL-Anweisung für ein Datenbankobjekt ausgeführt wird. Falls die SQL-Anweisung nicht für ein Objekt ausgeführt wird, lautet dieser Wert Null. Gültige Werte sind unter anderem: • `COMPOSITE TYPE` • `FOREIGN TABLE` • `FUNCTION` • `INDEX` • `MATERIALIZED VIEW` • `SEQUENCE` • `TABLE` • `TOAST TABLE` • `VIEW` • `UNKNOWN`
`paramList`	string	Ein Array durch Kommas getrennter Parameter, die an die SQL-Anweisung übergeben werden. Wenn die SQL-Anweisung keine Parameter beinhaltet, ist dieser Wert ein leeres Array.

Feld	Datentyp	Beschreibung
`pid`	int	Die Prozess-ID des Back-End-Prozesses, der für die Client-Verbindung zugewiesen wird.
`remoteHost`	string	Entweder die Client-IP-Adresse oder der Hostname. Was davon verwendet wird, ist bei Aurora PostgreSQL von der Parametereinstellung `log_hostname` der Datenbank abhängig. Der Wert `remoteHost` beinhaltet auch die Werte `[local]` und `localhost`, die auf Aktivität des `rdsadmin`-Benutzers hinweisen.
`remotePort`	Zeichenfolge	Die Portnummer des Clients.
`rowCount`	int	Die Anzahl der Tabellenzeilen, die von der SQL-Anweisung betroffen sind bzw. abgerufen werden. Dieses Feld wird nur für SQL-Anweisungen verwendet, bei denen es sich um DML-Anweisungen (DML = Data Manipulation Language) handelt. Falls die SQL-Anweisung keine DML-Anweisung ist, lautet dieser Wert Null.
`serverHost`	Zeichenfolge	Die Host-IP-Adresse des Datenbankservers. Der Wert `serverHost` beinhaltet auch die Werte `[local]` und `localhost`, die auf Aktivität des `rdsadmin`-Benutzers hinweisen.
`serverType`	Zeichenfolge	Der Datenbankservertyp, z. B `PostgreSQL`.
`serverVersion`	string	Die Datenbankserver-Version, z. B. `2.3.1` für Aurora PostgreSQL.
`serviceName`	string	Der Name des Service, beispielsweise `Amazon Aurora PostgreSQL-Compatible edition`.
`sessionId`	int	Eine pseudoeindeutige Sitzungskennung.
`sessionId`	int	Eine pseudoeindeutige Sitzungskennung.

Feld	Datentyp	Beschreibung
`startTime` (nur Datenbank-Aktivitätsdatensätze der Version 1.1)	string	Die Zeit, zu der die Ausführung für die SQL-Anweisung begann. Um die ungefähre Ausführungszeit der SQL-Anweisung zu berechnen, verwenden Sie `logTime - startTime` . Siehe auch das Feld `logTime`.
`statementId`	int	Eine ID für die SQL-Anweisung des Clients. Dieser Zähler auf Sitzungsebene erhöht sich mit jeder vom Client eingegebenen SQL-Anweisung.
`substatementId`	int	Eine ID für eine SQL-Unteranweisung. Dieser Wert zählt die enthaltenen Unteranweisungen für jede über das Feld `statementId` angegebene SQL-Anweisung.
`type`	string	Der Ereignistyp. Gültige Werte sind `record` oder `heartbeat` .

databaseActivityEventList-Felder für Aurora MySQL

Im Folgenden sind `databaseActivityEventList`-Felder für Aurora MySQL aufgeführt.

Feld	Datentyp	Beschreibung
`class`	string	Die Aktivitätsereignisklasse. Gültige Werte für Aurora MySQL sind die folgenden: • `MAIN` – das primäre Ereignis, das eine SQL-Anweisung darstellt. • `AUX` – ein zusätzliches Ereignis, das zusätzliche Details enthält. Beispielsweise kann eine Anweisung, mit der ein Objekt umbenannt wird, ein Ereignis der Klasse `AUX` aufweisen, das den neuen Namen wiedergibt.

Feld	Datentyp	Beschreibung
		Um `MAIN`- und `AUX`-Ereignisse zu finden, die derselben Anweisung entsprechen, suchen Sie nach verschiedenen Ereignissen, die dieselben Werte für das Feld `pid` und für das Feld `statementId` aufweisen.
`clientApplication`	string	Die Anwendung, die der Client laut Meldung für die Verbindung verwendet hat. Der Client muss diese Informationen nicht angeben, der Wert kann daher Null sein.

Feld	Datentyp	Beschreibung
command	string	Die allgemeine Kategorie der SQL-Anweisung. Die Werte für dieses Feld hängen vom Wert von a class. Wenn class MAIN ist, enthalten die Werte Folgendes: • CONNECT – wenn eine Client-Sitzung verbunden ist. • QUERY – eine SQL-Anweisung. Ebenfalls enthalten sind ein oder mehrere Ereignisse mit einem class-Wert von AUX. • DISCONNECT – wenn eine Client-Sitzung getrennt wird. • FAILED_CONNECT – wenn ein Client versucht, eine Verbindung herzustellen, dies aber nicht möglich ist. • CHANGEUSER – eine Statusänderung, die Teil des MySQL-Netzwerkprotokolls ist und nicht aus einer von Ihnen ausgegebenen Anweisung stammt. Wenn class AUX ist, enthalten die Werte Folgendes: • READ – eine Anweisung SELECT oder COPY, wenn es sich bei der Quelle um eine Relation oder Abfrage handelt. • WRITE – eine Anweisung INSERT, UPDATE, DELETE, TRUNCATE, oder COPY, wenn das Ziel eine Relation ist. • DROP – Löschen eines Objekts • CREATE – Erstellen eines Objekts. • RENAME – Umbenennen eines Objekts. • ALTER – Ändern der Eigenschaften eines Objekts.

Feld	Datentyp	Beschreibung
`commandText`	string	Bei Ereignissen mit dem `class`-Wert von `MAIN` stellt dieses Feld die tatsächliche, vom Benutzer eingegebene SQL-Anweisung dar. Dieses Feld wird für alle Arten von Datensätzen verwendet, mit Ausnahme von Verbindungs- oder Verbindungstrennungsdatensätzen, bei denen der Wert Null ist. Bei Ereignissen mit einem `class`-Wert von `AUX` enthält dieses Feld zusätzliche Informationen über die am Ereignis beteiligten Objekte. Bei Aurora MySQL wird Zeichen, z. B. Anführungszeichen, ein Backslash vorangestellt, der ein Escape-Zeichen darstellt. **Important** Der vollständige SQL-Text jeder Anweisung ist im Prüfprotokoll sichtbar, inklusive aller sensiblen Daten. Datenbankbenutzerkennwörter werden jedoch redigiert, wenn Aurora sie wie in der folgenden SQL-Anweisung aus dem Kontext ermitteln kann. `mysql> SET PASSWORD = 'my-password ';` **Note** Geben Sie aus Sicherheitsgründen ein anderes Passwort als hier angegeben an.
`databaseName`	Zeichenfolge	Die Datenbank, zu der der Benutzer eine Verbindung hergestellt hat.
`dbProtocol`	string	Das Datenbankprotokoll. Derzeit ist dieser Wert bei Aurora MySQL immer `MySQL`.

Feld	Datentyp	Beschreibung
`dbUserName`	string	Der Datenbankbenutzer, mit dem sich der Client authentifiziert hat.
`endTime` (nur Datenbank-Aktivitätsdatensätze der Version 1.2)	string	Die Zeit, zu der die Ausführung für die SQL-Anweisung endete. Sie wird im UTC-Format (Coordinated Universal Time) dargestellt. Um die Ausführungszeit der SQL-Anweisung zu berechnen, verwenden Sie `endTime - startTime` . Siehe auch das Feld `startTime` .
`errorMessage` (nur Datenbank-Aktivitätsdatensätze der Version 1.1)	string	Wenn ein Fehler aufgetreten ist, wird dieses Feld mit der Fehlermeldung gefüllt, die vom DB-Server generiert worden wäre. Der `errorMessage` -Wert ist null für normale Anweisungen, die nicht zu einem Fehler geführt haben. Ein Fehler wird als jede Aktivität definiert, die ein vom Client sichtbares MySQL-Fehlerprotokollereignis mit einem Schweregrad von `ERROR` oder höher erzeugen würde. Weitere Informationen finden Sie unter Fehlerprotokoll im MySQL-Referenzhandbuch. Beispielsweise erzeugen Syntaxfehler und Abfrageabbrüche eine Fehlermeldung. Interne MySQL-Serverfehler wie Hintergrund-Checkpointer-Prozessfehler erzeugen keine Fehlermeldung. Datensätze für solche Ereignisse werden jedoch weiterhin ausgegeben, unabhängig von der Einstellung des Schweregrads des Protokolls. Dadurch wird verhindert, dass Angreifer die Protokollierung deaktivieren, um eine Erkennung zu vermeiden. Siehe auch das Feld `exitCode`.

Feld	Datentyp	Beschreibung
`exitCode`	int	Ein Wert, der für einen Sitzungsbeendigungs-Datensatz verwendet wird. Bei einer sauberen Beendigung ist hier der Beendigungscode enthalten. In manchen Fehlersituationen kann nicht immer ein Beendigungscode erhalten werden. In solchen Fällen kann dieser Wert Null oder leer sein.
`logTime`	string	Ein Zeitstempel wie im Prüfcodepfad aufgezeichnet. Sie wird im UTC-Format (Coordinated Universal Time) dargestellt. Die genaueste Methode zum Berechnen der Anweisungsdauer finden Sie in den Feldern `startTime` und `endTime`.
`netProtocol`	string	Das Netzwerkkommunikationsprotokoll. Derzeit ist dieser Wert bei Aurora MySQL immer `TCP`.
`objectName`	string	Der Name des Datenbankobjekts, wenn die SQL-Anweisung für eines ausgeführt wird. Dieses Feld wird nur verwendet, wenn die SQL-Anweisung für ein Datenbankobjekt ausgeführt wird. Falls die SQL-Anweisung nicht für ein Objekt ausgeführt wird, ist dieser Wert leer. Um den vollständig qualifizierten Namen des Objekts zu erstellen, kombinieren Sie `databaseName` und `objectName`. Wenn die Abfrage mehrere Objekte umfasst, kann dieses Feld eine durch Komma getrennte Liste von Namen sein.
`objectType`	string	Der Datenbankobjekttyp, z. B. Tabelle, Index usw. Dieses Feld wird nur verwendet, wenn die SQL-Anweisung für ein Datenbankobjekt ausgeführt wird. Falls die SQL-Anweisung nicht für ein Objekt ausgeführt wird, lautet dieser Wert Null. Gültige Werte für Aurora MySQL sind unter anderem: • `INDEX` • `TABLE` • `UNKNOWN`

Feld	Datentyp	Beschreibung
`paramList`	string	Dieses Feld wird für Aurora MySQL nicht verwendet und ist immer Null.
`pid`	int	Die Prozess-ID des Back-End-Prozesses, der für die Client-Verbindung zugewiesen wird. Wenn der Datenbankserver neu gestartet wird, ändert sich die `pid` und der Zähler für das Feld `statementId` beginnt von vorn.
`remoteHost`	string	Entweder die IP-Adresse oder der Hostname des Clients, der die SQL-Anweisung ausgegeben hat. Was davon verwendet wird, ist bei Aurora MySQL von der Parametereinstellung `skip_name_resolve` der Datenbank abhängig. Der Wert `localhost` gibt die Aktivität des speziellen Benutzers `rdsadmin` an.
`remotePort`	string	Die Portnummer des Clients.
`rowCount`	int	Die Anzahl der Zeilen, die von der SQL-Anweisung zurückgegeben werden. Wenn eine SELECT-Anweisung beispielsweise 10 Zeilen zurückgibt, beträgt rowCount 10. Für INSERT- oder UPDATE-Anweisungen ist der RowCount 0.
`serverHost`	Zeichenfolge	Die Datenbankserver-Instance-ID.
`serverType`	Zeichenfolge	Der Datenbankservertyp, z. B `MySQL`.
`serverVersion`	string	Die Version des Datenbankservers. Derzeit ist dieser Wert bei Aurora MySQL immer `MySQL 5.7.12`.
`serviceName`	string	Name des Service. Derzeit ist dieser Wert bei Aurora MySQL immer `Amazon Aurora MySQL`.
`sessionId`	int	Eine pseudoeindeutige Sitzungskennung.

Feld	Datentyp	Beschreibung
`startTime` (nur Datenbank-Aktivitätsdatensätze der Version 1.1)	string	Die Zeit, zu der die Ausführung für die SQL-Anweisung begann. Sie wird im UTC-Format (Coordinated Universal Time) dargestellt. Um die Ausführungszeit der SQL-Anweisung zu berechnen, verwenden Sie `endTime - startTime` . Siehe auch das Feld `endTime`.
`statementId`	int	Eine ID für die SQL-Anweisung des Clients. Der Zähler erhöht sich mit jeder vom Client eingegebenen SQL-Anweisung. Der Zähler wird zurückgesetzt, wenn die DB-Instance neu gestartet wird.
`substatementId`	int	Eine ID für eine SQL-Unteranweisung. Dieser Wert ist 1 für Ereignisse mit der Klasse `MAIN` und 2 für Ereignisse mit der Klasse `AUX`. Verwenden Sie das `statementId` -Feld, um alle Ereignisse zu identifizieren, die von derselben Anweisung generiert werden.
`transactionId` (nur Datenbank-Aktivitätsdatensätze der Version 1.2)	int	Eine ID für eine Transaktion.
`type`	string	Der Ereignistyp. Gültige Werte sind `record` oder `heartbeat` .

Verarbeitung eines Datenbankaktivitäts-Streams mit dem AWS SDK

Sie können einen Aktivitäts-Stream mit Hilfe des AWS SDK programmgesteuert verarbeiten. Im Folgenden sehen Sie vollständig funktionsfähige Java- und Python-Beispiele für eine mögliche Verarbeitung des Kinesis-Datenstroms.

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
```

```
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-
resource-id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
```

```
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {
```

```
        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
        private static final Gson GSON = new
GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
            try {
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }

        private long nextCheckpointTimeInMillis;

        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
            for (final Record record : records) {
                processSingleBlob(record.getData());
            }

            if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
                checkpoint(checkpointer);
                nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
            }
        }

        @Override
        public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
            if (reason == ShutdownReason.TERMINATE) {
                checkpoint(checkpointer);
            }
        }
```

```
       private void processSingleBlob(final ByteBuffer bytes) {
           try {
               // JSON $Activity
               final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

               // Base64.Decode
               final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
               final byte[] decodedDataKey = Base64.decode(activity.key);

               Map<String, String> context = new HashMap<>();
               context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

               // Decrypt
               final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
               final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
               final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

               // GZip Decompress
               final byte[] decompressed = decompress(decrypted);
               // JSON $ActivityRecords
               final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

               // Iterate throught $ActivityEvents
               for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                   System.out.println(GSON.toJson(event));
               }
           } catch (Exception e) {
               // Handle error.
               e.printStackTrace();
           }
       }

       private static byte[] decompress(final byte[] src) throws IOException {
           ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
```

```
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }

        private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
            for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
                try {
                    checkpointer.checkpoint();
                    break;
                } catch (ShutdownException se) {
                    // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                    System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                    break;
                } catch (ThrottlingException e) {
                    // Backoff and re-attempt checkpoint upon transient failures
                    if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                        System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                        break;
                    } else {
                        System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                    }
                } catch (InvalidStateException e) {
                    // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                    System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                    break;
                }
                try {
                    Thread.sleep(BACKOFF_TIME_IN_MILLIS);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted sleep" + e);
                }
            }
        }
    }

    private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
```

```
        // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
        final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
                "BC", "DataKey", "AES/GCM/NoPadding");
        try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
             final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
            IOUtils.copy(decryptingStream, out);
            return out.toByteArray();
        }
    }

    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
":" + UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
                new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
CREDENTIALS_PROVIDER, workerId);

kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
                .recordProcessorFactory(new RecordProcessorFactory())
                .config(kinesisClientLibConfiguration)
                .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
APPLICATION_NAME, STREAM_NAME, workerId);

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
```

```
    }
}
```

Python

```
import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
RESOURCE_ID = '<external-resource-id>'      # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-cluster-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                        wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
```

```
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)


def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:dbc-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
```

```
        shard_iters = next_shard_iters


if __name__ == '__main__':
    main()
```

IAM-Richtlinien-Beispiele für Datenbankaktivitäts-Streams

Jeder Benutzer mit entsprechenden AWS Identity and Access Management (IAM-) Rollenberechtigungen für Datenbankaktivitätsstreams kann die Activity Stream-Einstellungen für eine erstellen, starten, beenden und ändern. Diese Aktionen sind im Prüfprotokoll des Streams enthalten. Um DBAs die Einhaltung von Vorschriften zu gewährleisten, empfehlen wir, dass Sie diese Rechte nicht gewähren.

Der Zugriff auf Datenbankaktivitäts-Streams wird mithilfe von IAM-Richtlinien festgelegt. Weitere Informationen zur Aurora-Authentifizierung finden Sie unter Identity and Access Management für Amazon Aurora. Weitere Informationen zum Erstellen von IAM-Richtlinien finden Sie unter Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff.

Example Richtlinie zum Zulassen der Konfiguration von Datenbankaktivitäts-Streams

Um Benutzern einen präzisen Zugriff auf die Änderung von Aktivitäts-Streams zu ermöglichen, verwenden Sie die servicespezifischen Operationskontextschlüssel `rds:StartActivityStream` und `rds:StopActivityStream` in einer IAM-Richtlinie. Das folgende IAM-Richtlinienbeispiel ermöglicht es einem Benutzer oder einer Rolle, Aktivitäts-Streams zu konfigurieren.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ConfigureActivityStreams",
            "Effect": "Allow",
            "Action": [
                "rds:StartActivityStream",
                "rds:StopActivityStream"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Example Richtlinie zum Zulassen des Startens von Datenbankaktivitäts-Streams

Das folgende IAM-Richtlinienbeispiel ermöglicht es einem Benutzer oder einer Rolle, Aktivitäts-Streams zu starten.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStartActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
    ]
}
```

Example Richtlinie zum Zulassen des Anhaltens von Datenbankaktivitäts-Streams

Das folgende IAM-Richtlinienbeispiel ermöglicht es einem Benutzer oder einer Rolle, Aktivitäts-Streams zu stoppen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
```

```
}
```

Example Richtlinie zum Ablehnen des Startens von Datenbankaktivitäts-Streams

Im folgenden IAM-Richtlinienbeispiel wird ein Benutzer oder eine Rolle daran gehindert, Aktivitäts-Streams zu starten.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example Richtlinie zum Ablehnen des Anhaltens von Datenbankaktivitäts-Streams

Im folgenden IAM-Richtlinienbeispiel wird ein Benutzer oder eine Rolle daran gehindert, Aktivitäts-Streams zu stoppen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

Überwachen von Bedrohungen mit Amazon GuardDuty RDS Protection für Amazon Aurora

Amazon GuardDuty ist ein Service zur Erkennung von Bedrohungen, der Sie beim Schutz Ihrer Konten, Container, Workloads und Daten in Ihrer AWS-Umgebung unterstützt. Mithilfe von Machine Learning (ML)-Modellen und Funktionen zur Erkennung von Anomalien und Bedrohungen überwacht GuardDuty kontinuierlich verschiedene Protokollquellen und Laufzeitaktivitäten, um potenzielle Sicherheitsrisiken und böswillige Aktivitäten in Ihrer Umgebung zu identifizieren und zu priorisieren.

Amazon GuardDuty RDS Protection analysiert und profiliert Anmeldeereignisse auf potenzielle Zugriffsbedrohungen, die für Ihre Amazon AuroraDatenbanken bestehen könnten. Wenn Sie RDS Protection aktivieren, verarbeitet GuardDuty RDS-Anmeldeereignisse aus Ihren Aurora-Datenbanken. RDS Protection überwacht diese Ereignisse und erstellt ein Profil für potenzielle interne Bedrohungen oder externe Akteure.

Weitere Informationen zum Aktivieren des RDS-Schutzes von GuardDuty finden Sie unter GuardDuty-RDS-Schutz im Amazon-GuardDuty-Benutzerhandbuch.

Wenn RDS Protection eine potenzielle Bedrohung erkennt, z. B. ein ungewöhnliches Muster bei erfolgreichen oder fehlgeschlagenen Anmeldeversuchen, generiert GuardDuty ein neues Ergebnis mit Details zur potenziell gefährdeten Datenbank. Sie können die Ergebnisdetails im Abschnitt mit der Ergebnisübersicht in der Amazon-GuardDuty-Konsole einsehen. Die Ergebnisdetails variieren je nach Ergebnistyp. Die primären Details, Ressourcentyp und Ressourcenrolle, bestimmen, welche Art von Informationen für jedes einzelne Ergebnis verfügbar ist. Weitere Informationen zu den allgemein verfügbaren Details für Ergebnisse und den Ergebnistypen finden Sie unter Ergebnisdetails bzw. Ergebnistypen des RDS-Schutzes von GuardDuty im Amazon-GuardDuty-Benutzerhandbuch.

Sie können die RDS-Protection-Funktion für jeden AWS-Konto in jeder AWS-Region, in der diese Funktion verfügbar ist, ein- oder ausschalten. Wenn RDS Protection nicht aktiviert ist, erkennt GuardDuty potenziell kompromittierte Aurora-Datenbanken nicht und gibt keine Details zu der Beeinträchtigung an.

Ein bestehendes GuardDuty-Konto kann den RDS-Schutz mit einem 30-tägigen Testzeitraum aktivieren. Für ein neues GuardDuty-Konto ist der RDS-Schutz bereits aktiviert und im kostenlosen 30-tägigen Testzeitraum enthalten. Weitere Informationen finden Sie unter Schätzung der GuardDuty-Kosten im Amazon-GuardDuty-Benutzerhandbuch.

Informationen zu den AWS-Regionen, in denen GuardDuty den RDS-Schutz noch nicht unterstützt, finden Sie unter Verfügbarkeit regionsspezifischer Funktionen im Amazon-GuardDuty-Benutzerhandbuch.

Informationen zu den von GuardDuty RDS Protection unterstützten Aurora-Datenbankversionen finden Sie unter Unterstützte Datenbanken von Amazon Aurora, Amazon RDS und Aurora Limitless im Benutzerhandbuch für Amazon GuardDuty.

Arbeiten mit Amazon Aurora MySQL

Amazon Aurora MySQL ist eine vollständig verwaltete, MySQL-kompatible, relationale Datenbank-Engine, die die Geschwindigkeit und Zuverlässigkeit kommerzieller hochwertiger Datenbanken mit der Einfachheit und Kosteneffizienz von Open-Source-Datenbanken kombiniert. Aurora MySQL ist ein Drop-In-Ersatz für MySQL und macht es einfach und kostengünstig, Ihre neuen und bestehenden MySQL-Implementierungen einzurichten, zu betreiben und zu skalieren, sodass Sie sich auf Ihr Unternehmen und Ihre Anwendungen konzentrieren können. Amazon RDS bietet Aurora die Verwaltung, indem es routinemäßige Datenbankaufgaben wie Bereitstellung, Patching, Sicherung, Wiederherstellung, Fehlererkennung und Reparatur übernimmt. Amazon RDS bietet auch Push-Button-Migrationstools zum Konvertieren Ihrer vorhandenen Amazon RDS für MySQL-Anwendungen in Aurora MySQL.

Themen

- Übersicht über Amazon Aurora MySQL
- Sicherheit in Amazon Aurora MySQL
- Aktualisieren von Anwendungen, um Verbindungen mit DB-Clustern von Aurora MySQL mithilfe neuer TLS-Zertifikate herzustellen
- Verwenden der Kerberos-Authentifizierung für Aurora MySQL
- Migrieren von Daten zu einem Amazon Aurora MySQL-DB-Cluster
- Verwalten von Amazon Aurora MySQL
- Optimieren von Aurora MySQL
- Parallel Query für Amazon Aurora MySQL
- Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster
- Replikation mit Amazon Aurora My SQL
- Verwendung der lokalen Schreibweiterleitung in einem DB-Cluster von Amazon Aurora MySQL
- Integrieren von Amazon Aurora MySQL in anderen AWS-Services
- Amazon Aurora MySQL-Labor-Modus
- Bewährte Methoden mit Amazon Aurora MySQL
- Beheben von Problemen mit der Datenbankleistung von Amazon Aurora MySQL
- Amazon Aurora MySQL-Referenz
- Datenbank-Engine-Updates für Amazon Aurora MySQL

Übersicht über Amazon Aurora MySQL

In den folgenden Abschnitten finden Sie eine Übersicht über Amazon Aurora MySQL.

Themen

- Amazon Aurora MySQL-Leistungserweiterungen
- Amazon Aurora MySQL und raumbezogene Daten
- Aurora mit MySQL Version 3 ist kompatibel mit MySQL 8.0
- Aurora-MySQL-Version 2, kompatibel mit MySQL 5.7

Amazon Aurora MySQL-Leistungserweiterungen

Amazon Aurora beinhaltet Leistungserweiterungen, um den verschiedenen Bedürfnissen von kommerziellen High-End-Datenbanken gerecht zu werden.

Schnelles Einfügen

Schnelles Einfügen beschleunigt parallele Insert-Vorgänge, geordnet nach Primärschlüssel, und wird speziell auf die Statements `LOAD DATA` und `INSERT INTO ... SELECT ...` angewandt. Schnelles Einfügen speichert die Position eines Cursors in einem Indexdurchlauf beim Ausführen des Statements zwischen. Dies verhindert ein unnötiges erneutes Durchlaufen des Index.

Schnelles Einfügen ist nur für reguläre InnoDB-Tabellen in Aurora MySQL Version 3.03.2 und höher aktiviert. Diese Optimierung funktioniert nicht für temporäre InnoDB-Tabellen. Sie ist in Aurora MySQL Version 2 für alle 2.11- und 2.12-Versionen deaktiviert. Die Optimierung für schnelles Einfügen funktioniert nur, wenn die Adaptive-Hash-Index-Optimierung deaktiviert ist.

Sie können die folgenden Metrik überwachen, um die Effektivität von schnellem Einfügen für Ihr DB-Cluster zu bestimmen:

- `aurora_fast_insert_cache_hits`: Ein Zähler, der erhöht wird, wenn der zwischengespeicherte Cursor erfolgreich abgerufen und bestätigt wurde.
- `aurora_fast_insert_cache_misses`: Ein Zähler, der erhöht wird, wenn der zwischengespeicherte Cursor nicht mehr gültig ist und Aurora einen normalen Index-Durchlauf durchführt.

Sie können den aktuellen Wert der Metriken von schnellem Einfügen mithilfe des folgenden Befehls abrufen:

```
mysql> show global status like 'Aurora_fast_insert%';
```

Sie erhalten eine Ausgabe, die dem Folgenden ähnelt:

```
+---------------------------------+-----------+
| Variable_name                   | Value     |
+---------------------------------+-----------+
| Aurora_fast_insert_cache_hits   | 3598300   |
| Aurora_fast_insert_cache_misses | 436401336 |
+---------------------------------+-----------+
```

Amazon Aurora MySQL und raumbezogene Daten

Die folgende Liste bietet eine Übersicht über die wichtigsten räumlichen Aurora MySQL-Funktionen und erläutert, wie diese den räumlichen Funktionen in MySQL entsprechen.

- Aurora MySQL Version 2 unterstützt dieselben räumlichen Datentypen und Funktionen für räumliche Beziehungen wie MySQL 5.7. Weitere Informationen zu diesen Datentypen und -Funktionen finden Sie unter Räumliche Datentypen und Geofunktionen in der MySQL 5.7-Dokumentation.
- Aurora MySQL Version 3 unterstützt dieselben räumlichen Datentypen und Funktionen für räumliche Beziehungen wie MySQL 8.0 Weitere Informationen zu diesen Datentypen und -Funktionen finden Sie unter Räumliche Datentypen und Geofunktionen in der MySQL 8.0-Dokumentation.
- Aurora MySQL unterstützt die räumliche Indizierung in InnoDB-Tabellen. Die räumliche Indizierung verbessert die Abfrageleistung in großen Datensätzen für Abfragen von räumlichen Daten. In MySQL ist die räumliche Indizierung für InnoDB-Tabellen in MySQL 5.7 und 8.0 verfügbar.

  Aurora MySQL verwendet einen anderen Ansatz für die räumliche Indizierung als MySQL für hohe Leistung bei räumlichen Abfragen. Die räumliche Aurora-Indeximplementierung verwendet die raumausfüllende Kurve eines B-Baums; dadurch soll eine höhere Leistung für räumliche Bereichsscans erzielt werden als mit einem R-Baum.

> **Note**
>
> In Aurora MySQL kann eine Transaktion in einer Tabelle mit einem räumlichen Index, der für eine Spalte mit einer räumlichen Referenz-ID (SRID) definiert ist, nicht in einen Bereich eingefügt werden, der für die Aktualisierung durch eine andere Transaktion ausgewählt wurde.

Die folgenden Anweisungen aus der Data Definition Language (DDL) bieten Unterstützung für das Erstellen von Indizes in Spalten, die raumbezogene Datentypen verwenden.

CREATE TABLE

Sie können die `SPATIAL INDEX`-Schlüsselwörter in einer `CREATE TABLE`-Anweisung verwenden, um einen räumlichen Index zu einer Spalte in einer neuen Tabelle hinzuzufügen. Im Folgenden sehen Sie ein Beispiel.

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

ALTER TABLE

Sie können die `SPATIAL INDEX`-Schlüsselwörter in einer `ALTER TABLE`-Anweisung verwenden, um einen räumlichen Index zu einer Spalte in einer vorhandenen Tabelle hinzuzufügen. Im Folgenden sehen Sie ein Beispiel.

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

CREATE INDEX

Sie können das `SPATIAL`-Schlüsselwort in einer `CREATE INDEX`-Anweisung verwenden, um einen räumlichen Index zu einer Spalte in einer vorhandenen Tabelle hinzuzufügen. Im Folgenden sehen Sie ein Beispiel.

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

Aurora mit MySQL Version 3 ist kompatibel mit MySQL 8.0

Sie können Aurora MySQL Version 3 verwenden, um die neuesten MySQL-kompatiblen Funktionen, Leistungsverbesserungen und Bugfixes zu erhalten. Im Folgenden erfahren Sie mehr über Aurora MySQL Version 3 mit MySQL 8.0 Kompatibilität. Sie können lernen, wie Sie Ihre Cluster und Anwendungen auf Aurora MySQL Version 3 aktualisieren.

Einige Aurora-Funktionen wie Aurora Serverless v2 erfordern Aurora MySQL Version 3.

Themen

- Funktionen aus der Community Edition von MySQL 8.0
- Aurora MySQL Version 3 als Voraussetzung für Aurora MySQL Serverless v2
- Versionshinweise für Aurora MySQL Version 3
- Neue -Optimierungen für parallele Abfragen
- Optimierungen reduzieren die Neustartzeit der Datenbank
- Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3
- Vergleich von Aurora-MySQL-Version 2 und Aurora-MySQL-Version 3
- Vergleich von Aurora-MySQL-Version 3 und MySQL 8.0 Community Edition
- Upgrade auf Aurora MySQL Version 3

Funktionen aus der Community Edition von MySQL 8.0

Die erste Version von Aurora MySQL Version 3 ist mit der Community Edition von MySQL 8.0.23 kompatibel. MySQL 8.0 führt mehrere neue Funktionen ein, darunter die folgenden:

- Unterstützung der Atomic Data Definition Language (DDL) Weitere Informationen finden Sie unter Unterstützung der Atomic Data Definition Language (DDL).
- JSON-Funktionen Weitere Informationen zur Nutzung finden Sie unterJSON-FunktionenimMySQL-Referenzhandbuchaus.
- Fensterfunktionen. Weitere Informationen zur Nutzung finden Sie unterFensterfunktionenimMySQL-Referenzhandbuchaus.
- Gemeinsame Tabellenausdrücke (CTEs) unter Verwendung der`WITH`-Klausel. Weitere Informationen zur Nutzung finden Sie unterWITH (Allgemeine Tabellenausdrücke)imMySQL-Referenzhandbuchaus.

- Optimiert`ADD COLUMN`und`RENAME COLUMN`-Klauseln für`ALTER TABLE`Estatement. Diese Optimierungen werden „Instant DDL“ genannt. Aurora MySQL Version 3 ist mit der MySQL-Instant-DDL-Funktion der Community kompatibel. Die ehemalige Aurora Fast DDL-Funktion wird nicht verwendet. Informationen zur Verwendung für Instant DDL finden Sie unterSofortige DDL (Aurora MySQL Version 3)aus.
- Absteigende, funktionale und unsichtbare Indizes. Weitere Informationen zur Nutzung finden Sie unterUnsichtbare Indizes,Absteigende Indizes, undCREATE INDEX-AnweisungimMySQL-Referenzhandbuchaus.
- Rollenbasierte Berechtigungen, die durch SQL-Anweisungen gesteuert werden. Weitere Informationen zu Änderungen am Berechtigungsmodell finden Sie unterRollenbasiertes Berechtigungsmodellaus.
- `NOWAIT`und`SKIP LOCKED`-Klauseln mit`SELECT ... FOR SHARE`Estatement. Diese Klauseln vermeiden es, darauf zu warten, dass andere Transaktionen Zeilensperren freigeben. Weitere Informationen zu Lesesperren finden Sie im MySQL-Referenzhandbuch.
- Verbesserungen der Binärprotokollreplikation (binlog). Die Aurora MySQL-Details finden Sie unterBinäre Protokoll-Replikationaus. Insbesondere können Sie eine gefilterte Replikation durchführen. Informationen zur Verwendung zur gefilterten Replikation finden Sie unterSo bewerten Server Replikationsfilterregeln imMySQL-Referenzhandbuchaus.
- Hinweise Einige der MySQL 8.0-kompatiblen Hinweise wurden bereits auf Aurora MySQL Version 2 zurückportiert. Weitere Informationen zur Sicherheit im Zusammenhang mit Aurora MySQL finden Sie unter Aurora-MySQL-Hinweise. Eine vollständige Liste der Hinweise in MySQL 8.0 finden Sie unterOptimierungshinweiseimMySQL-Referenzhandbuchaus.

Die vollständige Liste der Funktionen, die zur MySQL 8.0 Community Edition hinzugefügt wurden, finden Sie im BlogbeitragDie vollständige Liste der neuen Funktionen in MySQL 8.0aus.

Aurora MySQL Version 3 enthält auch Änderungen an Schlüsselwörtern für inklusive Sprache, die von der Community MySQL 8.0.26 zurückportiert wurden. Einzelheiten zu diesen Änderungen finden Sie unterInklusive Sprachänderungen für Aurora MySQL Version 3aus.

Aurora MySQL Version 3 als Voraussetzung für Aurora MySQL Serverless v2

Aurora MySQL Version 3 ist Voraussetzung für alle DB-Instances in einem Aurora-MySQL-Serverless-v2-Cluster. Aurora MySQL Serverless v2 enthält Unterstützung für Reader-Instances in einem DB-Cluster und andere Aurora-Funktionen, die für Aurora MySQL Serverless v1 nicht

verfügbar sind. Diese Version bietet auch eine schnellere und stärker granulare Skalierung als Aurora MySQL Serverless v1.

Versionshinweise für Aurora MySQL Version 3

Die Versionshinweise für alle Versionen von Aurora MySQL Version 3 finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora MySQL Version 3 in den Versionshinweisen für Aurora MySQL.

Neue -Optimierungen für parallele Abfragen

Die parallele Aurora-Abfrageoptimierung gilt nun für mehr SQL-Operationen:

- Die parallele Abfrage gilt jetzt für Tabellen, die die Datentypen enthaltenTEXT,BLOB,JSON,GEOMETRY, undVARCHARundCHARlänger als 768 Bytes.
- Parallele Abfragen können Abfragen mit partitionierten Tabellen optimieren.
- Eine parallele Abfrage kann Abfragen mit Aggregatfunktionsaufrufen in der Auswahlliste und derHAVING-Klausel.

Weitere Informationen zu Enhanced Monitoring finden Sie unter Upgrade paralleler Abfrage-Cluster auf Aurora-MySQL-Version 3. Allgemeine Informationen zu Aurora-Sicherungen finden Sie unter Parallel Query für Amazon Aurora MySQL.

Optimierungen reduzieren die Neustartzeit der Datenbank

Ihr Aurora-MySQL-DB-Cluster muss sowohl bei geplanten als auch bei ungeplanten Ausfällen hochverfügbar sein.

Datenbankadministratoren müssen gelegentlich Datenbankwartungen durchführen.
Diese Wartungsmaßnahmen umfassen Datenbank-Patches, Upgrades, Änderungen von Datenbankparametern, die einen manuellen Neustart erfordern, die Durchführung eines Failovers, um den Zeitaufwand für Instance-Klassenänderungen zu reduzieren, usw. Solche geplanten Aktionen machen Ausfallzeiten erforderlich.

Ausfallzeiten können jedoch auch durch ungeplante Aktionen verursacht werden, beispielsweise durch einen unerwarteten Failover aufgrund eines zugrunde liegenden Hardwarefehlers oder durch die Drosselung von Datenbankressourcen. Alle diese geplanten und ungeplanten Aktionen führen zu einem Neustart der Datenbank.

In Aurora MySQL Version 3.05 und höher haben wir Optimierungen eingeführt, die die Neustartzeit der Datenbank reduzieren. Diese Optimierungen sorgen für bis zu 65 % weniger Ausfallzeiten als ohne Optimierungen und weniger Unterbrechungen Ihrer Datenbank-Workloads nach einem Neustart.

Während des Datenbankstarts werden viele interne Speicherkomponenten initialisiert. Die größte dieser Komponenten ist der InnoDB-Pufferpool, der in Aurora MySQL standardmäßig 75 % der Instance-Speichergröße ausmacht. Unsere Tests haben ergeben, dass die Initialisierungszeit proportional zur Größe des InnoDB-Pufferpools ist und daher mit der Größe der DB-Instance-Klasse skaliert. Während dieser Initialisierungsphase kann die Datenbank keine Verbindungen akzeptieren, wodurch es zu längeren Ausfallzeiten bei Neustarts kommt. In der ersten Phase des schnellen Neustarts von Aurora MySQL wird die Initialisierung des Pufferpools optimiert, wodurch die Zeit für die Datenbankinitialisierung und damit die Gesamtneustartzeit reduziert werden kann.

Weitere Informationen finden Sie im Blog Reduce downtime with Amazon Aurora MySQL database restart time optimizations.

Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3

Aurora MySQL Version 3 behandelt temporäre Tabellen anders als frühere Aurora-MySQL-Versionen. Dieses neue Verhalten wird von der Community Edition von MySQL 8.0 geerbt. Es gibt zwei Arten von temporären Tabellen, die mit Aurora MySQL Version 3 erstellt werden können:

- Interne (oder implizite) temporäre Tabellen – werden von der Aurora-MySQL-Engine erstellt, um Vorgänge wie Sortieraggregation, abgeleitete Tabellen oder allgemeine Tabellenausdrücke (CTEs) zu behandeln.
- Vom Benutzer erstellte (oder explizite) temporäre Tabellen – werden von der Aurora-MySQL-Engine erstellt, wenn Sie die Anweisung `CREATE TEMPORARY TABLE` verwenden.

Es gibt zusätzliche Überlegungen sowohl für interne als auch für vom Benutzer erstellte temporäre Tabellen auf Aurora-Reader-DB-Instances. Diese werden in den folgenden Abschnitten erläutert.

Themen

- Speicher-Engine für interne (implizite) temporäre Tabellen
- Begrenzung der Größe interner temporärer Tabellen im Arbeitsspeicher
- Abschwächung von Füllungsgradproblemen bei internen temporären Tabellen auf Aurora Replicas
- Optimieren des temptable_max_mmap-Parameters auf Aurora MySQL-DB-Instances

- Vom Benutzer erstellte (explizite) temporäre Tabellen auf Reader-DB-Instances
- Fehler bei der Erstellung temporärer Tabellen und Abhilfemaßnahmen

Speicher-Engine für interne (implizite) temporäre Tabellen

Beim Generieren von Zwischenergebnismengen versucht Aurora MySQL zunächst, in temporäre Tabellen im Arbeitsspeicher zu schreiben. Dies ist aufgrund inkompatibler Datentypen oder konfigurierter Grenzwerte möglicherweise nicht erfolgreich. In diesem Fall wird die temporäre Tabelle in eine temporäre Tabelle auf dem Datenträger konvertiert, anstatt im Arbeitsspeicher abgelegt zu werden. Weitere Informationen hierzu finden Sie unter Verwendung interner temporärer Tabellen in MySQL in der MySQL-Dokumentation.

In Aurora-MySQL-Version 3 unterscheidet sich die Funktionsweise interner temporärer Tabellen von früheren Aurora-MySQL-Versionen. Anstatt sich bei solchen temporären Tabellen zwischen den InnoDB- und MyISAM-Speicher-Engines zu entscheiden, wählen Sie jetzt zwischen den `TempTable`- und `MEMORY`-Speicher-Engines.

Mit der `TempTable`-Speicher-Engine können Sie eine zusätzliche Auswahl für den Umgang mit bestimmten Daten treffen. Die betroffenen Daten überlaufen den Speicherpool, der alle internen temporären Tabellen für die DB-Instance enthält.

Diese Optionen können die Leistung von Abfragen beeinflussen, die hohe Mengen an temporären Daten generieren, z. B. während der Durchführung von Aggregationen wie `GROUP BY` auf großen Tabellen.

> **Tip**
>
> Wenn Ihre Workload Abfragen enthält, die interne temporäre Tabellen generieren, bestätigen Sie, wie Ihre Anwendung mit dieser Änderung funktioniert, indem Sie Benchmarks ausführen und leistungsbezogene Metriken überwachen.
> In einigen Fällen passt die Menge an temporären Daten in`TempTable`Speicherpool oder überläuft den Speicherpool nur um einen kleinen Betrag. In diesen Fällen empfehlen wir die `TempTable`-Einstellung für interne temporäre Tabellen und speicherzugeordnete Dateien, um Überlaufdaten zu speichern. Dies ist die Standardeinstellung.

Die `TempTable`-Speicher-Engine ist die Standardeinstellung. `TempTable` verwendet einen gemeinsamen Speicherpool für alle temporären Tabellen, die diese Engine verwenden, anstelle eines

maximalen Speicherlimits pro Tabelle. Die Größe dieses Speicherpools wird durch den Parameter temptable_max_ram angegeben. Der Wert dieses Parameters beträgt standardmäßig 1 GiB bei DB-Instances mit 16 oder mehr GiB Arbeitsspeicher und 16 MB bei DB-Instances mit weniger als 16 GB Arbeitsspeicher. Die Größe des Speicherpools beeinflusst den Speicherverbrauch auf Sitzungsebene.

Bei Verwendung der Speicher-Engine `TempTable` kann es vorkommen, dass die temporären Daten die Größe des Speicherpools überschreiten. In diesem Fall speichert Aurora MySQL die Überlaufdaten mithilfe eines sekundären Mechanismus.

Sie können den Parameter temptable_max_mmap festlegen, um anzugeben, ob die Daten zu temporären Dateien im Arbeitsspeicher oder zu internen temporären Tabellen von InnoDB auf der Festplatte überlaufen werden. Die verschiedenen Datenformate und Überlaufkriterien dieser Überlaufmechanismen können sich auf die Abfrageleistung auswirken. Sie tun dies, indem sie die Menge der auf die Festplatte geschriebenen Daten und die Nachfrage nach Festplattenspeicherdurchsatz beeinflussen.

Version 3 von Aurora MySQL speichert die Überlaufdaten auf folgende Weise:

- Bei der Writer-DB-Instance befinden sich die Dateien, die zu internen temporären Tabellen von InnoDB oder zu speicherzugeordneten Dateien überlaufen, im lokalen Speicher der Instance.
- Bei Reader-DB-Instances befinden sich die Überlaufdaten immer in speicherzugeordneten temporären Dateien im lokalen Speicher.

  Schreibgeschützte Instances können keine Daten auf dem Aurora-Cluster-Volume speichern.

Die Konfigurationsparameter für interne temporäre Tabellen gelten unterschiedlich für die Writer- und Reader-Instancees in Ihrem Cluster.

- Für Reader-Instances verwendet Aurora MySQL immer die Speicher–Engine `TempTable`.
- Die Größe für `temptable_max_mmap` beträgt für Writer- und Reader-Instances unabhängig von der Speichergröße der DB-Instance standardmäßig 1 GB. Sie können diesen Wert sowohl für Writer- als auch für Reader-Instances anpassen.
- Die Einstellung von `temptable_max_mmap` auf `0` deaktiviert die Verwendung von temporären Dateien mit Speicherzuordnung auf Writer-Instances.
- Sie können `0` auf Reader-Instances nicht auf `temptable_max_mmap` setzen.

> **Note**
>
> Es wird nicht empfohlen, den temptable_use_mmap-Parameter zu verwenden. Dieser ist veraltet, und es wird erwartet, dass die Unterstützung dafür in einer künftigen MySQL-Version entfernt wird.

Begrenzung der Größe interner temporärer Tabellen im Arbeitsspeicher

Wie in Speicher-Engine für interne (implizite) temporäre Tabellen erwähnt, können Sie temporäre Tabellenressourcen global steuern, indem Sie die Einstellungen temptable_max_ram und temptable_max_mmap verwenden.

Sie können die Größe jeder einzelnen internen temporären Tabelle im Arbeitsspeicher auch einschränken, indem Sie den DB-Parameter tmp_table_size verwendenParameter. Dieses Limit soll verhindern, dass einzelne Abfragen übermäßig viele globale temporäre Tabellenressourcen verbrauchen, was sich auf die Leistung gleichzeitiger Abfragen auswirken kann, die diese Ressourcen benötigen.

Der Parameter `tmp_table_size` definiert die maximale Größe temporärer Tabellen, die von der `MEMORY`-Speicher-Engine in Aurora-MySQL-Version 3 erstellt werden.

In Aurora-MySQL-Version 3.04 und höher definiert der Parameter `tmp_table_size` auch die maximale Größe temporärer Tabellen, die von der `TempTable`-Speicher-Engine erstellt werden, wenn der DB-Parameter `aurora_tmptable_enable_per_table_limit` auf `ON` festgelegt ist. Dieses Verhalten ist standardmäßig deaktiviert (`OFF`). Dies entspricht dem Verhalten von Aurora-MySQL-Version 3.03 und niedrigeren Versionen.

- Wenn `aurora_tmptable_enable_per_table_limit` `OFF` ist ,wird `tmp_table_size` nicht für interne temporäre Tabellen im Arbeitsspeicher berücksichtigt, die von der`TempTable`-Speicher-Engine erstellt wurden.

  Das globale `TempTable`-Ressourcenlimit gilt jedoch weiterhin. Aurora MySQL verhält sich wie folgt, wenn das globale `TempTable`-Ressourcenlimit erreicht wird:

  - Writer-DB-Instances – Aurora MySQL konvertiert die temporäre Tabelle im Arbeitsspeicher automatisch in eine temporäre InnoDB-Tabelle auf der Festplatte.
  - Reader-DB-Instances – Die Abfrage endet mit einem Fehler.

```
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
```

- Wenn `aurora_tmptable_enable_per_table_limit` `ON` ist, verhält sich Aurora MySQL wie folgt, wenn das `tmp_table_size`-Limit erreicht wird:
  - Writer-DB-Instances – Aurora MySQL konvertiert die temporäre Tabelle im Arbeitsspeicher automatisch in eine temporäre InnoDB-Tabelle auf der Festplatte.
  - Reader-DB-Instances – Die Abfrage endet mit einem Fehler.

    ```
    ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
    ```

    In diesem Fall gelten sowohl das globale `TempTable`-Ressourcenlimit als auch das Limit pro Tabelle.

> **Note**
>
> Der `aurora_tmptable_enable_per_table_limit`-Parameter hat keine Auswirkungen, wenn interne_tmp_mem_storage_engine auf `MEMORY` eingestellt ist. In diesem Fall wird die maximale Größe einer temporären Tabelle im Arbeitsspeicher durch den Wert tmp_table_size oder max_heap_table_size definiert, je nachdem, welcher Wert kleiner ist.

Die folgenden Beispiele zeigen das Verhalten des `aurora_tmptable_enable_per_table_limit`-Parameters für Writer- und Reader-DB-Instances.

Example einer Writer-DB-Instance mit der Einstellung **`aurora_tmptable_enable_per_table_limit`** auf **`OFF`**

Die temporäre Tabelle im Arbeitsspeicher wird nicht in eine temporäre InnoDB-Tabelle auf dem Datenträger konvertiert.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
```

```
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
|                  0 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (13.99 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example einer Writer-DB-Instance mit der Einstellung **aurora_tmptable_enable_per_table_limit** auf **ON**

Die temporäre Tabelle im Arbeitsspeicher wird in eine temporäre InnoDB-Tabelle auf dem Datenträger konvertiert.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+-----------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+-----------------+-----------------------------------------
+------------------+
|                  0 | 3.04.0          |                                        1 |
      16777216 |
+--------------------+-----------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
+---------+
| max(n)  |
+---------+
| 6000000 |
+---------+
1 row in set (4.10 sec)
```

```
mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example einer Reader-DB-Instance mit der Einstellung **aurora_tmptable_enable_per_table_limit** auf **OFF**

Die Abfrage wird ohne Fehler beendet, weil `tmp_table_size` nicht zutrifft, und das globale `TempTable`-Ressourcenlimit wurde nicht erreicht.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0          |                                        0 |
       1073741824 |           1073741824 |
+--------------------+-----------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (14.05 sec)
```

Example einer Reader-DB-Instance mit der Einstellung **`aurora_tmptable_enable_per_table_limit`** auf **`OFF`**

Diese Abfrage erreicht das globale TempTable-Ressourcenlimit, wobei `aurora_tmptable_enable_per_table_limit` auf OFF eingestellt ist. Die Abfrage endet mit einem Fehler auf den Reader-Instances.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.01 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 120000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_1586_2' is full
```

Example einer Reader-DB-Instance mit der Einstellung **`aurora_tmptable_enable_per_table_limit`** auf **`ON`**

Die Abfrage endet mit einem Fehler, wenn das `tmp_table_size`-Limit erreicht wird.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+----------------------------------------
+------------------+
```

```
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+------------------------------------------
+------------------+
|                  1 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+------------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_8_2' is full
```

Abschwächung von Füllungsgradproblemen bei internen temporären Tabellen auf Aurora Replicas

Sie können Probleme mit der Größenbeschränkung für temporäre Tabellen vermeiden, indem Sie die Parameter `temptable_max_ram` und `temptable_max_mmap` auf einen kombinierten Wert festlegen, der die Anforderungen Ihrer Workloads erfüllen kann.

Seien Sie vorsichtig, wenn Sie den Wert des Parameters `temptable_max_ram` festlegen. Wenn Sie den Wert zu hoch einstellen, wird der verfügbare Arbeitsspeicher auf der Datenbank-Instance reduziert, was zu einer Out-of-Memory-Bedingung führen kann. Überwachen Sie den durchschnittlichen möglichen freien Arbeitsspeicher der DB-Instance. Ermitteln Sie dann einen geeigneten Wert für `temptable_max_ram`, damit Sie immer noch angemessenen freien Speicherplatz auf der Instance zur Verfügung haben. Weitere Informationen finden Sie unter Probleme mit freisetzbarem Speicher in Amazon Aurora.

Es ist auch wichtig, die Größe des lokalen Speichers und die Speicherplatzbelegung durch die temporäre Tabelle zu überwachen. Sie können den für eine bestimmte DB-Instance verfügbaren temporären Speicher mit der Amazon-CloudWatch-Metrik `FreeLocalStorage` überwachen, die unter CloudWatch Amazon-Metriken für Amazon Aurora beschrieben wird.

> **Note**
>
> Dieses Verfahren funktioniert nicht, wenn der `aurora_tmptable_enable_per_table_limit`-Parameter auf `ON` festgelegt ist.

Weitere Informationen finden Sie unter Begrenzung der Größe interner temporärer Tabellen im Arbeitsspeicher.

Example 1

Sie wissen, dass Ihre temporären Tabellen auf eine kumulative Größe von 20 GiB anwachsen. Sie möchten temporäre In-Memory-Tabellen auf 2 GiB festlegen und auf maximal 20 GiB auf der Festplatte anwachsen lassen.

Setzen Sie `temptable_max_ram` auf **`2,147,483,648`** und `temptable_max_mmap` auf **`21,474,836,480`**. Diese Werte werden in Byte angegeben.

Diese Parametereinstellungen stellen sicher, dass Ihre temporären Tabellen auf eine kumulative Summe von 22 GiB anwachsen können.

Example 2

Ihre aktuelle Instance-Größe ist 16xlarge oder größer. Sie kennen nicht die Gesamtgröße der temporären Tabellen, die Sie möglicherweise benötigen. Sie möchten bis zu 4 GiB im Arbeitsspeicher und den maximal verfügbaren Speicher auf der Festplatte nutzen können.

Setzen Sie `temptable_max_ram` auf **`4,294,967,296`** und `temptable_max_mmap` auf **`1,099,511,627,776`**. Diese Werte werden in Byte angegeben.

Hier legen Sie `temptable_max_mmap` auf 1 TiB fest, was weniger als der maximale lokale Speicher von 1,2 TiB auf einer Aurora-DB-Instance der Größe 16xlarge ist.

Passen Sie bei einer kleineren Instance-Größe den Wert `temptable_max_mmap` an, damit der verfügbare lokale Arbeitsspeicher nicht vollständig belegt wird. Bei einer 2xlarge-Instance stehen für den lokalen Speicher beispielsweise nur 160 GiB zur Verfügung. Daher empfehlen wir, den Wert auf weniger als 160 GiB einzustellen. Weitere Informationen über den verfügbaren lokalen Speicher für DB-Instance-Größen finden Sie unter Temporäre Speicherlimits für Aurora MySQL.

Optimieren des temptable_max_mmap-Parameters auf Aurora MySQL-DB-Instances

Der `temptable_max_mmap`-Parameter in Aurora MySQL steuert die maximale Menge an lokalem Festplattenspeicher, die von speicherzugeordneten Dateien genutzt werden kann, bevor sie in die temporären InnoDB-Tabellen auf der Festplatte überlaufen (auf Writer-DB-Instances) oder einen Fehler verursachen (auf Reader-DB-Instances). Wenn Sie diesen DB-Instance-Parameter richtig einstellen, kann dies dazu beitragen, die Leistung Ihrer DB-Instances zu optimieren.

Voraussetzungen

1. Stellen Sie sicher, dass das Leistungsschema aktiviert ist. Sie können dies durch Ausführung des folgenden SQL-Befehls überprüfen:

   ```
   SELECT @@performance_schema;
   ```

   Ein Ausgabewert von 1 gibt an, dass es aktiviert ist.

2. Vergewissern Sie sich, dass die Instrumentierung für den Speicher der temporären Tabellen aktiviert ist. Sie können dies durch Ausführung des folgenden SQL-Befehls überprüfen:

   ```
   SELECT name, enabled FROM performance_schema.setup_instruments WHERE name LIKE
    '%memory%temptable%';
   ```

   In der `enabled`-Spalte wird für die relevanten Einträge der Instrumentierung für den temporären Tabellenspeicher `YES` angezeigt.

Überwachen der Verwendung temporärer Tabellen

Wenn Sie den Anfangswert für `temptable_max_mmap` festlegen, sollten Sie mit 80 % der lokalen Speichergröße für die DB-Instance-Klasse beginnen, die Sie verwenden. Dadurch wird sichergestellt, dass die temporären Tabellen über genügend Speicherplatz für einen effizienten Betrieb verfügen und Platz für andere Festplattennutzungen auf der Instance verbleibt.

Informationen über die lokale Speichergröße für Ihre DB-Instance-Klasse finden Sie unter Temporäre Speicherlimits für Aurora MySQL.

Wenn Sie beispielsweise die DB-Instance-Klasse db.r5.large verwenden, beträgt die lokale Speichergröße 32 GiB. In diesem Fall würden Sie den `temptable_max_mmap`-Parameter zunächst auf 80 % von 32 GiB setzen, was 25,6 GiB entspricht.

Nachdem Sie den Anfangswert von `temptable_max_mmap` festgelegt haben, führen Sie Ihren Spitzen-Workload auf den Aurora MySQL-Instances aus. Überwachen Sie die aktuelle und hohe Festplattennutzung durch die temporäre Tabelle mithilfe der folgenden SQL-Abfrage:

```
SELECT event_name, current_count, current_alloc, current_avg_alloc, high_count,
 high_alloc, high_avg_alloc
FROM sys.memory_global_by_current_bytes WHERE event_name LIKE 'memory/temptable/%';
```

Diese Abfrage ruft die folgenden Informationen ab:

- `event_name` – Der Name des Ereignisses für den temporären Tabellenspeicher oder die Festplattennutzung
- `current_count` – Die aktuelle Anzahl der zugewiesenen Blöcke für den temporären Tabellenspeicher oder die Festplatte
- `current_alloc` – Die aktuelle Menge an Speicher oder Festplatte, die temporären Tabellen zugewiesen ist
- `current_avg_alloc` – Die aktuelle durchschnittliche Größe des temporären Tabellenspeichers oder der Festplattenblöcke
- `high_count` – Die höchste Anzahl des zugewiesenen temporären Tabellenspeichers oder der Festplattenblöcke
- `high_alloc` – Die größte Menge an Speicher oder Festplatte, die temporären Tabellen zugewiesen ist
- `high_avg_alloc` – Die höchste durchschnittliche Größe des temporären Tabellenspeichers oder der Festplattenblöcke

Wenn Ihre Abfragen bei Verwendung dieser Einstellung mit dem Fehler „Table is full“ fehlschlagen, bedeutet dies, dass Ihr Workload mehr Festplattenspeicher für temporäre Tabellenoperationen benötigt. In diesem Fall sollten Sie erwägen, die Größe Ihrer DB-Instance auf eine mit mehr lokalem Speicherplatz zu erhöhen.

Den optimalen **`temptable_max_mmap`**-Wert festlegen

Gehen Sie wie folgt vor, um die richtige Größe für den `temptable_max_mmap`-Parameter festzulegen und ihn zu überwachen:

1. Überprüfen Sie die Ausgabe der vorherigen Abfrage und ermitteln Sie die maximale Festplattennutzung durch temporäre Dateien wie in der `high_alloc`-Spalte angegeben.
2. Passen Sie, basierend auf der maximalen Festplattennutzung durch temporäre Dateien, den `temptable_max_mmap`-Parameter in der DB-Parametergruppe für Ihre DB-Instances von Aurora MySQL an.

   Stellen Sie den Wert so ein, dass er etwas höher ist als die maximale Festplattennutzung durch temporäre Tabellen, um eine zukünftige Zunahme zu berücksichtigen.
3. Wenden Sie die Änderungen an der Parametergruppe auf Ihre DB-Instances an.
4. Überwachen Sie die Festplattennutzung der temporären Tabelle während des Spitzen-Workloads erneut, um sicherzustellen, dass der neue `temptable_max_mmap`-Wert angemessen ist.

5. Wiederholen Sie die vorherigen Schritte bei Bedarf, um den `temptable_max_mmap`-Parameter zu optimieren.

Vom Benutzer erstellte (explizite) temporäre Tabellen auf Reader-DB-Instances

Sie können explizite temporäre Tabellen erstellen, indem Sie das Schlüsselwort `TEMPORARY` in Ihrer Anweisung `CREATE TABLE` verwenden. Explizite temporäre Tabellen werden auf der Writer-DB-Instance in einem Aurora-DB-Cluster unterstützt. Sie können auch explizite temporäre Tabellen für Reader DB-Instances verwenden. Die Tabellen können jedoch die Verwendung der InnoDB-Speicher-Engine nicht erzwingen.

Um Fehler beim Erstellen expliziter temporärer Tabellen auf Reader-DB-Instances von Aurora MySQL zu vermeiden, stellen Sie sicher, dass alle `CREATE TEMPORARY TABLE`-Anweisungen mit einer der folgenden Methoden ausgeführt werden:

- Verzichten Sie auf die Angabe der `ENGINE=InnoDB`-Klausel.
- Legen Sie den SQL-Modus nicht auf `NO_ENGINE_SUBSTITUTION` fest.

Fehler bei der Erstellung temporärer Tabellen und Abhilfemaßnahmen

Der Fehler, den Sie erhalten, ist unterschiedlich, je nachdem, ob Sie ein einfache `CREATE TEMPORARY TABLE`-Aussage oder die Variation `CREATE TEMPORARY TABLE AS SELECT` verwenden. Das folgende Beispiel zeigt die Reihenfolge der verschiedenen Typen:

Dieses temporäre Tabellenverhalten gilt nur für schreibgeschützte Instances. Dieses erste Beispiel bestätigt, dass es sich um die Art von Instance handelt, mit der die Sitzung verbunden ist.

```
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

Für einfache `CREATE TEMPORARY TABLE`-Anweisungen schlägt die Anweisung fehl, wenn der `NO_ENGINE_SUBSTITUTION`-SQL-Modus aktiviert ist. Wenn `NO_ENGINE_SUBSTITUTION` deaktiviert wird (Standardeinstellung), wird die entsprechende Engine-Ersetzung vorgenommen und die temporäre Tabelle wird erfolgreich erstellt.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql>  CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
*************************** 1. row ***************************
       Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Für `CREATE TEMPORARY TABLE AS SELECT`-Anweisungen schlägt die Anweisung fehl, wenn der SQL-Modus `NO_ENGINE_SUBSTITUTION` aktiviert ist. Wenn `NO_ENGINE_SUBSTITUTION` deaktiviert wird (Standardeinstellung), wird die entsprechende Engine-Ersetzung vorgenommen und die temporäre Tabelle wird erfolgreich erstellt.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> show create table tt3;
+-------+-----------------------------------------------------------+
| Table | Create Table                                              |
+-------+-----------------------------------------------------------+
| tt3   | CREATE TEMPORARY TABLE `tt3` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

Weitere Informationen zu den Speicheraspekten und Leistungsauswirkungen temporärer Tabellen in Aurora-MySQL-Version 3 finden Sie im Blogbeitrag Verwenden der TempTable-Speicher-Engine in Amazon RDS für MySQL und Amazon Aurora MySQL.

Vergleich von Aurora-MySQL-Version 2 und Aurora-MySQL-Version 3

Verwenden Sie Folgendes, um mehr über Änderungen zu erfahren, die Sie beachten sollten, wenn Sie Ihren Aurora-MySQL-Version 2-Cluster auf Version 3 aktualisieren.

Themen

- Unterstützung der Atomic Data Definition Language (DDL)
- Funktionsunterschiede zwischen Aurora MySQL Version 2 und 3
- Unterstützung für Instance-Klassen
- Parameteränderungen für Aurora MySQL Version 3
- Statusvariablen.
- Inklusive Sprachänderungen für Aurora MySQL Version 3
- AUTO_INCREMENT-Werte
- Binäre Protokoll-Replikation

Unterstützung der Atomic Data Definition Language (DDL)

Eine der größten Änderungen von MySQL 5.7 auf 8.0 ist die Einführung des Atomic Data Dictionary. Vor MySQL 8.0 verwendete das MySQL-Datenwörterbuch einen dateibasierten Ansatz, um Metadaten wie Tabellendefinitionen (.frm), Trigger (.trg) und Funktionen getrennt von den Metadaten der Speicher-Engine (wie denen von InnoDB) zu speichern. Dabei gab es einige Probleme, darunter das Risiko, dass Tabellen „verwaist“ wurden, wenn während einer DDL-Operation etwas Unerwartetes passierte, wodurch die dateibasierten und die Speicher-Engine-Metadaten nicht mehr synchron waren.

Zum Beheben dieses Problems wurde mit MySQL 8.0 das Atomic Data Dictionary eingeführt, das alle Metadaten in einer Reihe interner InnoDB-Tabellen im `mysql`-Schema speichert. Diese neue Architektur bietet eine transaktionale, ACID-konforme Methode zur Verwaltung von Datenbankmetadaten und löst damit das „atomare DDL“-Problem des alten dateibasierten Ansatzes. Weitere Informationen zum Atomic Data Dictionary finden Sie unter Entfernung des dateibasierten Metadatenspeichers und Unterstützung von Atomic-Datendefinitionsanweisungen im MySQL-Referenzhandbuch.

Aufgrund dieser Architekturänderung müssen Sie beim Upgrade von Aurora-MySQL-Version 2 auf Version 3 Folgendes beachten:

- Die dateibasierten Metadaten von Version 2 müssen während des Upgrade-Prozesses auf Version 3 zu den neuen Datenwörterbuchtabellen migriert werden. Je nachdem, wie viele Datenbankobjekte migriert werden, kann dies einige Zeit dauern.
- Durch die Änderungen wurden auch einige neue Inkompatibilitäten eingeführt, die möglicherweise behoben werden müssen, bevor Sie ein Upgrade von MySQL 5.7 auf 8.0 durchführen können. Zum Beispiel enthält 8.0 einige neue reservierte Schlüsselwörter, die zu Konflikten mit bestehenden Datenbankobjektnamen führen könnten.

Um Ihnen zu helfen, diese Inkompatibilitäten vor dem Upgrade der Engine zu identifizieren, führt Aurora MySQL vor der Durchführung des Datenwörterbuch-Upgrades eine Reihe von Upgrade-Kompatibilitätsprüfungen (Vorabprüfungen) durch, um festzustellen, ob Ihr Datenbankwörterbuch inkompatible Objekte enthält. Weitere Informationen zu den Vorabprüfungen finden Sie unter Vorabprüfungen für Hauptversions-Upgrades von Aurora MySQL.

Funktionsunterschiede zwischen Aurora MySQL Version 2 und 3

Die folgenden Amazon-Aurora-MySQL-Funktionen werden in Aurora MySQL for MySQL 5.7 unterstützt, in Aurora MySQL for MySQL 8.0 derzeit jedoch nicht.

- Sie können Aurora-MySQL-Version 3 nicht für Aurora Serverless v1-Cluster verwenden. Aurora-MySQL-Version 3 arbeitet mit Aurora Serverless v2.
- Der Labor-Modus gilt nicht für Aurora-MySQL-Version 3. In Aurora-MySQL-Version 3 gibt es keine Labormodusfunktionen. Instant DDL ersetzt die schnelle Online-DDL-Funktion, die früher im Labormodus verfügbar war. Ein Beispiel finden Sie unter Sofortige DDL (Aurora MySQL Version 3).
- Der Abfrage-Cache wird aus der Community MySQL 8.0 und auch aus Aurora MySQL Version 3 entfernt.
- Aurora MySQL Version 3 ist mit der MySQL-Hash-Join-Funktion der Community kompatibel. Die Aurora-spezifische Implementierung von Hash-Joins in Aurora MySQL Version 2 wird nicht verwendet. Informationen zur Verwendung von Hash-Joins mit Aurora-Parallelabfrage finden Sie unterHash-Join für parallele Abfrage-Cluster aktivierenundAurora-MySQL-Hinweiseaus. Allgemeine Nutzungsinformationen zu Hash-Joins finden Sie unterHash Join-OptimierungimMySQL-Referenzhandbuchaus.
- Diemysql.lambda_asyncDie gespeicherte Prozedur, die in Aurora MySQL Version 2 veraltet war, wird in Version 3 entfernt. Verwenden Sie für Version 3 die asynchrone Funktionlambda_asyncStattdessen.

- Der Standardzeichensatz in Aurora MySQL Version 3 ist`utf8mb4`aus. In Aurora MySQL Version 2 war der Standardzeichensatz`latin1`aus. Weitere Informationen zu diesem Zeichensatz finden Sie unterDer utf8mb4-Zeichensatz (UTF-8-Unicode-Kodierung mit 4 Bytes)imMySQL-Referenzhandbuchaus.

Einige Aurora MySQL-Funktionen stehen für bestimmte Kombinationen vonAWSVersion der Region und der DB-Engine Details hierzu finden Sie unter Unterstützte Funktionen in Amazon Aurora by AWS-Region und der Aurora-DB-Engine.

Unterstützung für Instance-Klassen

Aurora-MySQL-Version 3 unterstützt einen anderen Satz von Instance-Klassen als Aurora-MySQL-Version 2:

- Für größere Instances können Sie die modernen Instance-Klassen wie`db.r5`, `db.r6g` und `db.x2g` verwenden.
- Für kleinere Instances können Sie die modernen Instance-Klassen wie `db.t3` und `db.t4g` verwenden.

> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter Verwendung von T-Instance-Klassen für Entwicklung und Tests.

Die folgenden Instance-Klassen aus Aurora-MySQL-Version 2 sind für Aurora-MySQL-Version 3 nicht verfügbar:

- `db.r4`
- `db.r3`
- `db.t3.small`
- `db.t2`

Überprüfen Sie Ihre Verwaltungsskripte auf CLI-Anweisungen, die DB-Instances von Aurora MySQL erstellen. Hardcode-Instance-Klassennamen, die für Aurora-MySQL-Version 3 nicht verfügbar sind. Ändern Sie ggf. die Namen der Instance-Klasse zu solchen, die Aurora-MySQL-Version 3 unterstützt.

> **Tip**
>
> Um die Instance-Klassen zu überprüfen, die Sie für eine bestimmte Kombination von Aurora MySQL-Version und AWS-Region verwenden können, verwenden Sie den `describe-orderable-db-instance-options` AWS CLI-Befehl.

Ausführliche Einzelheiten zu Aurora-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Parameteränderungen für Aurora MySQL Version 3

Aurora MySQL Version 3 enthält neue Konfigurationsparameter auf Cluster-Ebene und Instance-Ebene. Aurora MySQL Version 3 entfernt auch einige Parameter, die in Aurora MySQL Version 2 vorhanden waren. Einige Parameternamen werden infolge der Initiative zur inklusiven Sprache geändert. Aus Gründen der Abwärtskompatibilität können Sie die Parameterwerte weiterhin entweder mit den alten Namen oder den neuen Namen abrufen. Sie müssen jedoch die neuen Namen verwenden, um Parameterwerte in einer benutzerdefinierten Parametergruppe anzugeben.

In Aurora MySQL Version 3 ist der Wert des`lower_case_table_names`-Parameter wird zum Zeitpunkt der Erstellung des Clusters dauerhaft festgelegt. Wenn Sie für diese Option einen nicht standardmäßigen Wert verwenden, richten Sie Ihre benutzerdefinierte Parametergruppe Aurora MySQL Version 3 vor dem Upgrade ein. Geben Sie dann die Parametergruppe während des Wiederherstellungsvorgangs „Cluster wiederherstellen“ oder „Snapshot wiederherstellen“ an.

> **Note**
>
> Bei einer auf Aurora MySQL basierenden globalen Aurora-Datenbank können Sie kein direktes Upgrade von Aurora-MySQL-Version 2 zu Version 3 durchführen, wenn der Parameter `lower_case_table_names` aktiviert ist. Verwenden Sie stattdessen die Snapshot-Wiederherstellungsmethode.

In Aurora-MySQL-Version 3 gelten die Parameter `init_connect` und `read_only` nicht für Benutzer, die über die `CONNECTION_ADMIN`-Berechtigung verfügen. Dies schließt den Aurora-Hauptbenutzer ein. Weitere Informationen finden Sie unter Rollenbasiertes Berechtigungsmodell.

Eine Liste aller Aurora-MySQL-Clusterparameter finden Sie unter Parameter auf Cluster-Ebene. Die Tabelle umfasst alle Parameter aus Aurora-MySQL-Version 2 und 3. Die Tabelle enthält Hinweise

dazu, welche Parameter in Aurora-MySQL-Version 3 neu sind oder aus Aurora-MySQL-Version 3 entfernt wurden.

Eine vollständige Liste der Aurora-MySQL-Instance-Parameter finden Sie unter Parameter auf Instance-Ebene. Die Tabelle umfasst alle Parameter aus Aurora-MySQL-Version 2 und 3. Die Tabelle enthält Hinweise dazu, welche Parameter in Aurora-MySQL-Version 3 neu sind und welche Parameter aus Aurora-MySQL-Version 3 entfernt wurden. Sie enthält auch Hinweise dazu, welche Parameter in früheren Versionen modifizierbar waren, aber nicht Aurora-MySQL´-Version 3.

Weitere Informationen zu geänderten Parameternamen finden Sie unter Inklusive Sprachänderungen für Aurora MySQL Version 3.

Statusvariablen.

Informationen zu Statusvariablen, die nicht auf Aurora MySQL anwendbar sind, finden Sie unterMySQL-Statusvariablen, die nicht für Aurora MySQL geltenaus.

Inklusive Sprachänderungen für Aurora MySQL Version 3

Aurora MySQL Version 3 ist mit Version 8.0.23 aus der MySQL Community Edition kompatibel. Aurora MySQL Version 3 enthält auch Änderungen von MySQL 8.0.26 in Bezug auf Schlüsselwörter und Systemschemas für inklusive Sprache. Zum Beispiel, das`SHOW REPLICA STATUS`Befehl wird jetzt bevorzugt statt`SHOW SLAVE STATUS`aus.

Die folgenden Amazon CloudWatch-Metriken haben neue Namen in Aurora MySQL Version 3.

In Aurora MySQL Version 3 sind nur die neuen Metriknamen verfügbar. Stellen Sie sicher, dass Sie alle Alarme oder andere Automatisierungen aktualisieren, die auf Metriknamen beruhen, wenn Sie auf Aurora MySQL Version 3 aktualisieren.

Alter Name	Neuer Name
`ForwardingMasterDMLLatency`	`ForwardingWriterDMLLatency`
`ForwardingMasterOpenSessions`	`ForwardingWriterOpenSessions`
`AuroraDMLRejectedMasterFull`	`AuroraDMLRejectedWriterFull`

Alter Name	Neuer Name
`ForwardingMasterDMLThroughput`	`ForwardingWriterDMLThroughput`

Die folgenden Statusvariablen haben neue Namen in Aurora MySQL Version 3.

Aus Gründen der Kompatibilität können Sie beide Namen in der ersten Version 3 von Aurora MySQL verwenden. Die alten Statusvariablennamen sollen in einer zukünftigen Version entfernt werden.

Name, der entfernt werden soll	Neuer oder bevorzugter Name
`Aurora_fwd_master_dml_stmt_duration`	`Aurora_fwd_writer_dml_stmt_duration`
`Aurora_fwd_master_dml_stmt_count`	`Aurora_fwd_writer_dml_stmt_count`
`Aurora_fwd_master_select_stmt_duration`	`Aurora_fwd_writer_select_stmt_duration`
`Aurora_fwd_master_select_stmt_count`	`Aurora_fwd_writer_select_stmt_count`
`Aurora_fwd_master_errors_session_timeout`	`Aurora_fwd_writer_errors_session_timeout`
`Aurora_fwd_master_open_sessions`	`Aurora_fwd_writer_open_sessions`
`Aurora_fwd_master_errors_session_limit`	`Aurora_fwd_writer_errors_session_limit`
`Aurora_fwd_master_errors_rpc_timeout`	`Aurora_fwd_writer_errors_rpc_timeout`

Die folgenden Konfigurationsparameter haben neue Namen in Aurora MySQL Version 3.

Aus Gründen der Kompatibilität können Sie die Parameterwerte im `mysql`-Client unter Verwendung eines der beiden Namen in der ersten Version 3 von Aurora MySQL prüfen. Beim Ändern der Werte in einer benutzerdefinierten Parametergruppe können Sie nur die neuen Namen verwenden. Die alten Parameternamen werden in einer zukünftigen Version entfernt.

Name, der entfernt werden soll	Neuer oder bevorzugter Name
`aurora_fwd_master_idle_timeout`	`aurora_fwd_writer_idle_timeout`
`aurora_fwd_master_max_connections_pct`	`aurora_fwd_writer_max_connections_pct`
`master_verify_checksum`	`source_verify_checksum`
`sync_master_info`	`sync_source_info`
`init_slave`	`init_replica`
`rpl_stop_slave_timeout`	`rpl_stop_replica_timeout`
`log_slow_slave_statements`	`log_slow_replica_statements`
`slave_max_allowed_packet`	`replica_max_allowed_packet`
`slave_compressed_protocol`	`replica_compressed_protocol`
`slave_exec_mode`	`replica_exec_mode`
`slave_type_conversions`	`replica_type_conversions`

Name, der entfernt werden soll	Neuer oder bevorzugter Name	
slave_sql_verify_checksum	replica_sql_verify_checksum	
slave_parallel_type	replica_parallel_type	
slave_preserve_commit_order	replica_preserve_commit_order	
log_slave_updates	log_replica_updates	
slave_allow_batching	replica_allow_batching	
slave_load_tmpdir	replica_load_tmpdir	
slave_net_timeout	replica_net_timeout	
sql_slave_skip_counter	sql_replica_skip_counter	
slave_skip_errors	replica_skip_errors	
slave_checkpoint_period	replica_checkpoint_period	
slave_checkpoint_group	replica_checkpoint_group	
slave_transaction_retries	replica_transaction_retries	
slave_parallel_workers	replica_parallel_workers	
slave_pending_jobs_size_max	replica_pending_jobs_size_max	

Name, der entfernt werden soll	Neuer oder bevorzugter Name
`pseudo_slave_mode`	`pseudo_replica_mode`

Die folgenden gespeicherten Prozeduren haben neue Namen in Aurora MySQL Version 3.

Aus Gründen der Kompatibilität können Sie beide Namen in der ersten Version 3 von Aurora MySQL verwenden. Die alten Prozedurnamen sollen in einer zukünftigen Version entfernt werden.

Name, der entfernt werden soll	Neuer oder bevorzugter Name
`mysql.rds_set_master_auto_position`	`mysql.rds_set_source_auto_position`
`mysql.rds_set_external_master`	`mysql.rds_set_external_source`
`mysql.rds_set_external_master_with_auto_position`	`mysql.rds_set_external_source_with_auto_position`
`mysql.rds_reset_external_master`	`mysql.rds_reset_external_source`
`mysql.rds_next_master_log`	`mysql.rds_next_source_log`

AUTO_INCREMENT-Werte

In Aurora MySQL Version 3 bewahrt Aurora`AUTO_INCREMENT`Wert für jede Tabelle, wenn jede DB-Instance neu gestartet wird. In Aurora MySQL Version 2 wird`AUTO_INCREMENT`Der Wert wurde nach einem Neustart nicht beibehalten.

Die`AUTO_INCREMENT`-Wert wird nicht beibehalten, wenn Sie einen neuen Cluster einrichten, indem Sie aus einem Snapshot wiederherstellen, eine Point-in-Time-Wiederherstellung durchführen und einen Cluster klonen. In diesen Fällen wird die`AUTO_INCREMENT`value wird basierend auf dem größten Spaltenwert in der Tabelle zum Zeitpunkt der Erstellung des Snapshots auf den Wert

initialisiert. Dieses Verhalten ist anders als in RDS für MySQL 8.0, wo der `AUTO_INCREMENT`-Wert während dieser Vorgänge beibehalten wird.

Binäre Protokoll-Replikation

In der MySQL 8.0 Community Edition ist die Replikation des Binärprotokolls standardmäßig aktiviert. In Aurora MySQL Version 3 ist die Replikation des binären Protokolls standardmäßig deaktiviert.

> **Tip**
>
> Wenn Ihre Hochverfügbarkeitsanforderungen durch die integrierten Replikationsfunktionen von Aurora erfüllt werden, können Sie die Replikation von Binärprotokollen deaktiviert lassen. Auf diese Weise können Sie den Leistungsaufwand der Binärprotokollreplikation vermeiden. Sie können auch die zugehörige Überwachung und Fehlerbehebung vermeiden, die für die Verwaltung der Binärprotokollreplikation erforderlich sind.

Aurora unterstützt die Binärprotokollreplikation von einer MySQL 5.7 kompatiblen Quelle zu Aurora MySQL Version 3. Das Quellsystem kann ein Aurora MySQL-DB-Cluster, eine RDS für MySQL-DB-Instance oder eine lokale MySQL-DB-Instance sein.

Wie bei der Community MySQL unterstützt Aurora MySQL die Replikation von einer Quelle, auf der eine bestimmte Version ausgeführt wird, zu einem Ziel, auf dem dieselbe Hauptversion oder eine höhere Hauptversion ausgeführt wird. Beispielsweise wird die Replikation von einem MySQL 5.6 kompatiblen System auf Aurora MySQL Version 3 nicht unterstützt. Die Replikation von Aurora MySQL Version 3 auf ein mit MySQL 5.7 kompatibles oder MySQL 5.6—kompatibles System wird nicht unterstützt. Einzelheiten zur Verwendung der Binärprotokollreplikation finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation) aus.

Aurora MySQL Version 3 enthält Verbesserungen der Binärprotokollreplikation in der Community MySQL 8.0, z. B. gefilterte Replikation. Weitere Informationen zu den Verbesserungen der Community MySQL 8.0 finden Sie unter So bewerten Server Replikationsfilterregeln im MySQL-Referenzhandbuch aus.

Transaktionskomprimierung für die Binärprotokollreplikation

Informationen zur Verwendung zur Binärprotokollkomprimierung finden Sie unter Binärprotokoll-Transaktionskomprimierung im MySQL-Referenzhandbuch.

Die folgenden Einschränkungen gelten für die Binärprotokollkomprimierung in Aurora-MySQL-Version 3:

- Transaktionen, deren binäre Protokolldaten größer als die maximal zulässige Paketgröße sind, werden nicht komprimiert. Dies gilt unabhängig davon, ob die Einstellung für die Komprimierung des binären Protokolls in Aurora MySQL aktiviert ist. Solche Transaktionen werden repliziert, ohne komprimiert zu werden.
- Wenn Sie einen Konnektor für Change Data Capture (CDC) verwenden, der MySQL 8.0 noch nicht unterstützt, können Sie diese Funktion nicht verwenden. Es wird empfohlen, alle Konnektoren von Drittanbietern gründlich mit der Binärprotokollkomprimierung zu testen. Diese Tests sollten ausgeführt werden, bevor Sie die Binlog-Komprimierung in Systemen aktivieren, die Binlog-Replikation für CDC verwenden.

Vergleich von Aurora-MySQL-Version 3 und MySQL 8.0 Community Edition

Sie können die folgenden Informationen verwenden, um mehr über die Änderungen zu erfahren, die Sie beachten müssen, wenn Sie von einem anderen MySQL 8.0-kompatiblen System zu Aurora-MySQL-Version 3 konvertieren.

Im Allgemeinen unterstützt Aurora MySQL Version 3 den Feature-Set der Community MySQL 8.0.23. Einige neue Funktionen der MySQL 8.0 Community Edition gelten nicht für Aurora MySQL. Einige dieser Funktionen sind mit einigen Aurora-Aspekten, wie der Aurora-Speicherarchitektur, nicht kompatibel. Andere Funktionen sind nicht erforderlich, da der Amazon-RDS-Verwaltungsservice gleichwertige Funktionen bietet. Die folgenden Funktionen in der Community MySQL 8.0 werden in Aurora-MySQL-Version 3 nicht unterstützt oder funktionieren anders.

Die Versionshinweise für alle Versionen von Aurora MySQL Version 3 finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora MySQL Version 3 in den Versionshinweisen für Aurora MySQL.

Themen

- MySQL-8.0-Funktionen sind in Aurora-MySQL-Version 3 nicht verfügbar
- Rollenbasiertes Berechtigungsmodell
- Finden der Datenbankserver-ID
- Authentifizierung

MySQL-8.0-Funktionen sind in Aurora-MySQL-Version 3 nicht verfügbar

Die folgenden Funktionen der Community MySQL 8.0 sind in Aurora MySQL Version 3 nicht verfügbar oder funktionieren anders.

- Ressourcengruppen und zugehörige SQL-Anweisungen werden in Aurora MySQL nicht unterstützt.
- Aurora MySQL unterstützt keine benutzerdefinierten Undo-Tablespaces und die dazugehörigen SQL-Anweisungen wie `CREATE UNDO TABLESPACE`, `ALTER UNDO TABLESPACE ... SET INACTIVE` und `DROP UNDO TABLESPACE`.
- Aurora MySQL unterstützt keine Undo-Tablespace-Kürzungen für Aurora-MySQL-Versionen unter 3.06. In Aurora-MySQL-Version 3.06 und höher werden automatisierte Undo-Tablespace-Kürzungen unterstützt.
- Das Plugin für die Passwortvalidierung wird unterstützt.
- Sie können die Einstellungen von MySQL-Plugins nicht ändern, einschließlich des Plugins für die Passwortvalidierung.
- Das X-Plugin wird nicht unterstützt.
- Multisource-Replikation wird nicht unterstützt.

Rollenbasiertes Berechtigungsmodell

Mit Aurora MySQL Version 3 können Sie die Tabellen im`mysql`-Datenbank direkt. Insbesondere können Sie Benutzer nicht einrichten, indem Sie in die`mysql.user`-Tabelle Stattdessen verwenden Sie SQL-Anweisungen, um rollenbasierte Berechtigungen zu gewähren. Sie können auch keine anderen Objekte wie gespeicherte Prozeduren in der `mysql`-Datenbank erstellen. Sie können immer noch die `mysql`-Tabellen abfragen. Wenn Sie die Binärprotokollreplikation verwenden, werden Änderungen direkt am`mysql`Tabellen im Quellcluster werden nicht auf den Zielcluster repliziert.

In einigen Fällen verwendet Ihre Anwendung möglicherweise Verknüpfungen, um Benutzer oder andere Objekte zu erstellen, indem Sie sie in die`mysql`-Tabellen Wenn ja, ändern Sie Ihren Anwendungscode, um die entsprechenden Anweisungen wie `CREATE USER` zu verwenden. Wenn Ihre Anwendung gespeicherte Prozeduren oder andere Objekte in der `mysql`-Datenbank erstellt, verwenden Sie stattdessen eine andere Datenbank.

Um Metadaten für Datenbankbenutzer während der Migration aus einer externen MySQL-Datenbank zu exportieren, können Sie einen MySQL-Shell-Befehl anstelle von `mysqldump` verwenden. Weitere Informationen finden Sie unter Dienstprogramme für Instance-, Schema- und Tabellen-Dumps.

Um die Verwaltung von Berechtigungen für viele Benutzer oder Anwendungen zu vereinfachen, können Sie`CREATE ROLE`-Anweisung zum Erstellen einer Rolle mit einer Reihe von Berechtigungen. Dann können Sie die`GRANT`und`SET ROLE`-Anweisungen und die`current_role`Funktion, um Benutzern oder Anwendungen Rollen zuzuweisen, die aktuelle Rolle zu wechseln und zu überprüfen, welche Rollen in Kraft sind. Weitere Informationen zum rollenbasierten Berechtigungssystem in MySQL 8.0 finden Sie unter Verwenden von Rollen im MySQL-Referenzhandbuch.

> **Important**
>
> Wir empfehlen Ihnen, den Hauptbenutzer nicht direkt in Ihren Anwendungen zu verwenden. Bleiben Sie stattdessen bei der bewährten Methode, einen Datenbankbenutzer zu verwenden, der mit den Mindestberechtigungen erstellt wurde, die für Ihre Anwendung erforderlich sind.

Themen

- rds_superuser_role
- Benutzer zur Berechtigungsprüfung für die Replikation von Binärprotokollen
- Rollen für den Zugriff auf andere AWS Dienste

rds_superuser_role

Aurora-MySQL-Version 3 enthält eine spezielle Rolle, die alle folgenden Berechtigungen besitzt. Der Name der Rolle lautet `rds_superuser_role`. Dem primären Administratorbenutzer für jeden Cluster wurde diese Rolle bereits gewährt. Die`rds_superuser_role`enthält die folgenden Berechtigungen für alle Datenbankobjekte:

- `ALTER`
- `APPLICATION_PASSWORD_ADMIN`
- `ALTER ROUTINE`
- `CONNECTION_ADMIN`
- `CREATE`
- `CREATE ROLE`
- `CREATE ROUTINE`
- `CREATE TEMPORARY TABLES`

- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- DROP ROLE
- EVENT
- EXECUTE
- FLUSH_OPTIMIZER_COSTS (Aurora-MySQL-Version 3.09 und höher)
- FLUSH_STATUS (Aurora-MySQL-Version 3.09 und höher)
- FLUSH_TABLES (Aurora-MySQL-Version 3.09 und höher)
- FLUSH_USER_RESOURCES (Aurora-MySQL-Version 3.09 und höher)
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT
- SHOW DATABASES
- SHOW_ROUTINE (Aurora-MySQL-Version 3.04 und höher)
- SHOW VIEW
- TRIGGER
- UPDATE
- XA_RECOVER_ADMIN

Die Rollendefinition umfasst auch`WITH GRANT OPTION`damit ein Administratorbenutzer diese Rolle anderen Benutzern gewähren kann. Insbesondere muss der Administrator alle Berechtigungen erteilen, die zur Durchführung der Binärprotokollreplikation mit dem Aurora MySQL-Cluster als Ziel erforderlich sind.

> **Tip**
>
> Um die vollständigen Details der Berechtigungen anzuzeigen, geben Sie die folgenden Anweisungen ein.
>
> ```
> SHOW GRANTS FOR rds_superuser_role@'%';
> SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
> ```

Benutzer zur Berechtigungsprüfung für die Replikation von Binärprotokollen

Aurora-MySQL-Version 3 enthält einen Benutzer zur Berechtigungsprüfung für die Replikation von Binärprotokollen (Binlog), `rdsrepladmin_priv_checks_user`. Zusätzlich zu den Berechtigungen einer `rds_superuser_role` verfügt dieser Benutzer über die `replication_applier`-Berechtigung.

Wenn Sie die Binlog-Replikation aktivieren, indem Sie die gespeicherte Prozedur `mysql.rds_start_replication` aufrufen, wird `rdsrepladmin_priv_checks_user` erstellt.

Der Benutzer `rdsrepladmin_priv_checks_user@localhost` ist ein reservierter Benutzer. Ändern Sie ihn nicht.

Rollen für den Zugriff auf andere AWS Dienste

Aurora MySQL Version 3 enthält Rollen, mit denen Sie auf andere AWS Dienste zugreifen können. Viele dieser Rollen lassen sich alternativ zur Vergabe von Berechtigungen festlegen. Sie geben beispielsweise`GRANT AWS_LAMBDA_ACCESS TO user`INSTEAD OF`GRANT INVOKE LAMBDA ON *.* TO user`aus. Die Verfahren für den Zugriff auf andere AWS Dienste finden Sie unterIntegrieren von Amazon Aurora MySQL in anderen AWS-Services. Aurora MySQL Version 3 umfasst die folgenden Rollen im Zusammenhang mit dem Zugriff auf andere AWS Dienste:

- `AWS_LAMBDA_ACCESS` – Eine Alternative zur `INVOKE LAMBDA`-Berechtigung. Weitere Informationen zur Nutzung finden Sie unter Aufrufen einer Lambda-Funktion aus einem Amazon Aurora MySQL-DB-Cluster.

- `AWS_LOAD_S3_ACCESS` – Eine Alternative zur `LOAD FROM S3`-Berechtigung. Weitere Informationen zur Nutzung finden Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket.
- `AWS_SELECT_S3_ACCESS` – Eine Alternative zur `SELECT INTO S3`-Berechtigung. Weitere Informationen zur Nutzung finden Sie unter Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket.
- `AWS_COMPREHEND_ACCESS` – Eine Alternative zur `INVOKE COMPREHEND`-Berechtigung. Weitere Informationen zur Nutzung finden Sie unter Erteilen des Zugriffs auf Aurora Machine Learning für Datenbankbenutzer.
- `AWS_SAGEMAKER_ACCESS` – Eine Alternative zur `INVOKE SAGEMAKER`-Berechtigung. Weitere Informationen zur Nutzung finden Sie unter Erteilen des Zugriffs auf Aurora Machine Learning für Datenbankbenutzer.
- `AWS_BEDROCK_ACCESS` – Es gibt keine entsprechende `INVOKE`-Berechtigung für Amazon Bedrock. Weitere Informationen zur Nutzung finden Sie unter Erteilen des Zugriffs auf Aurora Machine Learning für Datenbankbenutzer.

Wenn Sie Zugriff mithilfe von Rollen in Aurora MySQL Version 3 gewähren, aktivieren Sie die Rolle auch mithilfe der`SET ROLE` *`role_name`*oder`SET ROLE ALL`statement. Im folgenden Beispiel wird gezeigt, wie dies geschieht. Ersetzen Sie den entsprechenden Rollennamen für `AWS_SELECT_S3_ACCESS`.

```
# Grant role to user.

mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role
 has not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
```

```
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-------------------------------------------------------+
| CURRENT_ROLE()                                        |
+-------------------------------------------------------+
| `AWS_SELECT_S3_ACCESS`@`%`,`rds_superuser_role`@`%` |
+-------------------------------------------------------+
```

Finden der Datenbankserver-ID

Die Datenbankserver-ID (`server_id`) ist für die Replikation von Binärprotokollen (Binlog) erforderlich. Die Methode, die Server-ID zu finden, unterscheidet sich in Aurora MySQL von Community MySQL.

In Community MySQL ist die Server-ID eine Zahl, die Sie mithilfe der folgenden Syntax erhalten, während Sie beim Server angemeldet sind:

```
mysql> select @@server_id;

+-------------+
| @@server_id |
+-------------+
| 2           |
+-------------+
1 row in set (0.00 sec)
```

In Aurora MySQL ist die Server-ID die DB-Instance-ID, die Sie mithilfe der folgenden Syntax erhalten, während Sie bei der DB-Instance angemeldet sind:

```
mysql> select @@aurora_server_id;

+------------------------+
| @@aurora_server_id     |
+------------------------+
| mydbcluster-instance-2 |
+------------------------+
1 row in set (0.00 sec)
```

Weitere Informationen zur Binlog-Replikation finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Authentifizierung

In Community MySQL 8.0 ist das Standard-Authentifizierungs-Plugin `caching_sha2_password`. Aurora MySQL Version 3 verwendet immer noch`mysql_native_password`-Plugin. Sie können die `default_authentication_plugin`-Einstellung nicht ändern. Sie können jedoch neue Benutzer erstellen und aktuelle Benutzer sowie deren individuelle Passwörter ändern, indem Sie das neue Authentifizierungs-Plugin verwenden. Im Folgenden sehen Sie ein Beispiel.

```
mysql> CREATE USER 'testnewsha'@'%' IDENTIFIED WITH caching_sha2_password BY
 'aNewShaPassword';
Query OK, 0 rows affected (0.74 sec)
```

Upgrade auf Aurora MySQL Version 3

Weitere Informationen zu Upgrades Ihrer Datenbank von Aurora-MySQL-Version 2 zu Version 3 finden Sie unter Aktualisieren der Hauptversion eines DB-Clusters von Amazon Aurora MySQL.

Aurora-MySQL-Version 2, kompatibel mit MySQL 5.7

In diesem Thema werden die Unterschiede zwischen Aurora-MySQL-Version 2 und MySQL 5.7 Community Edition beschrieben.

> **Important**
>
> Aurora-MySQL-Version 2 erreichte am 31. Oktober 2024 das Ende des Standard-Supports. Weitere Informationen finden Sie unter Vorbereitung auf das Ende des Standard-Supports für die Amazon-Aurora-MySQL-kompatible Edition der Version 2.

In Aurora-MySQL-Version 2 nicht unterstützte Funktionen

Die folgenden Funktionen werden in MySQL 5.7 unterstützt, in Aurora MySQL Version 2 derzeit jedoch nicht:

- `CREATE TABLESPACE`-SQL-Anweisung
- Plugin für die Gruppenreplikation
- Größere Seitengröße

- Laden des InnoDB-Pufferpools beim Starten
- Plugin für den InnoDB-Volltext-Parser
- Replikation aus mehreren Quellen
- Größenanpassung des Online-Pufferpools
- Plugin zur Passwortüberprüfung – Sie können das Plugin installieren, es wird jedoch nicht unterstützt. Sie können das Plugin nicht anpassen.
- Plugins für die Umformulierung von Abfragen
- Replikationsfilter
- X Protocol

Weitere Informationen über diese Funktionen finden Sie in der MySQL 5.7 Dokumentation.

Temporäres Tabellenverhalten in Aurora-MySQL-Version 2

In MySQL 5.7 wird der temporäre Tabellenbereich automatisch erweitert und vergrößert sich nach Bedarf, um temporäre Tabellen auf der Festplatte unterzubringen. Wenn temporäre Tabellen gelöscht werden, kann der freigewordene Speicherplatz für neue temporäre Tabellen wiederverwendet werden, der temporäre Tabellenbereich behält jedoch die erweiterte Größe bei und wird nicht verkleinert. Der temporäre Tabellenbereich wird gelöscht und neu erstellt, wenn die Engine neu gestartet wird.

In Aurora MySQL Version 2 gilt das folgende Verhalten:

- Bei neuen DB-Clustern von Aurora MySQL, die mit Version 2.10 und höher erstellt wurden, wird der temporäre Tabellenbereich entfernt und neu erstellt, wenn Sie die Datenbank neu starten. Dadurch kann die Funktion zur dynamischen Größenanpassung Speicherplatz zurückgewinnen.
- Bei bestehenden DB-Clustern von Aurora MySQL, die auf folgende Versionen aktualisiert wurden:
  - Version 2.10 oder höher – Der temporäre Tabellenbereich wird entfernt und neu erstellt, wenn Sie die Datenbank neu starten. Dadurch kann die Funktion zur dynamischen Größenanpassung Speicherplatz zurückgewinnen.
  - Version 2.09 – Der temporäre Tabellenbereich wird nicht entfernt, wenn Sie die Datenbank neu starten.

Sie können die Größe des temporären Tabellenbereichs auf Ihrem DB-Cluster von Aurora MySQL Version 2 mithilfe der folgenden Abfrage überprüfen:

```
SELECT
    FILE_NAME,
    TABLESPACE_NAME,
    ROUND((TOTAL_EXTENTS * EXTENT_SIZE) / 1024 / 1024 / 1024, 4) AS SIZE
FROM
    INFORMATION_SCHEMA.FILES
WHERE
    TABLESPACE_NAME = 'innodb_temporary';
```

Weitere Informationen finden Sie unter Der temporäre Tabellenbereich in der MySQL-Dokumentation.

Speicher-Engine für temporäre Tabellen auf der Festplatte

Aurora MySQL Version 2 verwendet je nach Rolle der Instance unterschiedliche Speicher-Engines für interne temporäre Tabellen auf der Festplatte.

- Auf der Writer-Instance verwenden temporäre Tabellen auf der Festplatte standardmäßig die InnoDB-Speicher-Engine. Sie sind im temporären Tabellenbereich im Aurora-Cluster-Volume gespeichert.

  Sie können dieses Verhalten auf der Writer-Instance ändern, indem Sie den Wert für den DB-Parameter `internal_tmp_disk_storage_engine` ändern. Weitere Informationen finden Sie unter Parameter auf Instance-Ebene.
- Auf Reader-Instances nutzen temporäre Tabellen auf der Festplatte die MyISAM-Speicher-Engine, die lokalen Speicher verwendet. Dies liegt daran, dass schreibgeschützte Instances keine Daten auf dem Aurora-Cluster-Volume speichern können.

Sicherheit in Amazon Aurora MySQL

Die Sicherheit für Amazon Aurora MySQL wird auf drei Ebenen verwaltet:

- Mit AWS Identity and Access Management (IAM) können Sie steuern, wer Verwaltungsaktionen in Amazon RDS für Aurora-MySQL-DB-Cluster und DB-Instances ausführen darf. Wenn Sie eine Verbindung zu AWS mithilfe von IAM-Anmeldeinformationen herstellen, muss Ihr AWS-Konto über IAM-Richtlinien verfügen, die die erforderlichen Berechtigungen zum Ausführen von Amazon-RDS-Verwaltungsvorgängen gewähren. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora

  Wenn Sie IAM verwenden, um auf die Amazon-RDS-Konsole zuzugreifen, stellen Sie sicher, dass Sie sich zuerst mit Ihren IAM-Benutzeranmeldeinformationen bei der AWS-Managementkonsole anmelden. Öffnen Sie dann die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
- Stellen Sie sicher, dass Sie Aurora-MySQL-DB-Cluster in einer Virtual Public Cloud (VPC) basierend auf dem Amazon-VPC-Service erstellen. Mithilfe einer VPC-Sicherheitsgruppe können Sie steuern, welche Geräte und Amazon-EC2-Instances Verbindungen zum Endpunkt und Port der DB-Instance für Aurora MySQL-DB-Cluster in einer VPC herstellen können. Diese Endpunkt- und Portverbindungen können mithilfe von Transport Layer Security (TLS) erstellt werden. Zusätzlich können Firewall-Regeln in Ihrem Unternehmen steuern, ob in Ihrem Unternehmen verwendete Geräte Verbindungen mit einer DB-Instance herstellen dürfen. Weitere Informationen zu VPCs finden Sie unter Amazon VPC und Amazon Aurora.

  Die unterstützte VPC-Tenancy hängt von der DB-Instance-Klasse ab, die von Ihren Aurora MySQL DB-Clustern verwendet wird. Mit `default`-VPC-Tenancy, the VPC läuft der VPC auf gemeinsam genutzter Hardware. Mit der `dedicated`-VPC-Tenancy läuft die VPC auf einer dedizierten Hardware-Instance. Die DB-Instance-Klassen mit Spitzenlastleistung unterstützen nur die Standard-VPC-Tenancy. Die Burstable-Performance-DB-Instance-Klassen umfassen die DB-Instance-Klassen db.t2, db.t3 und db.t4g. Alle anderen Aurora MySQL DB-Instance-Klassen unterstützen sowohl die Standard- als auch die dedizierte VPC-Mieterschaft.

  > **Note**
  >
  > Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter Verwendung von T-Instance-Klassen für Entwicklung und Tests.

Weitere Informationen zu Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora. Weitere Informationen über die `default`- und `dedicated`-VPC-Tenancy finden Sie unter Dedicated Instances im Amazon Elastic Compute Cloud-Benutzerhandbuch.

- Sie können eine der folgenden Anweisungen, oder eine Kombination davon, befolgen, um die Anmeldung und die Berechtigungen für ein Amazon Aurora MySQL-DB-Cluster zu bestätigen:
  - Sie können denselben Ansatz wie mit einer Stand-Alone Instance in MySQL wählen.

    Befehle wie `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE` und `SET PASSWORD` funktionieren genau wie auf lokalen Datenbanken, so wie auch das direkte Ändern von Datenbank-Schema-Tabellen. Weitere Informationen finden Sie unter Access Control and Account Management in der MySQL-Dokumentation.
  - Sie können auch die IAM-Datenbank-Authentifizierung verwenden.

    Mit der IAM-Datenbank-Authentifizierung können Sie mithilfe eines IAM-Benutzers, einer IAM-Rolle oder eines Authentifizierungstokens Ihr DB-Cluster bestätigen. Ein Authentifizierungstoken ist ein eindeutiger Wert, der mithilfe des Signatur-Version 4-Signiervorgangs erstellt wird. Durch das Verwenden der IAM-Datenbank-Authentifizierung können Sie dieselben Anmeldeinformationen verwenden, um den Zugang zu Ihren AWS-Ressourcen und Ihrer Datenbank zu steuern. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung.

> **Note**
>
> Weitere Informationen finden Sie unter Sicherheit in Amazon Aurora.

In den folgenden Abschnitten finden Sie Informationen zu Benutzerberechtigungen für Aurora-MySQL- und TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

Themen

- Berechtigungen von Masterbenutzerkonten mit Amazon Aurora MySQL.
- TLS-Verbindungen mit DB-Clustern von Aurora MySQL

Berechtigungen von Masterbenutzerkonten mit Amazon Aurora MySQL.

Wenn Sie eine DB-Instance von Amazon Aurora MySQL erstellen, verfügt der Hauptbenutzer standardmäßig über die Berechtigungen, die unter Berechtigungen von Hauptbenutzerkonten aufgeführt sind.

Um Verwaltungsservices für jeden DB-Cluster bereitzustellen, werden die `admin`- und `rdsadmin`-Benutzer erstellt, wenn der DB-Cluster erstellt wird. Ein Versuch, das Passwort auszulassen oder umzubenennen, oder Berechtigungen für das `rdsadmin` Konto zu ändern, führt zu einem Fehler.

In Aurora-MySQL-Version 2-DB-Clustern werden die `admin`- und `rdsadmin`-Benutzer erstellt, wenn der DB-Cluster erstellt wird. In Aurora-MySQL-Version 3-Clustern werden die `admin`-, `rdsadmin` und `rds_superuser_role`-Benutzer erstellt.

> **Important**
>
> Wir empfehlen Ihnen, den Hauptbenutzer nicht direkt in Ihren Anwendungen zu verwenden. Bleiben Sie stattdessen bei der bewährten Methode, einen Datenbankbenutzer zu verwenden, der mit den Mindestberechtigungen erstellt wurde, die für Ihre Anwendung erforderlich sind.

Für die Verwaltung des Aurora MySQL-DB-Clusters wurden die Standardbefehle `kill` und `kill_query` eingeschränkt. Verwenden Sie stattdessen die Amazon-RDS-Befehle `rds_kill` und `rds_kill_query`, um Benutzersitzungen oder Abfragen in Ihren Aurora MySQL-DB-Instances zu beenden.

> **Note**
>
> Die Verschlüsselung einer Datenbank-Instance und von Snapshots wird für die Region China (Ningxia) nicht unterstützt.

TLS-Verbindungen mit DB-Clustern von Aurora MySQL

DB-Cluster von Amazon Aurora MySQL unterstützen Transport Layer Security (TLS)-Verbindungen von Anwendungen mithilfe des gleichen Prozesses und öffentlichen Aktivierungsschlüssels wie DB-Instances von RDS für MySQL.

Amazon RDS erstellt ein TLS-Zertifikat und installiert das Zertifikat auf der DB-Instance, wenn Amazon RDS die Instance bereitstellt. Diese Zertifikate werden von einer Zertifizierungsstelle signiert. Das TLS-Zertifikat enthält den DB-Instance-Endpunkt als allgemeinen Namen (Common Name, CN) für das TLS-Zertifikat, um gegen Spoofing-Angriffe zu schützen. Dadurch können Sie nur den DB-Cluster-Endpunkt verwenden, um sich mit einem DB-Cluster mithilfe von TLS zu verbinden, wenn Ihr Client SAN (Subject Alternative Names) unterstützt. Andernfalls müssen Sie den Instance-Endpunkt einer Writer-Instance verwenden.

Informationen zum Herunterladen von Zertifikaten finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

Wir empfehlen den AWS-JDBC-Treiber als Client, der SAN mit TLS unterstützt. Weitere Informationen zum AWS-JDBC-Treiber und vollständige Anleitungen zur Verwendung finden Sie im GitHub-Repository für den JDBC-Treiber von Amazon Web Services (AWS).

Themen

- Anfordern einer TLS-Verbindung mit einem DB-Cluster von Aurora MySQL
- TLS-Versionen für Aurora MySQL
- Konfigurieren von Cipher-Suites für Verbindungen mit Aurora-MySQL-DB-Clustern
- Verschlüsseln von Verbindungen zu einem Aurora MySQL-DB-Cluster

Anfordern einer TLS-Verbindung mit einem DB-Cluster von Aurora MySQL

Mit dem DB-Clusterparameter `require_secure_transport` können Sie es erforderlich machen, dass Benutzerverbindungen mit Ihrem DB-Cluster von Aurora MySQL TLS verwenden. Standardmäßig ist der `require_secure_transport`-Parameter auf `OFF` festgelegt. Sie können den Parameter `require_secure_transport` auf `ON` festlegen und damit erforderlich machen, dass Verbindungen mit Ihrem DB-Cluster TLS verwenden.

Sie können den Parameterwert `require_secure_transport` festlegen, indem Sie die Parametergruppe für Ihren DB-Cluster aktualisieren. Sie müssen Ihren DB-Cluster nicht neu starten, damit die Änderung wirksam wird. Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

> **Note**
>
> Der Parameter `require_secure_transport` ist nur für Aurora-MySQL-Version 2 und 3 verfügbar. Sie können diesen Parameter in einer benutzerdefinierten DB-Cluster-

Parametergruppe festlegen. Der Parameter ist in DB-Instance-Parametergruppen nicht verfügbar.

Wenn der Parameter `require_secure_transport` für einen DB-Cluster auf `ON` festgelegt ist, kann ein Datenbankclient eine Verbindung zu ihm herstellen, wenn er eine verschlüsselte Verbindung herstellen kann. Andernfalls wird eine Fehlermeldung ähnlich der folgenden an den Client zurückgegeben:

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --
require_secure_transport=ON.
```

TLS-Versionen für Aurora MySQL

Aurora MySQL unterstützt die Transport Layer Security (TLS)-Versionen 1.0, 1.1, 1.2 und 1.3. Ab Aurora-MySQL-Version 3.04.0 und höher können Sie das TLS-1.3-Protokoll verwenden, um Ihre Verbindungen zu schützen. In der folgenden Tabelle wird die TLS-Unterstützung für Aurora MySQL-Versionen gezeigt.

Aurora MySQL Version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Standard
Aurora-MySQL-Version 2	Als veraltet gekennzeichnet	Als veraltet gekennzeichnet	Unterstützt	Nicht unterstützt	Alle unterstützten TLS-Versionen
Aurora-MySQL-Version 3 (niedriger als 3.04.0)	Als veraltet gekennzeichnet	Als veraltet gekennzeichnet	Unterstützt	Nicht unterstützt	Alle unterstützten TLS-Versionen
Aurora-MySQL-Version 3 (3.04.0 und höher)	Nicht unterstützt	Nicht unterstützt	Unterstützt	Unterstützt	Alle unterstützten TLS-Versionen

> **Important**
>
> Wenn Sie benutzerdefinierte Parametergruppen für Ihre Aurora-MySQL-Cluster mit Version 2 und Versionen vor 3.04.0 verwenden, empfehlen wir die Verwendung von TLS 1.2, da TLS 1.0 und 1.1 weniger sicher sind. In der Community-Edition von MySQL 8.0.26 und Aurora MySQL 3.03 und ihren Unterversionen wurde die Unterstützung für die TLS-Versionen 1.1 und 1.0 eingestellt.
> Die Community-Edition von MySQL 8.0.28 und die kompatiblen Aurora-MySQL-Versionen 3.04.0 und höher unterstützen TLS 1.1 und TLS 1.0 nicht. Wenn Sie Aurora-MySQL-Version 3.04.0 oder höher verwenden, legen Sie das TLS-Protokoll in Ihrer benutzerdefinierten Parametergruppe nicht auf 1.0 oder 1.1 fest.
> Für Aurora-MySQL-Versionen 3.04.0 und höher ist die Standardeinstellung TLS 1.3 und TLS 1.2.

Sie können den DB-Cluster-Parameter `tls_version` zur Angabe der zulässigen Protokollversionen verwenden. Vergleichbare Clientparameter sind für die meisten Client-Tools oder Datenbanktreiber verfügbar. Einige ältere Clients unterstützen möglicherweise keine neueren TLS-Versionen. Standardmäßig versucht der DB-Cluster, die höchste TLS-Protokollversion zu verwenden, die sowohl von der Server- als auch von der Clientkonfiguration zugelassen wird.

Legen Sie den `tls_version`-DB-Cluster-Parameter auf einen der folgenden Werte fest:

- `TLSv1.3`
- `TLSv1.2`
- `TLSv1.1`
- `TLSv1`

Sie können auch den Parameter `tls_version` auch als Zeichenfolge einer durch Kommas getrennten Liste einstellen. Wenn Sie sowohl die Protokolle TLS 1.2 als auch TLS 1.3 verwenden möchten, muss der Parameter `tls_version` alle Protokolle vom niedrigsten bis zum höchsten enthalten. In diesem Fall ist `tls_version` eingestellt als:

```
tls_version=TLSv1.2,TLSv1.3
```

Weitere Informationen zum Ändern von Parametern in einer DB-Cluster-Parametergruppe finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

Wenn Sie AWS CLI verwenden, um den DB-Cluster-Parameter `tls_version` zu ändern, muss `ApplyMethod` auf `pending-reboot` gesetzt werden. Wenn die Anwendungsmethode `pending-reboot` ist, werden Änderungen an Parametern angewendet, nachdem Sie die der Parametergruppe zugeordneten DB-Cluster gestoppt und neu gestartet haben.

Konfigurieren von Cipher-Suites für Verbindungen mit Aurora-MySQL-DB-Clustern

Durch die Verwendung von konfigurierbaren Chiffrier-Suiten können Sie mehr Kontrolle über die Sicherheit Ihrer Datenbankverbindungen haben. Sie können eine Liste von Verschlüsselungssammlungen angeben, die Sie zum Sichern von TLS-Verbindungen mit Ihrer Datenbank des Clients zulassen möchten. Mit konfigurierbaren Chiffrier-Suiten können Sie die Verbindungsverschlüsselung steuern, die Ihr Datenbankserver akzeptiert. Dadurch wird verhindert, dass unsichere oder veraltete Chiffren verwendet werden.

Konfigurierbare Verschlüsselungssuites werden in Aurora-MySQL-Version 3 und Aurora-MySQL-Version 2 unterstützt. Um die Liste der zulässigen Verschlüsselungsverfahren TLS 1.2, TLS 1.1, TLS 1.0 für die Verschlüsselung von Verbindungen anzugeben, ändern Sie den Cluster-Parameter `ssl_cipher`. Legen Sie den Wert für `ssl_cipher`-Parameter in einer Cluster-Parametergruppe mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API fest.

Legen Sie den Parameter `ssl_cipher` auf eine Zeichenfolge von kommagetrennten Verschlüsselungswerten für Ihre TLS-Version fest. Für die Clientanwendung können Sie die Chiffren angeben, die für verschlüsselte Verbindungen verwendet werden sollen, indem Sie beim Herstellen der Verbindung mit einer Datenbank die Option `--ssl-cipher` verwenden. Weitere Information zur Herstellung einer Verbindung mit Ihrer Datenbank finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora MySQL-DB-Cluster.

Ab Aurora-MySQL-Version 3.04.0 und höher können Sie TLS-1.3-Verschlüsselungssuiten angeben. Um die zulässigen TLS-1.3-Verschlüsselungssuites zu spezifizieren, ändern Sie den Parameter `tls_ciphersuites` in Ihrer Parametergruppe. TLS 1.3 hat die Anzahl der verfügbaren Verschlüsselungssuites reduziert, da die Benennungskonvention geändert wurde, wodurch der Schlüsselaustauschmechanismus und das verwendete Zertifikat entfernt werden. Legen Sie den Wert für `tls_ciphersuites` auf eine Zeichenfolge von kommagetrennten Verschlüsselungswerten für TLS 1.3 fest.

Die folgende Tabelle zeigt die unterstützten Chiffren zusammen mit dem TLS-Verschlüsselungsprotokoll und gültigen Aurora-MySQL-Engine-Versionen für jede Chiffre.

Verschlüsselungsverfahren	Verschlüsselungsprotokoll	Unterstützte Aurora-MySQL-Versionen
ECDHE-RSA-AES128-SHA	TLS 1.0	3.04.0 und höher, 2.11.0 und höher
ECDHE-RSA-AES128-SHA256	TLS 1.2	3.04.0 und höher, 2.11.0 und höher
ECDHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.04.0 und höher, 2.11.0 und höher
ECDHE-RSA-AES256-SHA	TLS 1.0	3.04.0 und höher, 2.11.0 und höher
ECDHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.04.0 und höher, 2.11.0 und höher
ECDHE-RSA-CHACHA20-POLY1305	TLS 1.2	3.04.0 und höher, 2.11.0 und höher
ECDHE-ECDSA-AES128-SHA	TLS 1.0	3.04.0 und höher, 2.11.0 und höher
ECDHE-ECDSA-AES256-SHA	TLS 1.0	3.04.0 und höher, 2.11.0 und höher
ECDHE-ECDSA-AES128-GCM-SHA256	TLS 1.2	3.04.0 und höher, 2.11.0 und höher
ECDHE-ECDSA-AES256-GCM-SHA384	TLS 1.2	3.04.0 und höher, 2.11.0 und höher
ECDHE-ECDSA-CHACHA20-POLY1305	TLS 1.2	3.04.0 und höher, 2.11.0 und höher
TLS_AES_128_GCM_SHA256	TLS 1.3	3.04.0 und höher

Verschlüsselungsverfahren	Verschlüsselungsprotokoll	Unterstützte Aurora-MySQL-Versionen
`TLS_AES_256_GCM_SHA384`	TLS 1.3	3.04.0 und höher
`TLS_CHACHA20_POLY1305_SHA256`	TLS 1.3	3.04.0 und höher

Weitere Informationen zum Ändern von Parametern in einer DB-Cluster-Parametergruppe finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora. Wenn Sie die CLI verwenden, um den DB-Cluster-Parameter `ssl_cipher` zu ändern, stellen Sie sicher, dass Sie die `ApplyMethod` auf `pending-reboot` festlegen. Wenn die Anwendungsmethode `pending-reboot` ist, werden Änderungen an Parametern angewendet, nachdem Sie die der Parametergruppe zugeordneten DB-Cluster gestoppt und neu gestartet haben.

Sie können auch den describe-motor-default-cluster-parameters-CLI-Befehl verwenden, um festzustellen, welche Chiffrier-Suiten derzeit für eine bestimmte Parametergruppenfamilie unterstützt werden. Im folgenden Beispiel wird veranschaulicht, wie Sie die zulässigen Werte für `ssl_cipher`-Cluster-Parameter für Aurora-MySQL-Version 2 abrufen.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
mysql5.7

        ...some output truncated...
 {
        "ParameterName": "ssl_cipher",
        "ParameterValue": "ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-
AES128-GCM-SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-CHACHA20-
POLY1305,ECDHE-ECDSA-AES256-SHA,ECDHE-ECDSA-CHACHA20-POLY1305,ECDHE-ECDSA-AES256-GCM-
SHA384,ECDHE-ECDSA-AES128-GCM-SHA256,ECDHE-ECDSA-AES128-SHA",
        "Description": "The list of permissible ciphers for connection encryption.",
        "Source": "system",
        "ApplyType": "static",
        "DataType": "list",
        "AllowedValues": "ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-
AES128-GCM-SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-CHACHA20-
POLY1305,ECDHE-ECDSA-AES256-SHA,ECDHE-ECDSA-CHACHA20-POLY1305,ECDHE-ECDSA-AES256-GCM-
SHA384,ECDHE-ECDSA-AES128-GCM-SHA256,ECDHE-ECDSA-AES128-SHA",
        "IsModifiable": true,
```

```
        "SupportedEngineModes": [
            "provisioned"
        ]
    },
       ...some output truncated...
```

Weitere Informationen zu Chiffren finden Sie in der ssl_cipher-Variablen in der MySQL-Dokumentation. Weitere Informationen zu Chiffrier-Suite-Formaten finden Sie unter openssl-ciphers list format und openssl-ciphers string format-Dokumentation auf der OpenSSL-Website.

Verschlüsseln von Verbindungen zu einem Aurora MySQL-DB-Cluster

Um die vom Standard-`mysql`-Client verwendeten Verbindungen zu verschlüsseln, starten Sie den mysql-Client mithilfe des Parameters `--ssl-ca`, um auf den öffentlichen Schlüssel zu verweisen, beispielsweise:

Für MySQL 5.7 und 8.0:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

Für MySQL 5.6:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

Ersetzen Sie *`full_path_to_CA_certificate`* mit dem vollständigen Pfad zum Zertifikat Ihrer Zertifizierungsstelle (CA). Informationen zum Herunterladen von Zertifikaten finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

Sie können TLS-Verbindungen für bestimmte Benutzerkonten anfordern. Verwenden Sie beispielsweise eine der folgenden Anweisungen – abhängig von Ihrer MySQL-Version – um /TLS-Verbindungen für das Benutzerkonto `encrypted_user` erforderlich zu machen.

Für MySQL 5.7 und 8.0:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

Für MySQL 5.6:

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

Wenn Sie einen RDS-Proxy verwenden, stellen Sie eine Verbindung mit dem Proxyendpunkt anstelle des üblichen Cluster-Endpunkts her. Sie können SSL/TLS für Verbindungen mit dem Proxy auf die gleiche Weise wie für Verbindungen direkt mit dem Aurora-DB-Cluster als erforderlich oder optional festlegen. Informationen zur Verwendung von RDS-Proxy finden Sie unter Amazon-RDS-Proxy für Aurora.

> **Note**
>
> Weitere Informationen zu TLS-Verbindungen mit MySQL finden Sie in der MySQL-Dokumentation.

Aktualisieren von Anwendungen, um Verbindungen mit DB-Clustern von Aurora MySQL mithilfe neuer TLS-Zertifikate herzustellen

Am 13. Januar 2023 veröffentlichte Amazon RDS neue Zertifizierungsstellen-Zertifikate (Certificate Authority, CA) zum Herstellen von Verbindungen mit Ihren Aurora-DB-Clustern mithilfe von Transport Layer Security (TLS). Im Folgenden finden Sie Informationen dazu, wie Sie Ihre Anwendungen aktualisieren, um die neuen Zertifikate verwenden zu können.

In diesem Thema finden Sie Informationen dazu, wie Sie ermitteln, ob Client-Anwendungen für die Herstellung von Verbindungen mit Ihren DB-Clustern TLS verwenden. Wenn dies der Fall ist, können Sie weiter überprüfen, ob diese Anwendungen zur Herstellung von Verbindungen Zertifikatverifizierungen erfordern.

> **Note**
>
> Einige Anwendungen sind so konfiguriert, dass sie nur dann Verbindungen mit Aurora MySQL-DB-Clustern herstellen, wenn sie das Zertifikat auf dem Server erfolgreich identifizieren können.
> Für solche Anwendungen müssen Sie die Trust Stores Ihrer Client-Anwendung aktualisieren, damit diese die neuen CA-Zertifikate enthalten.

Nach der Aktualisierung der CA-Zertifikate in den Trust Stores Ihrer Client-Anwendungen können Sie die Zertifikate auf Ihren DB-Clustern rotieren. Es wird nachdrücklich empfohlen, diese Verfahren vor der Implementierung in Produktionsumgebungen in einer Entwicklungs- oder Testumgebung zu testen.

Weitere Informationen zur Zertifikatrotation finden Sie unter Rotieren Ihrer SSL/TLS-Zertifikate. Weitere Informationen zum Herunterladen von Zertifikaten finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln. Informationen zum Verwenden von TLS mit DB-Clustern von Aurora MySQL finden Sie unter TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

Themen

- Ermitteln, ob Anwendungen Verbindungen mit Ihrem DB-Cluster von Aurora MySQL mithilfe von TLS herstellen
- Ermitteln, ob ein Client zum Herstellen von Verbindungen Zertifikatverifizierungen erfordert

- Aktualisieren des Trust Stores Ihrer Anwendung
- Java-Beispielcode für die Herstellung von TLS-Verbindungen

Ermitteln, ob Anwendungen Verbindungen mit Ihrem DB-Cluster von Aurora MySQL mithilfe von TLS herstellen

Wenn Sie Aurora MySQL Version 2 verwenden (kompatibel mit MySQL 5.7) und das Performance-Schema aktiviert ist, können Sie die folgende Abfrage ausführen, um zu prüfen, ob Verbindungen TLS verwenden. Informationen zum Aktivieren des Performance-Schemas finden Sie unter Performance Schema Quick Start in der MySQL-Dokumentation.

```
mysql> SELECT id, user, host, connection_type
       FROM performance_schema.threads pst
       INNER JOIN information_schema.processlist isp
       ON pst.processlist_id = isp.id;
```

In dieser Beispielausgabe können Sie sehen, dass sowohl Ihre eigene Sitzung (`admin`), als auch eine als `webapp1` angemeldete Anwendung TLS verwenden.

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

Ermitteln, ob ein Client zum Herstellen von Verbindungen Zertifikatverifizierungen erfordert

Sie können überprüfen, ob JDBC-Clients und MySQL-Clients zum Herstellen von Verbindungen Zertifikatverifizierungen erfordern.

JDBC

Das folgende Beispiel mit MySQL Connector/J 8.0 zeigt eine Möglichkeit, wie Sie die JDBC-Verbindungseigenschaften einer Anwendung überprüfen können, um zu ermitteln, ob zum

erfolgreichen Herstellen von Verbindungen ein gültiges Zertifikat benötigt wird. Weitere Informationen zu allen JDBC-Verbindungsoptionen für MySQL finden Sie unter Configuration Properties in der MySQL-Dokumentation.

Wenn MySQL Connector/J 8.0 verwendet wird, erfordert eine TLS-Verbindung die Verifizierung anhand des CA-Serverzertifikats, wenn in den Verbindungseigenschaften `sslMode` auf `VERIFY_CA` oder `VERIFY_IDENTITY` festgelegt ist, wie im folgenden Beispiel gezeigt.

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

> **Note**
>
> Wenn Sie entweder den MySQL Java Connector v5.1.38 oder höher oder den MySQL Java Connector v8.0.9 oder höher verwenden, um eine Verbindung mit Ihren Datenbanken herzustellen, verwenden diese Clienttreiber selbst dann standardmäßig TLS, wenn Sie Ihre Anwendungen nicht explizit zur Verwendung von TLS beim Verbinden mit Ihren Datenbanken konfiguriert haben. Darüber hinaus führen sie bei Verwendung von TLS eine teilweise Zertifikatüberprüfung durch und stellen keine Verbindung her, wenn das Datenbankserverzertifikat abgelaufen ist.

MySQL

Die folgenden Beispiele mit dem MySQL-Client zeigen zwei Möglichkeiten, wie Sie die MySQL-Verbindung eines Skripts überprüfen, um zu ermitteln, ob zum Herstellen von Verbindungen ein gültiges Zertifikat erforderlich ist. Weitere Informationen zu allen Verbindungsoptionen mit dem MySQL-Client finden Sie unter Client-Side Configuration for Encrypted Connections in der MySQL-Dokumentation.

Wenn der MySQL-5.7- oder MySQL-8.0-Client verwendet wird, erfordert eine TLS-Verbindung die Verifizierung anhand des CA-Serverzertifikats, wenn Sie für die Option `--ssl-mode` den Wert `VERIFY_CA` oder `VERIFY_IDENTITY` angeben, wie im folgenden Beispiel gezeigt.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

Wenn der MySQL 5.6-Client verwendet wird, erfordert eine SSL-Verbindung die Verifizierung anhand des CA-Serverzertifikats, wenn Sie die Option `--ssl-verify-server-cert` angeben wie im folgenden Beispiel gezeigt.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

Aktualisieren des Trust Stores Ihrer Anwendung

Informationen zum Aktualisieren des Trust Stores für MySQL-Anwendungen finden Sie unter Installing SSL Certificates in der MySQL-Dokumentation.

> **Note**
>
> Wenn Sie den Trust Store aktualisieren, können Sie ältere Zertifikate beibehalten und die neuen Zertifikate einfach hinzufügen.

Aktualisieren des Trust Stores Ihrer Anwendung für JDBC

Sie können den Trust Store für Anwendungen aktualisieren, die JDBC für TLS-Verbindungen verwenden.

Informationen zum Herunterladen des Stammverzeichnisses finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

Beispiele für Skripte, die Zertifikate importieren, finden Sie unter Beispielskript für den Import von Zertifikaten in Ihren Trust Store.

Wenn Sie den JDBC-Treiber von MySQL in einer Anwendung verwenden, legen Sie in der Anwendung die folgenden Eigenschaften fest.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

> **Note**
>
> Geben Sie aus Sicherheitsgründen ein anderes Passwort als hier angegeben an.

Legen Sie während des Startens der Anwendung die folgenden Eigenschaften fest.

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

Java-Beispielcode für die Herstellung von TLS-Verbindungen

Im folgenden Codebeispiel wird gezeigt, wie Sie die SSL-Verbindung einrichten, die das Serverzertifikat mithilfe von JDBC validiert.

```
public class MySQLSSLTest {

        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");


        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
}
```

Important

Wenn Sie festgestellt haben, dass Ihre Datenbankverbindungen TLS verwenden, und Sie den Trust Store Ihrer Anwendung aktualisiert haben, können Sie Ihre Datenbank so aktualisieren, dass sie die rds-ca-rsa2048-g1-Zertifikate verwendet. Anweisungen hierzu finden Sie in Schritt 3 unter Aktualisieren des CA-Zertifikats durch Ändern der DB-Instance.

Verwenden der Kerberos-Authentifizierung für Aurora MySQL

Sie können die Kerberos-Authentifizierung verwenden, um Benutzer zu authentifizieren, wenn diese sich mit Ihrem DB-Cluster von Aurora MySQL verbinden. Konfigurieren Sie Ihren DB-Cluster dazu so, dass AWS Directory Service for Microsoft Active Directory für die Kerberos-Authentifizierung verwendet wird. AWS Directory Service for Microsoft Active Directory wird auch als AWS Managed Microsoft AD bezeichnet. Es ist eine Funktion, die mit Directory Service verfügbar ist. Weitere Informationen finden Sie unter Was ist Directory Service? im Administratorhandbuch für AWS Directory Service.

Zunächst erstellen Sie ein AWS Managed Microsoft AD-Verzeichnis, um Benutzeranmeldeinformationen zu speichern. Anschließend stellen Sie Ihrem DB-Cluster von Aurora MySQL die Active Directory Domain und weitere Informationen zur Verfügung. Wenn sich Benutzer beim DB-Cluster von Aurora MySQL authentifizieren, werden Authentifizierungsanforderungen an das AWS Managed Microsoft AD-Verzeichnis weitergeleitet.

Wenn Sie alle Ihre Anmeldeinformationen im selben Verzeichnis aufbewahren, können Sie Zeit und Mühe sparen. Mit diesem Ansatz haben Sie einen zentralen Ort für die Speicherung und Verwaltung von Anmeldeinformationen für mehrere DB-Cluster. Die Verwendung eines Verzeichnisses kann auch Ihr allgemeines Sicherheitsprofil verbessern.

Außerdem können Sie von Ihrem eigenen On-Premises Microsoft Active Directory auf Anmeldeinformationen zugreifen. Dazu erstellen Sie eine vertrauensvolle Domain-Beziehung, damit das AWS Managed Microsoft AD-Verzeichnis Ihrem On-Premises Microsoft Active Directory vertraut. Auf diese Weise können Ihre Benutzer auf Ihre DB-Cluster von Aurora SQL mit derselben Windows Single Sign-On-Oberfläche (SSO) zugreifen, die sie auch für den Zugriff auf Workloads in Ihrem lokalen Netzwerk verwenden.

Eine Datenbank kann Kerberos, AWS Identity and Access Management (IAM) oder sowohl Kerberos- als auch IAM-Authentifizierung nutzen. Da die Kerberos- und IAM-Authentifizierung jedoch unterschiedliche Authentifizierungsmethoden bereitstellen, kann sich ein bestimmter Benutzer nur mit der einen oder anderen Authentifizierungsmethode bei einer Datenbank anmelden, jedoch nicht mit beiden. Weitere Informationen zur IAM-Authentifizierung finden Sie unter IAM-Datenbankauthentifizierung.

Inhalt

- Übersicht über die Kerberos-Authentifizierung für DB-Cluster von Aurora MySQL
- Einschränkungen bei der Kerberos-Authentifizierung für Aurora MySQL

- Einrichten der Kerberos-Authentifizierung für DB-Cluster von Aurora MySQL
  - Schritt 1: Erstellen Sie ein Verzeichnis mit AWS Managed Microsoft AD
  - Schritt 2: (Optional) Erstellen einer Vertrauensstellung für ein On-Premise-Active-Directory
  - Schritt 3: Erstellen einer IAM-Rolle zur Verwendung durch Amazon Aurora
  - Schritt 4: Anlegen und Konfigurieren von Benutzern
  - Schritt 5: Erstellen oder Ändern eines DB-Clusters von Aurora MySQL
  - Schritt 6: Erstellen von Aurora-MySQL-Benutzern, die die Kerberos-Authentifizierung verwenden
    - Ändern einer vorhandenen Aurora-MySQL-Anmeldung
  - Schritt 7: Konfigurieren eines MySQL-Clients
  - Schritt 8: (Optional) Konfigurieren eines Benutzernamenvergleichs ohne Berücksichtigung der Groß-/Kleinschreibung
- Herstellen einer Verbindung mit Aurora MySQL mit Kerberos-Authentifizierung
  - Verwenden der Kerberos-Anmeldung von Aurora MySQL, um eine Verbindung mit dem DB-Cluster herzustellen
  - Kerberos-Authentifizierung mit globalen Aurora-Datenbanken
  - Migration von RDS für MySQL zu Aurora MySQL
  - Verhindern einer Zwischenspeicherung von Tickets
  - Protokollieren für die Kerberos-Authentifizierung
- Verwalten eines DB-Clusters in einer Domain
  - Grundlegendes zur Domänenmitgliedschaft

Übersicht über die Kerberos-Authentifizierung für DB-Cluster von Aurora MySQL

Wenn Sie die Kerberos-Authentifizierung für einen DB-Cluster von Aurora MySQL einrichten möchten, führen Sie die folgenden allgemeinen Schritte aus. Diese Schritte werden später ausführlich beschrieben.

1. Verwenden Sie AWS Managed Microsoft AD zum Erstellen eines AWS Managed Microsoft AD-Verzeichnisses. Zur Erstellung des Verzeichnisses können Sie AWS-Managementkonsole, AWS CLI oder Directory Service verwenden. Ausführliche Anweisungen dazu finden Sie unter Erstellen Ihres AWS Managed Microsoft AD-Verzeichnisses im AWS Directory Service-Administratorhandbuch.

2. Erstellen Sie eine AWS Identity and Access Management-(IAM)-Rolle, die die verwaltete IAM-Richtlinie `AmazonRDSDirectoryServiceAccess` verwendet. Die Rolle erlaubt Amazon Aurora, Aufrufe an Ihr Verzeichnis zu senden.

   Damit die Rolle Zugriff gewährt, muss der AWS Security Token Service (AWS STS)-Endpunkt in der AWS-Region für Ihr AWS-Konto aktiviert sein. AWS STS-Endpunkte sind standardmäßig in allen AWS-Regionen aktiv und Sie können sie ohne weitere Maßnahmen nutzen. Weitere Informationen finden Sie unter Aktivieren und Deaktivieren von AWS STS in einer AWS-Region im IAM-Benutzerhandbuch.
3. Erstellen und konfigurieren Sie Benutzer im Verzeichnis AWS Managed Microsoft AD mithilfe der Tools aus dem Microsoft Active Directory. Weitere Informationen zum Erstellen von Benutzern in Ihrem Active Directory finden Sie unter Verwalten von Benutzern und Gruppen in AWS Managed Microsoft AD im AWS Directory Service Administrationshandbuch.
4. Erstellen oder ändern Sie einen DB-Cluster von Aurora MySQL. Wenn Sie entweder die CLI oder die RDS-API für die Erstellungsanforderung verwenden, geben Sie eine Domänen-ID mit dem Parameter `Domain` an. Verwenden Sie die `d-*`-ID, die bei der Erstellung Ihres Verzeichnisses generiert wurde, und den Namen der von Ihnen erstellten IAM-Rolle.

   Wenn Sie einen vorhandenen DB-Cluster von Aurora MySQL so ändern, dass er die Kerberos-Authentifizierung verwendet, legen Sie die Parameter für die Domain und die IAM-Rolle für den DB-Cluster fest. Suchen Sie den DB-Cluster in derselben VPC wie das Domain-Verzeichnis.
5. Verwenden Sie die Hauptbenutzer-Anmeldeinformationen von Amazon RDS, um sich mit dem DB-Cluster von Aurora MySQL zu verbinden. Erstellen Sie den Datenbankbenutzer in Aurora MySQL gemäß den Anweisungen in Schritt 6: Erstellen von Aurora-MySQL-Benutzern, die die Kerberos-Authentifizierung verwenden.

   Benutzer, die Sie auf diese Weise anlegen, können sich mit der Kerberos-Authentifizierung beim DB-Cluster von Aurora MySQL anmelden. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit Aurora MySQL mit Kerberos-Authentifizierung.

Wenn Sie die Kerberos-Authentifizierung mit einem On-Premises oder einem selbst gehosteten Microsoft Active Directory verwenden möchten, erstellen Sie eine Gesamtstruktur-Vertrauensstellung. Eine Gesamtstruktur-Vertrauensstellung ist eine Vertrauensbeziehung zwischen zwei Gruppen von Domains. Die Vertrauensstellung kann uni- oder bidirektional sein. Weitere Informationen zur Einrichtung einer gesamtstrukturbasierten Vertrauensstellung mit Directory Service finden Sie unter ?Wann sollte eine Vertrauensstellung erstellt werden im AWS Directory Service-Administrationshandbuch.

Einschränkungen bei der Kerberos-Authentifizierung für Aurora MySQL

Die folgenden Einschränkungen gelten für die Kerberos-Authentifizierung für Aurora MySQL:

- Die Kerberos-Authentifizierung wird für Aurora MySQL Version 3.03 und höher unterstützt.

  Weitere Informationen zur AWS-Region-Unterstützung finden Sie unter Kerberos-Authentifizierung mit Aurora MySQL.
- Um die Kerberos-Authentifizierung mit Aurora MySQL zu verwenden, muss Ihr MySQL-Client oder Connector Version 8.0.26 oder höher auf Unix-Plattformen bzw. 8.0.27 oder höher unter Windows verwenden. Andernfalls ist das clientseitige `authentication_kerberos_client`-Plugin nicht verfügbar und Sie können sich nicht authentifizieren.
- Nur AWS Managed Microsoft AD wird in Aurora MySQL unterstützt. Sie können jedoch DB-Cluster von Aurora MySQL zu gemeinsam genutzten verwalteten Microsoft-AD-Domains zusammenführen, die verschiedenen Konten in derselben AWS-Region gehören.

  Außerdem können Sie ein eigenes On-Premises Active Directory verwenden. Weitere Informationen finden Sie unter Schritt 2: (Optional) Erstellen einer Vertrauensstellung für ein On-Premise-Active-Directory.
- Wenn Kerberos verwendet wird, um einen Benutzer zu authentifizieren, der sich von MySQL-Clients oder von Treibern auf dem Windows-Betriebssystem aus mit dem Aurora-MySQL-Cluster verbindet, muss die Groß-/Kleinschreibung des Datenbankbenutzernamens standardmäßig mit der Groß-/Kleinschreibung des Benutzers im Active Directory übereinstimmen. Wenn der Benutzer im Active Directory beispielsweise als `Admin` angezeigt wird, muss der Datenbankbenutzername `Admin` lauten.

  Mit dem Plug-in `authentication_kerberos` können Sie jetzt jedoch den Benutzernamenvergleich ohne Berücksichtigung der Groß-/Kleinschreibung verwenden. Weitere Informationen finden Sie unter Schritt 8: (Optional) Konfigurieren eines Benutzernamenvergleichs ohne Berücksichtigung der Groß-/Kleinschreibung.
- Sie müssen die Reader-DB-Instances neu starten, nachdem Sie die Funktion zur Installation des `authentication_kerberos`-Plugins aktiviert haben.
- Die Replikation auf DB-Instances, die das `authentication_kerberos`-Plugin nicht unterstützen, kann zu einem Replikationsfehler führen.
- Damit globale Aurora-Datenbanken die Kerberos-Authentifizierung verwenden können, müssen Sie diese für jeden DB-Cluster in der globalen Datenbank konfigurieren.

- Der Domain-Name muss weniger als 62 Zeichen lang sein.
- Ändern Sie den DB-Cluster-Port nicht, nachdem Sie die Kerberos-Authentifizierung aktiviert haben. Wenn Sie den Port ändern, funktioniert die Kerberos-Authentifizierung nicht mehr.

Einrichten der Kerberos-Authentifizierung für DB-Cluster von Aurora MySQL

Wird verwendet AWS Managed Microsoft AD , um die Kerberos-Authentifizierung für einen Aurora MySQL-DB-Cluster einzurichten. Um die Kerberos-Authentifizierung einzurichten, führen Sie die folgenden Schritte aus.

Themen

- Schritt 1: Erstellen Sie ein Verzeichnis mit AWS Managed Microsoft AD
- Schritt 2: (Optional) Erstellen einer Vertrauensstellung für ein On-Premise-Active-Directory
- Schritt 3: Erstellen einer IAM-Rolle zur Verwendung durch Amazon Aurora
- Schritt 4: Anlegen und Konfigurieren von Benutzern
- Schritt 5: Erstellen oder Ändern eines DB-Clusters von Aurora MySQL
- Schritt 6: Erstellen von Aurora-MySQL-Benutzern, die die Kerberos-Authentifizierung verwenden
- Schritt 7: Konfigurieren eines MySQL-Clients
- Schritt 8: (Optional) Konfigurieren eines Benutzernamenvergleichs ohne Berücksichtigung der Groß-/Kleinschreibung

Schritt 1: Erstellen Sie ein Verzeichnis mit AWS Managed Microsoft AD

Directory Service erstellt ein vollständig verwaltetes Active Directory in der AWS Cloud. Wenn Sie ein AWS Managed Microsoft AD Verzeichnis erstellen, Directory Service erstellt in Ihrem Namen zwei Domänencontroller und DNS-Server (Domain Name System). Die Verzeichnisserver werden in verschiedenen Subnetzen in einer VPC erstellt. Diese Redundanz trägt dazu bei, dass Ihr Verzeichnis auch im Fehlerfall erreichbar bleibt.

Wenn Sie ein AWS Managed Microsoft AD Verzeichnis erstellen, Directory Service führt er in Ihrem Namen die folgenden Aufgaben aus:

- Einrichten eines Active Directory innerhalb der VPC.
- Erstellt ein Konto für den Verzeichnisadministrator mit dem Benutzernamen `Admin` und dem angegebenen Passwort. Mit diesem Konto verwalten Sie das Verzeichnis.

> **Note**
>
> Achten Sie darauf, dieses Passwort zu speichern. Directory Service speichert es nicht. Sie können es zurücksetzen, aber Sie können es nicht abrufen.

- Erstellt eine Sicherheitsgruppe für die Verzeichniscontroller.

Wenn Sie eine starten AWS Managed Microsoft AD, AWS erstellt eine Organisationseinheit (OU), die alle Objekte Ihres Verzeichnisses enthält. Diese OU erhält den NetBIOS-Namen, den Sie beim Erstellen des Verzeichnisses eingegeben haben. Es befindet sich im Domänenstamm, der Eigentümer ist und von diesem verwaltet wird AWS.

Das `Admin` Konto, das mit Ihrem AWS Managed Microsoft AD Verzeichnis erstellt wurde, verfügt über Berechtigungen für die gängigsten Verwaltungsaktivitäten Ihrer Organisationseinheit, darunter:

- Erstellen, Aktualisieren oder Löschen von Benutzern
- Hinzufügen von Ressourcen zu Ihrer Domain, etwa Datei- oder Druckserver, und anschließendes Gewähren der zugehörigen Ressourcenberechtigungen für Benutzer in der OU
- Zusätzliche OUs Container erstellen
- Delegieren von Befugnissen
- Wiederherstellen von gelöschten Objekten aus dem Active Directory-Papierkorb
- Führen Sie AD- und PowerShell DNS-Windows-Module im Active Directory-Webdienst aus

Das `Admin`-Konto hat auch die Berechtigung, die folgenden domänenweiten Aktivitäten durchzuführen:

- Verwalten von DNS-Konfigurationen (Hinzufügen, Entfernen oder Aktualisieren von Datensätzen, Zonen und Weiterleitungen)
- Aufrufen von DNS-Ereignisprotokollen
- Anzeigen von Sicherheitsereignisprotokollen

Um ein Verzeichnis zu erstellen mit AWS Managed Microsoft AD

1. Melden Sie sich bei an AWS-Managementkonsole und öffnen Sie die Directory Service Konsole unter https://console.aws.amazon.com/directoryservicev2/.

2. Wählen Sie im Navigationsbereich Directories (Verzeichnisse) aus. Wählen Sie denn Set up Directory (Verzeichnis einrichten) aus.
3. Wähle AWS Managed Microsoft AD. AWS Managed Microsoft AD ist die einzige Option, die Sie derzeit mit Amazon RDS verwenden können.
4. Geben Sie die folgenden Informationen ein:

   DNS-Name des Verzeichnisses

   Den vollständig qualifizierten Namen für das Verzeichnis, z. B. **`corp.example.com`**.

   NetBIOS-Name des Verzeichnisses

   Die kurzen Namen für das Verzeichnis, z. B. **`CORP`**.

   Verzeichnisbeschreibung

   (Optional) Eine Beschreibung für das Verzeichnis.

   Administratorpasswort

   Das Passwort für den Verzeichnisadministrator. Während des Verzeichniserstellungsprozesses wird ein Administratorkonto mit dem Benutzernamen Admin und diesem Passwort angelegt.

   Das Passwort für den Verzeichnisadministrator das nicht das Wort "admin" enthalten. Beachten Sie beim Passwort die Groß- und Kleinschreibung und es muss 8 bis 64 Zeichen lang sein. Zudem muss es mindestens ein Zeichen aus dreien der vier folgenden Kategorien enthalten:

   - Kleinbuchstaben (a–z)
   - Großbuchstaben (A–Z)
   - Zahlen (0–9)
   - Nicht-alphanumerische Zeichen (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

   Passwort bestätigen

   Das wiederholte Administrator-Passwort.
5. Wählen Sie Weiter aus.
6. Geben Sie die folgenden Informationen in den Abschnitt Networking ein. Wählen Sie dann Next (Weiter) aus:

VPC

Die VPC für das Verzeichnis. Erstellen Sie den DB-Cluster von Aurora MySQL in derselben VPC.

Subnets

Subnetze für die Verzeichnisserver. Die beiden Subnetze müssen zu verschiedenen Availability-Zonen gehören.

7. Prüfen Sie die Verzeichnisinformationen und nehmen Sie ggf. Änderungen vor. Wenn die Informationen richtig sind, wählen Sie Create directory (Verzeichnis erstellen).
Verzeichnis-Detailseite während der Erstellung

Es dauert einige Minuten, bis das Verzeichnis erstellt wird. Wenn es erfolgreich erstellt wurde, ändert sich der Wert Status in Active (Aktiv).

Um Informationen über Ihr Verzeichnis anzuzeigen, wählen Sie den Verzeichnisnamen in der Verzeichnisliste aus. Notieren Sie sich den Wert Verzeichnis-ID. Sie benötigen diesen Wert, wenn Sie Ihren DB-Cluster von Aurora MySQL erstellen oder ändern.
Verzeichnis-ID auf der Seite mit den Verzeichnisdetails

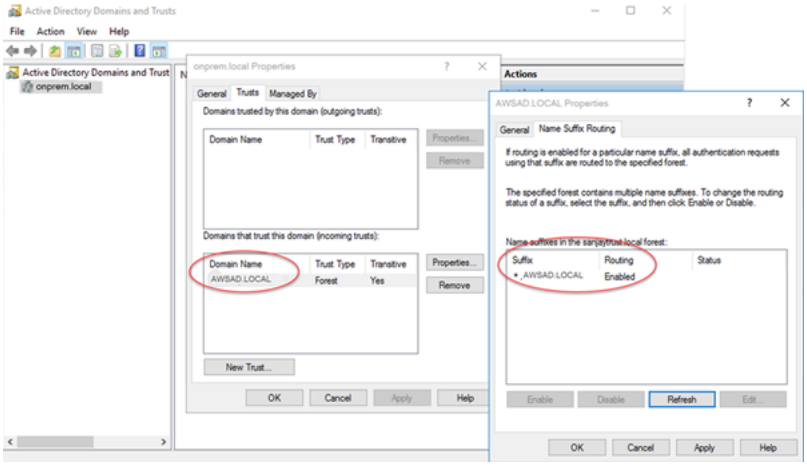
Schritt 2: (Optional) Erstellen einer Vertrauensstellung für ein On-Premise-Active-Directory

Wenn Sie Ihr eigenes lokales Microsoft Active Directory nicht verwenden möchten, fahren Sie mit for Schritt 3: Erstellen einer IAM-Rolle zur Verwendung durch Amazon Aurora.

Um die Kerberos-Authentifizierung mit Ihrem lokalen Active Directory zu verwenden, müssen Sie eine vertrauensvolle Domänenbeziehung mithilfe einer Gesamtvertrauensstellung zwischen Ihrem lokalen Microsoft Active Directory und dem AWS Managed Microsoft AD Verzeichnis (erstellt in) einrichten. Schritt 1: Erstellen Sie ein Verzeichnis mit AWS Managed Microsoft AD Die Vertrauensstellung kann einseitig erfolgen, wobei das AWS Managed Microsoft AD -Verzeichnis dem lokalen Microsoft Active Directory vertraut. Die Vertrauensstellung kann auch bidirektional erfolgen, wobei beide Active Directories einander vertrauen. Weitere Informationen zum Einrichten von Vertrauensstellungen mithilfe von Directory Service finden Sie unter Wann sollte eine Vertrauensstellung eingerichtet werden? im Administratorhandbuch.AWS Directory Service

> **Note**
>
> Verwendung eines lokalen Microsoft Active Directory:
>
> - Windows-Clients können keine Verbindung über benutzerdefinierten Aurora-Endpunkte herstellen. Weitere Informationen hierzu finden Sie unter Amazon-Aurora-Endpunktverbindungen.
> - Globale Datenbanken:
>   - Windows-Clients können sich nur über Instance- oder Cluster-Endpunkte in der primären AWS-Region der globalen Datenbank verbinden.
>   - Windows-Clients können über sekundäre Clusterendpunkte keine Verbindung herstellen. AWS-Regionen

Stellen Sie sicher, dass der lokale Microsoft Active Directory-Domänenname ein DNS-Suffix-Routing enthält, das der neu erstellten Vertrauensstellung entspricht. Im folgenden Screenshot wird ein Beispiel gezeigt.



Schritt 3: Erstellen einer IAM-Rolle zur Verwendung durch Amazon Aurora

Damit Amazon Aurora für Sie anrufen Directory Service kann, benötigen Sie eine AWS Identity and Access Management (IAM-) Rolle, die die verwaltete IAM-Richtlinie verwendet. `AmazonRDSDirectoryServiceAccess` Diese Rolle ermöglicht es Aurora, Aufrufe an Directory Service durchzuführen.

Wenn Sie mit dem AWS-Managementkonsole einen DB-Cluster erstellen und Sie die `iam:CreateRole` entsprechende Berechtigung haben, erstellt die Konsole diese Rolle

automatisch. In diesem Fall lautet der Rollenname `rds-directoryservice-kerberos-access-role`. Andernfalls müssen Sie die IAM-Rolle manuell erstellen. Wenn Sie diese IAM-Rolle erstellen`Directory Service`, wählen Sie die AWS verwaltete Richtlinie aus und fügen Sie sie `AmazonRDSDirectoryServiceAccess` an.

Weitere Informationen zum Erstellen von IAM-Rollen für einen Dienst finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS Dienst im IAM-Benutzerhandbuch.

Optional können Sie Richtlinien mit den erforderlichen Berechtigungen erstellen, anstatt die verwaltete IAM-Richtlinie zu verwende `AmazonRDSDirectoryServiceAccess`. In diesem Fall muss die IAM-Rolle die folgende IAM-Vertrauensrichtlinie haben.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Die Rolle muss auch über die folgende IAM-Rollenrichtlinie verfügen.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Schritt 4: Anlegen und Konfigurieren von Benutzern

Sie können Benutzer mit dem Tool "Active Directory-Benutzer und -Computer" erstellen. Dieses Tool ist Teil der Tools Active Directory Domain Services und Active Directory Lightweight Directory Services. „Benutzer“ sind Einzelpersonen oder Entitäten, die Zugriff auf Ihr Verzeichnis haben.

Um Benutzer in einem Directory Service Verzeichnis zu erstellen, verwenden Sie eine lokale oder EC2 Amazon-Instance, die auf Microsoft Windows basiert und mit Ihrem Directory Service Verzeichnis verknüpft ist. Gleichzeitig müssen Sie bei der Instance als Benutzer angemeldet sein, der über Berechtigungen zum Erstellen von Benutzern verfügt. Weitere Informationen finden Sie unter Verwalten von Benutzern und Gruppen AWS Managed Microsoft AD im AWS Directory-Service-Administrationshandbuch.

Schritt 5: Erstellen oder Ändern eines DB-Clusters von Aurora MySQL

Erstellen oder ändern Sie einen DB-Cluster von Aurora MySQL zur Verwendung mit Ihrem Verzeichnis. Sie können die Konsole oder die RDS-API verwenden AWS CLI, um einen DB-Cluster einem Verzeichnis zuzuordnen. Sie können diese Aufgabe mit einer der folgenden Methoden durchführen:

- Erstellen Sie mit der Konsole, dem create-db-clusterCLI-Befehl oder dem Vorgang Create DBCluster RDS API einen neuen Aurora MySQL-DB-Cluster.

  Detaillierte Anweisungen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.
- Ändern Sie einen vorhandenen Aurora MySQL-DB-Cluster mithilfe der Konsole, des modify-db-clusterCLI-Befehls oder der Operation Modify DBCluster RDS API.

  Detaillierte Anweisungen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

- Stellen Sie einen Aurora MySQL-DB-Cluster aus einem DB-Snapshot mithilfe der Konsole, des restore-db-cluster-fromCLI-Befehls -snapshot oder der Operation Restore DBCluster FromSnapshot RDS API wieder her.

  Detaillierte Anweisungen finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.
- Stellen Sie einen Aurora MySQL-DB-Cluster point-in-time mithilfe der Konsole, des Befehls restore-db-cluster-to- point-in-time CLI oder der Operation Restore DBCluster ToPointInTime RDS API auf einem wieder her.

  Detaillierte Anweisungen finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit.

Die Kerberos-Authentifizierung wird nur für DB-Cluster von Aurora MySQL in einer VPC unterstützt. Der DB-Cluster kann sich in derselben VPC wie das Verzeichnis oder in einer anderen VPC befinden. Die VPC des DB-Clusters muss über eine VPC-Sicherheitsgruppe verfügen, die ausgehende Kommunikation mit Ihrem Verzeichnis zulässt.

Konsole

Wenn Sie die Konsole zum Erstellen, Ändern oder Wiederherstellen eines DB-Clusters verwenden, wählen Sie Kerberos-Authentifizierung im Datenbank-Authentifizierung-Abschnitt. Wählen Sie Verzeichnis durchsuchen und dann das Verzeichnis aus, oder klicken Sie auf Neues Verzeichnis erstellen.
Kerberos-Authentifizierungseinstellung beim Erstellen eines DB-Clusters

AWS CLI

Wenn Sie die AWS CLI oder die RDS-API verwenden, verknüpfen Sie einen DB-Cluster mit einem Verzeichnis. Die folgenden Parameter sind erforderlich, damit der DB-Cluster das von Ihnen erstellte Domain-Verzeichnis verwendet:

- Für den `--domain`-Parameter verwenden Sie den Domänenbezeichner („d-*“-Bezeichner), der beim Erstellen des Verzeichnisses generiert wurde.
- Verwenden Sie für den `--domain-iam-role-name`-Parameter die von Ihnen erstellte Rolle, die die verwaltete IAM-Richtlinie `AmazonRDSDirectoryServiceAccess` verwendet.

Beispielsweise ändert der folgende CLI-Befehl einen DB-Cluster so, dass er ein Verzeichnis verwendet.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --domain d-ID \
    --domain-iam-role-name role-name
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

> **Important**
>
> Wenn Sie einen DB-Cluster ändern, um die Kerberos-Authentifizierung zu aktivieren, starten Sie die Reader-DB-Instances neu, nachdem Sie die Änderung vorgenommen haben.

Schritt 6: Erstellen von Aurora-MySQL-Benutzern, die die Kerberos-Authentifizierung verwenden

Der DB-Cluster ist mit der AWS Managed Microsoft AD Domäne verbunden. Auf diese Weise können Sie Aurora-MySQL-Benutzer aus den Active-Directory-Benutzern Ihrer Domain bereitstellen. Die Datenbankberechtigungen werden durch Standardberechtigungen von Aurora MySQL verwaltet, die diesen Benutzern gewährt und entzogen werden.

Sie können einem Active-Directory-Benutzer erlauben, sich bei Aurora MySQL zu authentifizieren. Dazu verwenden Sie zunächst die Hauptbenutzer-Anmeldeinformationen von Amazon RDS, um sich mit der DB-Instance von Aurora MySQL sowie mit jedem anderen DB-Cluster zu verbinden. Nachdem Sie angemeldet sind, erstellen Sie einen extern authentifizierten Benutzer mit Kerberos-Authentifizierung in Aurora MySQL wie folgt:

```
CREATE USER user_name@'host_name' IDENTIFIED WITH 'authentication_kerberos' BY
 'realm_name';
```

- Ersetzen Sie *user_name* durch den Benutzernamen. Benutzer (sowohl Menschen als auch Anwendungen) aus Ihrer Domain können sich nun über einen der Domain beigetretenen Client per Kerberos-Authentifizierung mit dem DB-Cluster verbinden.
- Ersetzen Sie *host_name* durch den Hostnamen. Sie können `%` als Platzhalter verwenden. Sie können auch bestimmte IP-Adressen für den Hostnamen verwenden.
- *realm_name*Ersetzen Sie es durch den Namen des Verzeichnisbereichs der Domäne. Der Bereichsname entspricht normalerweise dem DNS-Domain-Namen in Großbuchstaben, z. B. `CORP.EXAMPLE.COM`. Ein Bereich ist eine Gruppe von Systemen, die dasselbe Kerberos Key Distribution Center verwenden.

Im folgenden Beispiel wird ein Datenbankbenutzer mit dem Namen `Admin` erstellt, der sich gegenüber dem Active Directory mit dem Bereichsnamen `MYSQL.LOCAL` authentifiziert.

```
CREATE USER Admin@'%' IDENTIFIED WITH 'authentication_kerberos' BY 'MYSQL.LOCAL';
```

Ändern einer vorhandenen Aurora-MySQL-Anmeldung

Sie können auch eine vorhandene Aurora-MySQL-Anmeldung ändern, um die Kerberos-Authentifizierung zu verwenden. Nutzen Sie dazu die folgende Syntax:

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

Schritt 7: Konfigurieren eines MySQL-Clients

Gehen Sie folgendermaßen vor, um einen MySQL-Client zu konfigurieren:

1. Erstellen Sie eine `krb5.conf`-Datei (oder eine vergleichbare Datei), um auf die Domain zu verweisen.
2. Stellen Sie sicher, dass der Datenverkehr zwischen dem Client-Host und fließen kann Directory Service. Verwenden Sie ein Netzwerk-Dienstprogramm wie Netcat für die folgenden Aufgaben:
   - Überprüfen Sie den Datenverkehr über DNS für Port 53.
   - Überprüfen Sie den Datenverkehr TCP/UDP für Port 53 und für Kerberos, einschließlich der Ports 88 und 464 für. Directory Service
3. Stellen Sie sicher, dass der Datenverkehr zwischen dem Client-Host und der DB-Instance über den Datenbank-Port fließen kann. Verwenden Sie beispielsweise `mysql`, um eine Verbindung herzustellen und auf die Datenbank zuzugreifen.

Im Folgenden finden Sie `krb5.conf` Beispielinhalte für. AWS Managed Microsoft AD

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

Nachfolgend ist ein Beispiel für den Inhalt von `krb5.conf` für ein On-Premises Microsoft Active Directory aufgeführt.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

Schritt 8: (Optional) Konfigurieren eines Benutzernamenvergleichs ohne Berücksichtigung der Groß-/Kleinschreibung

Standardmäßig muss die Groß-/Kleinschreibung des MySQL-Datenbankbenutzernamens mit der Groß-/Kleinschreibung der Active-Directory-Anmeldung übereinstimmen. Mit dem

Plug-in `authentication_kerberos` können Sie jetzt jedoch den Benutzernamenvergleich ohne Berücksichtigung der Groß-/Kleinschreibung verwenden. Setzen Sie dazu den Parameter `authentication_kerberos_caseins_cmp` des DB-Clusters auf `true`.

So verwenden Sie den Benutzernamenvergleich ohne Berücksichtigung der Groß-/Kleinschreibung

1. Erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe. Folgen Sie den Verfahren in Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora.
2. Bearbeiten Sie die neue Parametergruppe, um den Wert für `authentication_kerberos_caseins_cmp` auf `true` zu setzen. Folgen Sie den Verfahren in Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.
3. Ordnen Sie die DB-Clusterparametergruppe dem DB-Cluster von Aurora MySQL zu. Folgen Sie den Verfahren in Zuordnen einer DB-Cluster-Parametergruppe zu einem DB-Cluster.
4. Starten Sie den DB-Cluster neu.

Herstellen einer Verbindung mit Aurora MySQL mit Kerberos-Authentifizierung

Um Fehler zu vermeiden, verwenden Sie einen MySQL-Client mit Version 8.0.26 oder höher auf Unix-Plattformen, 8.0.27 oder höher unter Windows.

Verwenden der Kerberos-Anmeldung von Aurora MySQL, um eine Verbindung mit dem DB-Cluster herzustellen

Wenn Sie eine Verbindung mit Aurora MySQL mit Kerberos-Authentifizierung herstellen möchten, melden Sie sich als Datenbankbenutzer an, den Sie gemäß den Anweisungen unter Schritt 6: Erstellen von Aurora-MySQL-Benutzern, die die Kerberos-Authentifizierung verwenden erstellt haben.

Stellen Sie über die Eingabeaufforderung eine Verbindung mit einem der Endpunkte her, die mit Ihrem DB-Cluster von Aurora MySQL verbunden sind. Wenn Sie zur Eingabe des Passworts aufgefordert werden, geben Sie das mit diesem Benutzernamen verknüpfte Kerberos-Passwort ein.

Wenn Sie sich bei Kerberos authentifizieren, wird ein Ticket-Granting Ticket (TGT) generiert, falls noch keines existiert. Das `authentication_kerberos`-Plugin verwendet das TGT, um ein Serviceticket zu erhalten, das dann dem Aurora-MySQL-Datenbankserver vorgelegt wird.

Sie können den MySQL-Client verwenden, um sich mit Kerberos-Authentifizierung unter Windows oder Unix mit Aurora MySQL zu verbinden.

Unix

Sie können die Verbindung mit einer der folgenden Methoden herstellen:

- Rufen Sie das TGT manuell ab. In diesem Fall müssen Sie dem MySQL-Client das Passwort nicht nennen.
- Stellen Sie das Passwort für die Active-Directory-Anmeldung direkt dem MySQL-Client zur Verfügung.

Das clientseitige Plugin wird auf Unix-Plattformen für MySQL-Client-Versionen ab 8.0.26 unterstützt.

So stellen Sie eine Verbindung her, indem Sie das TGT manuell abrufen

1. Verwenden Sie an der Befehlszeilenschnittstelle den folgenden Befehl, um das TGT abzurufen.

   ```
   kinit user_name
   ```

2. Verwenden Sie den folgenden `mysql`-Befehl, um sich beim DB-Instance-Endpunkt Ihres DB-Clusters anzumelden.

   ```
   mysql -h DB_instance_endpoint -P 3306 -u user_name -p
   ```

   > **Note**
   >
   > Die Authentifizierung kann fehlschlagen, wenn der Keytab auf der DB-Instance rotiert wird. Rufen Sie in diesem Fall ein neues TGT ab, indem Sie `kinit` erneut ausführen.

So stellen Sie eine direkte Verbindung her

1. Verwenden Sie an der Befehlszeilenschnittstelle den folgenden `mysql`-Befehl, um sich beim DB-Instance-Endpunkt Ihres DB-Clusters anzumelden.

   ```
   mysql -h DB_instance_endpoint -P 3306 -u user_name -p
   ```

2. Geben Sie das Passwort für den Active-Directory-Benutzer an.

Windows

Unter Windows erfolgt die Authentifizierung normalerweise bei der Anmeldung, sodass Sie das TGT nicht manuell abrufen müssen, um eine Verbindung mit dem DB-Cluster von Aurora MySQL herzustellen. Die Groß- und Kleinschreibung des Datenbankbenutzernamens muss mit der Groß-/Kleinschreibung des Benutzers im Active Directory übereinstimmen. Wenn der Benutzer im Active Directory beispielsweise als `Admin` angezeigt wird, muss der Datenbankbenutzername `Admin` lauten.

Das clientseitige Plugin wird unter Windows für MySQL-Client-Versionen ab 8.0.27 unterstützt.

So stellen Sie eine direkte Verbindung her

- Verwenden Sie an der Befehlszeilenschnittstelle den folgenden `mysql`-Befehl, um sich beim DB-Instance-Endpunkt Ihres DB-Clusters anzumelden.

  ```
  mysql -h DB_instance_endpoint -P 3306 -u user_name
  ```

Kerberos-Authentifizierung mit globalen Aurora-Datenbanken

Die Kerberos-Authentifizierung für Aurora MySQL wird für globale Aurora-Datenbanken unterstützt. Wenn Sie Benutzer im sekundären DB-Cluster mithilfe des Active Directory des primären DB-Clusters authentifizieren möchten, replizieren Sie das Active Directory auf die sekundäre AWS-Region. Sie aktivieren die Kerberos-Authentifizierung auf dem sekundären Cluster mit derselben Domain-ID wie für den primären Cluster. Die AWS Managed Microsoft AD-Replikation wird nur mit der Enterprise-Version von Active Directory unterstützt. Weitere Informationen finden Sie unter Regionsübergreifende Replikation im AWS Directory Service-Administratorhandbuch.

Migration von RDS für MySQL zu Aurora MySQL

Nachdem Sie von RDS für MySQL mit aktivierter Kerberos-Authentifizierung zu Aurora MySQL migriert haben, ändern Sie Benutzer, die mit dem `auth_pam`-Plugin erstellt wurden, so, dass sie das `authentication_kerberos`-Plugin verwenden. Zum Beispiel:

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

Verhindern einer Zwischenspeicherung von Tickets

Wenn beim Start der MySQL-Client-Anwendung kein gültiges TGT vorhanden ist, kann die Anwendung das TGT abrufen und zwischenspeichern. Wenn Sie verhindern möchten, dass das TGT

zwischengespeichert wird, legen Sie in der Datei `/etc/krb5.conf` einen Konfigurationsparameter fest.

> **Note**
>
> Diese Konfiguration gilt nur für Client-Hosts, auf denen Unix ausgeführt wird, nicht für Windows.

So verhindern Sie das Zwischenspeichern von TGTs

- Fügen Sie `/etc/krb5.conf` wie folgt einen `[appdefaults]`-Abschnitt hinzu:

```
[appdefaults]
  mysql = {
    destroy_tickets = true
  }
```

Protokollieren für die Kerberos-Authentifizierung

Die Umgebungsvariable `AUTHENTICATION_KERBEROS_CLIENT_LOG` legt die Protokollierungsebene für die Kerberos-Authentifizierung fest. Sie können die Protokolle für das clientseitige Debugging verwenden.

Die zulässigen Werte sind 1–5. Protokollmeldungen werden in die Standardfehlerausgabe geschrieben. In der folgenden Tabelle ist jede Protokollierungsebene beschrieben.

Protokollierungsstufe	Beschreibung
1 oder nicht festgelegt	Keine Protokollierung
2	Fehlermeldungen
3	Fehler- und Warnmeldungen
4	Fehler-, Warn- und Informationsmeldungen
5	Fehler-, Warn-, Informations- und Debug-Meldungen

Verwalten eines DB-Clusters in einer Domain

Sie können die AWS CLI oder die RDS-API verwenden, um Ihren DB-Cluster und seine Beziehung zu Ihrem verwalteten Active Directory zu verwalten. Sie können z. B. ein Active Directory für die Kerberos-Authentifizierung zuordnen und ein Active Directory trennen, um die Kerberos-Authentifizierung zu deaktivieren. Sie können auch einen DB-Cluster, der extern von einem Active Directory authentifiziert werden soll, in ein anderes Active Directory verschieben.

Sie können z. B. mithilfe der Amazon RDS-API Folgendes tun:

- Um erneut zu versuchen, die Kerberos-Authentifizierung für eine fehlgeschlagene Mitgliedschaft zu aktivieren, verwenden Sie die API-Operation `ModifyDBInstance` und geben Sie die Verzeichnis-ID der aktuellen Mitgliedschaft an.
- Um den IAM-Rollennamen für die Mitgliedschaft zu aktualisieren, verwenden Sie die `ModifyDBInstance`-API-Operation und geben Sie die Verzeichnis-ID der aktuellen Mitgliedschaft und die neue IAM-Rolle an.
- Wenn Sie die Kerberos-Authentifizierung in einem DB-Cluster deaktivieren möchten, verwenden Sie die API-Operation `ModifyDBInstance` und geben Sie `none` als Domain-Parameter an.
- Um einen DB-Cluster von einer Domain in eine andere zu verschieben, verwenden Sie die API-Operation `ModifyDBInstance`. Geben Sie die Domain-ID der neuen Domain als Domain-Parameter an.
- Wenn Sie die Mitgliedschaft für jeden DB-Cluster auflisten möchten, verwenden Sie die API-Operation `DescribeDBInstances`.

Grundlegendes zur Domänenmitgliedschaft

Nachdem Sie Ihren DB-Cluster erstellt oder geändert haben, wird er Mitglied der Domain. Sie können den Status der Domain-Mitgliedschaft für den DB-Cluster anzeigen, indem Sie den CLI-Befehl describe-db-clusters verwenden. Der Status des DB-Clusters kann einer der folgenden sein:

- `kerberos-enabled`: Für den DB-Cluster ist die Kerberos-Authentifizierung aktiviert.
- `enabling-kerberos`: AWS ist dabei, die Kerberos-Authentifizierung auf diesem DB-Cluster zu aktivieren.
- `pending-enable-kerberos`: Das Aktivieren der Kerberos-Authentifizierung in diesem DB-Cluster steht aus.

- `pending-maintenance-enable-kerberos`: AWS versucht, die Kerberos-Authentifizierung auf dem DB-Cluster während des nächsten geplanten Wartungsfensters zu aktivieren.
- `pending-disable-kerberos`: Das Deaktivieren der Kerberos-Authentifizierung in diesem DB-Cluster steht aus.
- `pending-maintenance-disable-kerberos`: AWS versucht, die Kerberos-Authentifizierung auf dem DB-Cluster während des nächsten geplanten Wartungsfensters zu deaktivieren.
- `enable-kerberos-failed`: Ein Konfigurationsproblem hat AWS daran gehindert, die Kerberos-Authentifizierung auf dem DB-Cluster zu aktivieren. Überprüfen und korrigieren Sie Ihre Konfiguration, bevor Sie den Befehl zum Ändern des DB-Clusters erneut ausführen.
- `disabling-kerberos`: AWS ist dabei, die Kerberos-Authentifizierung auf diesem DB-Cluster zu deaktivieren.

Eine Anfrage zur Aktivierung der Kerberos-Authentifizierung kann wegen eines Netzwerkverbindungsproblems oder einer falschen IAM-Rolle fehlschlagen. Angenommen, Sie erstellen einen DB-Cluster oder ändern einen vorhandenen DB-Cluster und der Versuch, die Kerberos-Authentifizierung zu aktivieren, schlägt fehl. Wenn dies geschieht, führen Sie den Befehl zum Ändern erneut aus oder ändern Sie den neu erstellte DB-Cluster, um der Domain beizutreten.

Migrieren von Daten zu einem Amazon Aurora MySQL-DB-Cluster

Sie haben mehrere Möglichkeiten, Daten aus einer vorhandenen Datenbank in einen Amazon Aurora MySQL-DB-Cluster zu migrieren. Die verfügbaren Migrationsoptionen sind auch von der Quelldatenbank und der Menge der zu migrierenden Daten abhängig.

Es gibt zwei verschiedene Migrationstypen: physisch und logisch. Bei der physischen Migration werden physische Kopien von Datenbankdateien verwenden, um die Datenbank zu migrieren. Bei der logischen Migration erfolgt die Migration durch Anwendung logischer Datenbankänderungen, wie beispielsweise Einfügungen, Aktualisierungen und Löschungen.

Die physische Migration hat die folgenden Vorteile:

- Die physische Migration ist schneller als die logische Migration, insbesondere für große Datenbanken.
- Die Datenbankleistung leidet nicht, wenn für eine physische Migration eine Sicherung durchgeführt wird.
- Die physische Migration kann alles in der Quelldatenbank migrieren, auch komplexe Datenbankkomponenten.

Die physische Migration hat die folgenden Einschränkungen:

- Der Parameter `innodb_page_size` muss auf seinen Standardwert (16KB) gesetzt sein.
- Der `innodb_data_file_path`-Parameter darf nur mit einer Datendatei konfiguriert werden, die den Standarddateinamen `"ibdata1:12M:autoextend"` verwendet. Datenbanken mit zwei Datendateien oder nur einer Datendatei eines anderen Namens können mit dieser Methode nicht migriert werden.

  Nachfolgend finden Sie Beispiele für unzulässige Dateinamen: `"innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend"` und `"innodb_data_file_path=ibdata01:50M:autoextend"`.
- Der Parameter `innodb_log_files_in_group` muss auf seinen Standardwert (2) gesetzt sein.

Die logische Migration hat die folgenden Vorteile:

- Sie können Untermengen der Datenbank migrieren, beispielsweise bestimmte Tabellen oder Teile einer Tabelle.

- Die Daten können unabhängig von der physischen Speicherstruktur migriert werden.

Die logische Migration hat die folgenden Einschränkungen:

- Die logische Migration ist normalerweise langsamer als die physische Migration.
- Komplexe Datenbankkomponenten können den logischen Migrationsprozess verlangsamen. In einigen Fällen können komplexe Datenbankkomponenten den logischen Migrationsprozess sogar blockieren.

In der folgenden Tabelle sind Ihre Optionen und die Migrationstypen für jede Option aufgelistet.

Migration von	Migrationstyp	Lösung
Eine RDS für MySQL-DB-Instance	Physisch	Sie können Daten aus einer RDS für MySQL-DB-Instance migrieren, indem Sie zunächst eine Aurora MySQL-Read Replica einer MySQL-DB-Instance erstellen. Wenn die Replica-Verzögerung zwischen der MySQL-DB-Instance und der Aurora MySQL-Read Replica 0 beträgt, können Sie Ihre Client-Anwendungen anweisen, aus der Aurora Read Replica zu lesen und dann die Replikation stoppen, um die Aurora MySQL Read Replica in ein eigenständiges Aurora MySQL-DB-Cluster für das Lesen und Schreiben zu transformieren. Details hierzu finden Sie unter Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat).
Ein RDS für MySQL-DB-Snapshot	Physisch	Sie können Daten direkt aus einem RDS für MySQL-DB-Snapshot in einen Amazon Aurora MySQL-DB-Cluster migrieren. Details hierzu finden Sie unter Migrieren eines RDS für MySQL-Snapshots zu Aurora.
einer MySQL-Datenbank außerhalb von Amazon RDS	Logisch	Sie können mithilfe des Dienstprogramms `mysqldump` eine Dump-Datei mit den Daten erstellen und diese dann in einen bestehenden Amazon Aurora MySQL-DB-Cluster importieren. Details hierzu finden Sie unter Logische

Migration von	Migrationstyp	Lösung
		Migration von MySQL zu Amazon Aurora MySQL mithilfe von mysqldump. Um Metadaten für Datenbankbenutzer während der Migration aus einer externen MySQL-Datenbank zu exportieren, können Sie auch einen MySQL-Shell-Befehl anstelle von `mysqldump` verwenden. Weitere Informationen finden Sie unter Dienstprogramme für Instance-, Schema- und Tabellen-Dumps. **Note** Das mysqlpump-Dienstprogramm ist seit MySQL 8.0.34 veraltet.
einer MySQL-Datenbank außerhalb von Amazon RDS	Physisch	Sie können die Sicherungsdateien aus Ihrer Datenbank in einen Amazon Simple Storage Service (Amazon S3)-Bucket kopieren und dann einen Amazon Aurora MySQL-DB-Cluster aus diesen Dateien wiederherstellen. Dieser Weg ist bedeutend schneller als eine Datenmigration mit `mysqldump` . Details hierzu finden Sie unter Physische Migration von MySQL mithilfe von Percona XtraBackup und Amazon S3.

Migration von	Migrationstyp	Lösung
einer MySQL-Datenbank außerhalb von Amazon RDS	Logisch	Sie können die Daten in der Datenbank als Textdateien speichern und diese in einen Amazon S3-Bucket kopieren. Anschließend können die Daten in einen vorhandenen Aurora MySQL-DB-Cluster laden, indem Sie den MySQL-Befehl `LOAD DATA FROM S3` verwenden. Weitere Informationen finden Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket.
Eine Datenbank, die nicht MySQL-kompatibel ist	Logisch	Sie können AWS Database Migration Service (AWS DMS) verwenden, um Daten aus einer Datenbank zu migrieren , die nicht MySQL-kompatibel ist. Weitere Informationen zu AWS DMS finden Sie unter Was ist AWSDatabase Migration Service?

Note

Wenn Sie eine MySQL-Datenbank extern zu Amazon RDS migrieren, werden die in der Tabelle beschriebenen Migrationsoptionen nur unterstützt, wenn Ihre Datenbank die InnoDB- oder MyISAM-Tabellenbereiche unterstützt.
Wenn die MySQL-Datenbank, die Sie zu Aurora MySQL migrieren, `memcached` verwendet, müssen Sie `memcached` vor der Migration entfernen.
Sie können von einigen älteren Versionen von MySQL 8.0, einschließlich 8.0.11, 8.0.13 und 8.0.15, nicht zu Aurora MySQL Version 3.05 und höher migrieren. Wir empfehlen, vor der Migration auf MySQL Version 8.0.28 zu aktualisieren.

Migrieren von Daten aus einer externen MySQL-Datenbank zu einem Amazon-Aurora-MySQL-DB-Cluster

Wenn Ihre Datenbank die InnoDB- oder MyISAM-Tabellenräume unterstützt, können Sie die Daten wie folgt in einen Amazon Aurora MySQL-DB-Cluster migrieren:

- Sie können mithilfe des Dienstprogramms `mysqldump` eine Dump-Datei mit den Daten erstellen und diese dann in einen bestehenden Amazon Aurora MySQL-DB-Cluster importieren. Weitere Informationen finden Sie unter Logische Migration von MySQL zu Amazon Aurora MySQL mithilfe von mysqldump.
- Sie können die Dateien der vollständigen und inkrementellen Datenbanksicherungen in einen Amazon S3-Bucket kopieren und sie dann in einen Amazon Aurora MySQL-DB-Cluster wiederherstellen. Dieser Weg ist bedeutend schneller als eine Datenmigration mit `mysqldump`. Weitere Informationen finden Sie unter Physische Migration von MySQL mithilfe von Percona XtraBackup und Amazon S3.

Themen

- Physische Migration von MySQL mithilfe von Percona XtraBackup und Amazon S3
- Logische Migration von MySQL zu Amazon Aurora MySQL mithilfe von mysqldump

Physische Migration von MySQL mithilfe von Percona XtraBackup und Amazon S3

Sie können die Dateien der vollständigen und inkrementellen Sicherung aus Ihrer MySQL-Quelldatenbank (Version 5.7 oder 8.0) in einen Amazon S3-Bucket kopieren. Anschließend können Sie die Wiederherstellung zu einem Amazon Aurora MySQL-DB-Cluster mit derselben Hauptversion wie der dieser Dateien durchführen.

Diese Option kann erheblich schneller sein als das Migrieren der Daten mithilfe von `mysqldump`, da durch die Verwendung von `mysqldump` alle Befehle für die erneute Erstellung des Schemas und der Daten aus Ihrer Quelldatenbank im neuen Aurora MySQL-DB-Cluster erneut ausgeführt werden. Da die MySQL-Quelldatendateien kopiert werden, können sie von Aurora MySQL sofort als Daten für einen Aurora MySQL-DB-Cluster verwendet werden.

Dazu können Sie mithilfe der Binärprotokollreplikation während des Migrationsprozesses die Ausfallzeit minimieren. Wenn Sie die Binärprotokollreplikation verwenden, bleibt die externe MySQL-Datenbank für Transaktionen geöffnet, während die Daten in das Aurora MySQL-DB-Cluster migriert werden. Nachdem das Aurora MySQL-DB-Cluster erstellt wurde, können Sie das Aurora MySQL-

DB-Cluster mithilfe der Binärprotokollreplikation mit den Transaktionen synchronisieren, die nach der Sicherung ausgeführt wurden. Sobald das Aurora MySQL-DB-Cluster den Stand der MySQL-Datenbank erreicht hat, können Sie die Migration beenden, indem Sie bei neuen Transaktionen vollständig zum Aurora MySQL-DB-Cluster wechseln. Weitere Informationen finden Sie unter Synchronisieren des Amazon Aurora MySQL-DB-Clusters mit der MySQL-Datenbank mittels Replikation.

Inhalt

- Einschränkungen und Überlegungen
- Bevor Sie beginnen
  - Percona installieren XtraBackup
  - Erforderliche Berechtigungen
  - Erstellen der IAM-Servicerolle
- Sichern der wiederherzustellenden Dateien als Amazon Aurora MySQL-DB-Cluster
  - Erstellen Sie ein vollständiges Backup mit Percona XtraBackup
  - Verwenden von inkrementellen Backups mit Percona XtraBackup
  - Überlegungen zu Sicherungen
- Wiederherstellen eines Amazon Aurora-MySQL-DB-Clusters aus einem Amazon S3-Bucket
- Synchronisieren des Amazon Aurora MySQL-DB-Clusters mit der MySQL-Datenbank mittels Replikation
  - Konfigurieren der externen MySQL-Datenbank und des Aurora MySQL-DB-Clusters für die verschlüsselte Replikation
  - Synchronisieren des Amazon Aurora MySQL-DB-Clusters mit der externen MySQL-Datenbank
- Verkürzung der Zeit für die physische Migration zu Amazon Aurora MySQL
  - Nicht unterstützte Tabellentypen
  - Benutzerkonten mit nicht unterstützten Rechten
  - Dynamische Rechte in Aurora MySQL Version 3
  - Gespeicherte Objekte mit 'rdsadmin'@'localhost' als Definierer

Einschränkungen und Überlegungen

Die folgenden Einschränkungen und Überlegungen gelten für die Wiederherstellung zu einem Amazon Aurora MySQL DB-Cluster von einem Amazon S3-Bucket:

- Sie können Ihre Daten nur zu einem neuen und nicht zu einem bereits vorhandenen DB-Cluster migrieren.
- Sie müssen Percona verwenden XtraBackup , um Ihre Daten auf S3 zu sichern. Weitere Informationen finden Sie unter Percona installieren XtraBackup.
- Der Amazon S3 S3-Bucket und der Aurora MySQL-DB-Cluster müssen sich in derselben AWS Region befinden.
- Sie können nicht von folgenden Quellen aus wiederherstellen:
  - DB-Cluster-Snapshot-Export zu Amazon S3. Sie können auch keine Daten von einem DB-Cluster-Snapshot-Export in Ihren Amazon-S3-Bucket migrieren.
  - Verschlüsselte Quelldatenbank; Sie können jedoch die migrierten Daten verschlüsseln. Sie können die Daten während des Migrationsprozesses auch unverschlüsselt lassen.
  - MySQL 5.5- oder 5.6-Datenbank
- Percona Server für MySQL wird nicht als Quelldatenbank unterstützt, da `compression_dictionary*`-Tabellen im `mysql`-Schema enthalten sein können.
- Eine Wiederherstellung auf einen Aurora Serverless-DB-Cluster ist nicht möglich.
- Für Haupt- und Nebenversionen wird keine abwärtskompatible Migration unterstützt. Das heißt, Sie können nicht von MySQL Version 8.0 zu Aurora MySQL Version 2 (kompatibel mit MySQL 5.7) und auch nicht von MySQL Version 8.0.32 zu Aurora MySQL Version 3.03 migrieren, das mit der MySQL-Community-Version 8.0.26 kompatibel ist.
- Sie können von einigen älteren Versionen von MySQL 8.0, einschließlich 8.0.11, 8.0.13 und 8.0.15, nicht zu Aurora MySQL Version 3.05 und höher migrieren. Wir empfehlen, vor der Migration auf MySQL Version 8.0.28 zu aktualisieren.
- Das Importieren von Amazon S3 wird von der DB-Instance-Klasse db.t2.micro nicht unterstützt. Sie können jedoch eine Wiederherstellung zu einer anderen DB-Instance-Klasse ausführen und die DB-Instance-Klasse später ändern. Weitere Informationen zu DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
- Amazon S3 begrenzt die Größe einer Datei, die in einen S3-Bucket hochgeladen werden kann, auf 5 TB. Wenn eine Sicherungsdatei größer als 5 TB ist, müssen Sie die Sicherungsdatei in kleinere Dateien aufteilen.
- Amazon RDS begrenzt die Anzahl der Dateien, die in einen S3-Bucket hochgeladen werden können, auf 1 Million. Wenn es mehr als 1 Million Sicherungsdateien für Ihre Datenbank gibt (einschließlich aller vollständigen und inkrementellen Sicherungen), speichern Sie diese mittels Gzip (.gz), tar (.tar.gz) oder Percona xbstream (.xbstream) in dem S3-Bucket. Percona XtraBackup 8.0 unterstützt nur Percona xbstream für die Komprimierung.

- Um Verwaltungsdienste für jedes DB-Cluster bereitzustellen, wird der `rdsadmin`-Benutzer erstellt, wenn das DB-Cluster erstellt wird. Da dies ein reservierter Benutzer in RDS ist, gelten die folgenden Einschränkungen:
  - Funktionen, Prozeduren, Ansichten, Ereignisse und Trigger mit `'rdsadmin'@'localhost'`-Definierer werden nicht importiert. Weitere Informationen erhalten Sie unter Gespeicherte Objekte mit 'rdsadmin'@'localhost' als Definierer und Berechtigungen von Masterbenutzerkonten mit Amazon Aurora MySQL..
  - Wenn der Aurora MySQL-DB-Cluster erstellt wird, wird ein Master-Benutzer mit den maximal unterstützten Rechten erstellt. Bei der Wiederherstellung aus der Sicherung werden alle nicht unterstützten Rechte, die den importierten Benutzern zugewiesen wurden, beim Import automatisch entfernt.

    Informationen zu Benutzern, die davon betroffen sein könnten, finden Sie unter Benutzerkonten mit nicht unterstützten Rechten. Weitere Informationen zu den in Aurora MySQL unterstützten Rechten finden Sie unter Rollenbasiertes Berechtigungsmodell.
- Für Aurora MySQL Version 3 werden keine dynamischen Rechte importiert. Von Aurora unterstützte dynamische Rechte können nach der Migration importiert werden. Weitere Informationen finden Sie unter Dynamische Rechte in Aurora MySQL Version 3.
- Vom Benutzer erstellte Tabellen im `mysql`-Schema werden nicht migriert.
- Der `innodb_data_file_path`-Parameter darf nur mit einer Datendatei konfiguriert werden, die den Standarddateinamen `ibdata1:12M:autoextend` verwendet. Datenbanken mit zwei Datendateien oder nur einer Datendatei eines anderen Namens können mit dieser Methode nicht migriert werden.

  Nachfolgend finden Sie Beispiele für unzulässige Dateinamen: `innodb_data_file_path=ibdata1:50M`, `ibdata2:50M:autoextend` und `innodb_data_file_path=ibdata01:50M:autoextend`.
- Sie können nicht von einer Quelldatenbank migrieren, deren Tabellen außerhalb des standardmäßigen MySQL-Datenverzeichnisses definiert sind.
- Die maximal unterstützte Größe für unkomprimierte Sicherungen mit dieser Methode ist derzeit auf 64 TiB begrenzt. Bei komprimierten Sicherungen wird dieser Grenzwert gesenkt, um den Speicheranforderungen bei der Dekomprimierung Rechnung zu tragen. In solchen Fällen wäre die maximal unterstützte Backup-Größe (`64 TiB – compressed backup size`).
- Aurora MySQL unterstützt den Import von MySQL und anderen externen Komponenten und Plugins nicht.

- Aurora MySQL stellt nicht die gesamte Datenbank wieder her. Sie sollten das Datenbankschema und die Werte der folgenden Elemente der MySQL-Quelldatenbank speichern und dem Aurora MySQL-DB-Cluster nach der Wiederherstellung hinzufügen:
  - Benutzerkonten
  - Funktionen
  - Gespeicherte Prozeduren
  - Zeitzoneninformation. Die Zeitzoneninformation wird vom lokalen Betriebssystem des Amazon Aurora MySQL-DB-Clusters übernommen. Weitere Informationen finden Sie unter Lokale Zeitzone für Amazon Aurora-DB-Cluster.

Bevor Sie beginnen

Bevor Sie Ihre Daten in einen Amazon S3-Bucket kopieren und daraus einen DB-Cluster wiederherstellen können, müssen Sie folgende Schritte durchführen:

- Installieren Sie Percona XtraBackup auf Ihrem lokalen Server.
- Autorisieren Sie Aurora MySQL für den Zugriff auf den Amazon S3-Bucket in Ihrem Namen.

Percona installieren XtraBackup

Amazon Aurora kann einen DB-Cluster aus Dateien wiederherstellen, die mit Percona XtraBackup erstellt wurden. Sie können Percona XtraBackup über Software-Downloads — Percona installieren.

Verwenden Sie für die MySQL 5.7-Migration Percona XtraBackup 2.4.

Verwenden Sie für die MySQL 8.0-Migration Percona XtraBackup 8.0. Stellen Sie sicher, dass die XtraBackup Percona-Version mit der Engine-Version Ihrer Quelldatenbank kompatibel ist.

Erforderliche Berechtigungen

Für die Migration von MySQL-Daten in einen Amazon Aurora MySQL-DB-Cluster sind mehrere Berechtigungen erforderlich:

- Der Benutzer, der Aurora auffordert, einen neuen Cluster aus einem Amazon S3 S3-Bucket zu erstellen, muss über die Berechtigung verfügen, die Buckets für Ihr AWS Konto aufzulisten. Sie gewähren dem Benutzer diese Berechtigung mithilfe einer AWS Identity and Access Management (IAM-) Richtlinie.

- Aurora benötigt die Berechtigung, in Ihrem Namen auf den Amazon S3-Bucket mit den Dateien zum Erstellen des Amazon Aurora MySQL-DB-Clusters zuzugreifen. Sie erteilen Aurora die erforderlichen Berechtigungen, indem Sie eine IAM-Servicerolle verwenden.
- Der anfordernde Benutzer muss zudem über die Berechtigung zum Auflisten der IAM-Rollen Ihres AWS -Kontos verfügen.
- Wenn der anfordernde Benutzer die IAM-Servicerolle erstellen oder anfordern möchte, dass die IAM-Servicerolle von Aurora erstellt wird (über die Konsole), muss er über die Berechtigung zum Erstellen einer IAM-Rolle für Ihr AWS -Konto verfügen.
- Wenn Sie beabsichtigen, die Daten während des Migrationsprozesses zu verschlüsseln, aktualisieren Sie die IAM-Richtlinie des Benutzers, der die Migration durchführen wird, um RDS-Zugriff auf die für die Verschlüsselung der Backups AWS KMS keys verwendeten Daten zu gewähren. Detaillierte Anweisungen finden Sie unter Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS KMS.

Die folgende IAM-Richtlinie gewährt einem Benutzer beispielsweise die erforderlichen Mindestberechtigungen zur Verwendung der Konsole, um IAM-Rollen aufzulisten, eine IAM-Rolle zu erstellen, die Amazon S3-Buckets für Ihr Konto aufzulisten und die KMS-Schlüssel aufzulisten.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "kms:ListKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

Damit ein IAM-Benutzer eine IAM-Rolle mit einem Amazon S3-Bucket verknüpfen kann, muss der IAM-Benutzer die Berechtigung `iam:PassRole` für diese IAM-Rolle besitzen. Mit dieser Berechtigung kann ein Administrator einschränken, welche IAM-Rollen ein Benutzer mit Amazon S3-Buckets verknüpfen kann.

So ermöglicht die folgende IAM-Richtlinie einem Benutzer, die Rolle namens `S3Access` mit einem Amazon S3-Bucket zu verknüpfen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowS3AccessRole",
            "Effect":"Allow",
            "Action":"iam:PassRole",
            "Resource":"arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

Weitere Informationen zu IAM-Benutzerberechtigungen finden Sie unter Verwalten des Zugriffs mit Richtlinien.

Erstellen der IAM-Servicerolle

Sie können eine Rolle für sich selbst AWS-Managementkonsole erstellen lassen, indem Sie die Option Neue Rolle erstellen wählen (siehe weiter unten in diesem Thema). Wenn Sie diese Option auswählen und einen Namen für die neue Rolle angeben, erstellt Aurora die benötigte IAM-Servicerolle, damit Aurora mit dem von Ihnen angegebenen Namen auf Ihren Amazon S3-Bucket zugreifen kann.

Alternativ können Sie die Rolle auf folgende Weise manuell erstellen.

So erstellen Sie eine IAM-Rolle für Aurora zum Zugriff auf Amazon S3

1. Führen Sie die Schritte unter au Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen.

2. Führen Sie die Schritte unter au Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.
3. Führen Sie die Schritte unter au Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster.

Sichern der wiederherzustellenden Dateien als Amazon Aurora MySQL-DB-Cluster

Sie können mit Percona eine vollständige Sicherung Ihrer MySQL-Datenbankdateien erstellen XtraBackup und die Sicherungsdateien in einen Amazon S3 S3-Bucket hochladen. Wenn Sie Percona bereits XtraBackup zum Sichern Ihrer MySQL-Datenbankdateien verwenden, können Sie alternativ Ihre vorhandenen vollständigen und inkrementellen Backup-Verzeichnisse und -Dateien in einen Amazon S3 S3-Bucket hochladen.

Themen

- Erstellen Sie ein vollständiges Backup mit Percona XtraBackup
- Verwenden von inkrementellen Backups mit Percona XtraBackup
- Überlegungen zu Sicherungen

Erstellen Sie ein vollständiges Backup mit Percona XtraBackup

Um eine vollständige Sicherung Ihrer MySQL-Datenbankdateien zu erstellen, die aus Amazon S3 wiederhergestellt werden kann, um einen Aurora MySQL-DB-Cluster zu erstellen, verwenden Sie das XtraBackup Percona-Hilfsprogramm (`xtrabackup`), um Ihre Datenbank zu sichern.

Mit dem folgenden Befehl können Sie beispielsweise eine Sicherung einer MySQL-Datenbank erstellen und die Dateien im Ordner `/on-premises/s3-restore/backup` speichern.

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/
s3-restore/backup>
```

Wenn Sie die Sicherung in einer Archivdatei komprimieren möchten (die bei Bedarf aufgeteilt werden kann), können Sie mit der Option `--stream` eines der folgenden Formate festlegen:

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

Mit dem folgenden Befehl wird eine Sicherung einer MySQL-Datenbank erstellt und in mehreren Gzip-Dateien gespeichert.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar.gz
```

Mit dem folgenden Befehl wird ein Backup einer MySQL-Datenbank erstellt und in mehreren tar-Dateien gespeichert.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar
```

Mit dem folgenden Befehl wird ein Backup einer MySQL-Datenbank erstellt und in mehreren xbstream-Dateien gespeichert.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.xbstream
```

> **Note**
>
> Falls der folgende Fehler angezeigt wird, haben Sie möglicherweise unterschiedliche Dateiformate in Ihrem Befehl verwendet:
>
> ```
> ERROR:/bin/tar: This does not look like a tar archive
> ```

Sobald Sie Ihre MySQL-Datenbank mit dem XtraBackup Percona-Hilfsprogramm gesichert haben, können Sie Ihre Backup-Verzeichnisse und -Dateien in einen Amazon S3 S3-Bucket kopieren.

Weitere Informationen zum Erstellen und Hochladen einer Datei in einen Amazon-S3-Bucket finden Sie unter Erste Schritte mit Amazon Simple Storage Service im Amazon-S3-Handbuch "Erste Schritte".

Verwenden von inkrementellen Backups mit Percona XtraBackup

Amazon Aurora MySQL unterstützt sowohl vollständige als auch inkrementelle Backups, die mit XtraBackup Percona erstellt wurden. Wenn Sie Percona bereits verwenden XtraBackup , um

vollständige und inkrementelle Backups Ihrer MySQL-Datenbankdateien durchzuführen, müssen Sie kein vollständiges Backup erstellen und die Sicherungsdateien auf Amazon S3 hochladen. Sie können stattdessen viel Zeit sparen, indem Sie die vorhandenen Sicherungsverzeichnisse und -dateien der vollständigen und inkrementelle Sicherungen in einen Amazon S3-Bucket hochladen. Weitere Informationen finden Sie unter Ein inkrementelles Backup erstellen auf der Percona-Website.

Wenn Sie die Dateien der vollständigen und inkrementellen Backups in einen Amazon S3-Bucket hochladen, müssen Sie den Inhalt des Basisverzeichnisses rekursiv kopieren. Es müssen sämtliche Verzeichnisse und Dateien der vollständigen und inkrementellen Backups enthalten sein. Diese Kopie muss die Verzeichnisstruktur im Amazon-S3-Bucket beibehalten. Aurora durchläuft alle Dateien und Verzeichnisse. Aurora verwendet die in jedem inkrementellen Backup enthaltene `xtrabackup-checkpoints`-Datei, um das Basisverzeichnis zu identifizieren und inkrementelle Backups nach dem Bereich der Protokollsequenznummer (Log Sequence Number, LSN) zu ordnen.

Weitere Informationen zum Erstellen und Hochladen einer Datei in einen Amazon-S3-Bucket finden Sie unter Erste Schritte mit Amazon Simple Storage Service im Amazon-S3-Handbuch "Erste Schritte".

Überlegungen zu Sicherungen

Aurora unterstützt keine Teil-Backups, die mit Percona XtraBackup erstellt wurden. Sie können beim Sichern der Quelldateien Ihrer Datenbank nicht die folgenden Optionen verwenden, um eine Teilsicherung zu erstellen: `--tables`, `--tables-exclude`, `--tables-file`, `--databases`, `--databases-exclude` oder `--databases-file`.

Weitere Informationen zum Sichern Ihrer Datenbank mit Percona finden Sie unter Percona XtraBackup XtraBackup — Dokumentation und Arbeiten mit Binärprotokollen auf der Percona-Website.

Aurora unterstützt inkrementelle Backups, die mit XtraBackup Percona erstellt wurden. Weitere Informationen finden Sie unter Ein inkrementelles Backup erstellen auf der Percona-Website.

Aurora verarbeitet Sicherungsdateien auf Basis des Dateinamens. Achten Sie daher unbedingt darauf, dass die Namenserweiterungen der Sicherungsdateien dem Dateiformat entsprechen — z. B. `.xbstream` für Dateien, die im xbstream-Format von Percona gespeichert wurden.

Aurora verarbeitet Sicherungsdateien in alphanumerischer Reihenfolge. Verwenden Sie immer die Option `split` für den Befehl `xtrabackup`, um sicherzustellen, dass die Sicherungsdateien in der richtigen Reihenfolge geschrieben und benannt werden.

Amazon S3 begrenzt die Größe einer Datei, die in einen Amazon S3-Bucket hochgeladen werden kann, auf 5 TB. Wenn die Sicherungsdaten Ihrer Datenbank 5 TB überschreiten, müssen Sie die Sicherungsdateien mit dem Befehl `split` in mehrere Dateien aufteilen, die jeweils kleiner als 5 TB sind.

Aurora begrenzt die Anzahl der Quelldateien, die in einen Amazon S3-Bucket hochgeladen werden können, auf 1 Million Dateien. In manchen Situationen können die Sicherungsdaten Ihrer Datenbank einschließlich aller vollständigen und inkrementellen Sicherungen aus sehr vielen Dateien bestehen. Verwenden Sie in diesen Fällen ein tar-Archiv (.tar.gz), um die Dateien der vollständigen und inkrementellen Sicherungen im Amazon S3-Bucket zu speichern.

Sie können Dateien beim Hochladen in einen Amazon S3-Bucket serverseitig verschlüsseln lassen. Anschließend können Sie aus diesen verschlüsselten Dateien ein Amazon-Aurora-MySQL-DB-Cluster wiederherstellen. Amazon Aurora MySQL kann DB-Cluster mit Dateien wiederherstellen, die mittels der folgenden Arten der serverseitigen Verschlüsselung verschlüsselt wurden:

- Serverseitige Verschlüsselung mit von Amazon S3 verwalteten Schlüsseln (SSE-S3): Jedes Objekt wird mit einem eindeutigen Schlüssel mit starker Multifaktor-Verschlüsselung verschlüsselt.
- Serverseitige Verschlüsselung mit AWS KMS verwalteten Schlüsseln (SSE-KMS) — Ähnlich wie SSE-S3, aber Sie haben die Möglichkeit, Verschlüsselungsschlüssel selbst zu erstellen und zu verwalten, und es gibt auch andere Unterschiede.

Informationen zur serverseitigen Verschlüsselung beim Hochladen von Dateien in einen Amazon S3-Bucket finden Sie unter Schützen von Daten mithilfe serverseitiger Verschlüsselung im Amazon S3-Entwicklerhandbuch.

Wiederherstellen eines Amazon Aurora-MySQL-DB-Clusters aus einem Amazon S3-Bucket

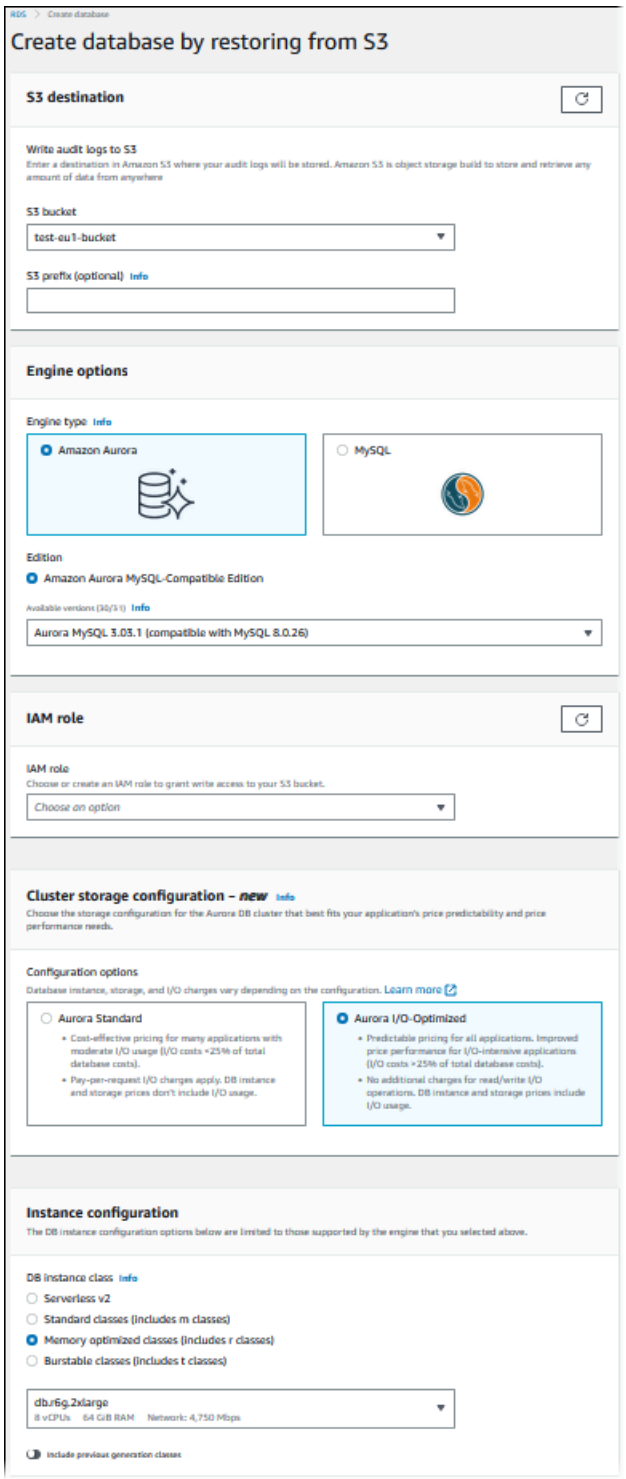
Sie können Ihre Sicherungsdateien aus Ihrem Amazon-S3-Bucket wiederherstellen, um einen neuen Amazon-Aurora-MySQL-DB-Cluster zu erstellen, indem Sie die Amazon-RDS-Konsole verwenden.

So stellen Sie ein Amazon Aurora MySQL-DB-Cluster aus Dateien in einem Amazon S3-Bucket wieder her

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der oberen rechten Ecke der Amazon RDS-Konsole die AWS Region aus, in der Sie Ihren DB-Cluster erstellen möchten. Wählen Sie dieselbe AWS Region wie der Amazon S3 S3-Bucket, der Ihr Datenbank-Backup enthält.

3. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann die Option Restore from S3 (Aus S3 wiederherstellen) aus.
4. Wählen Sie Von S3 wiederherstellen.

   Die Seite Datenbank durch Wiederherstellen von S3 erstellen wird angezeigt.



5. Unter S3-Ziel:

   a. Wählen Sie den S3-Bucket aus, in dem sich die Sicherungsdateien befinden.

   b. (Optional) Geben Sie für S3 folder path prefix (S3-Ordnerpfadpräfix) ein Dateipfadpräfix für die Dateien ein, die in Ihrem Amazon S3-Bucket gespeichert sind.

      Wenn Sie kein Präfix angeben, dann erstellt RDS Ihre DB-Instance unter Verwendung aller Dateien und Ordner im Stammordner des S3-Bucket. Wenn Sie ein Präfix angeben, erstellt RDS Ihre DB-Instance mit den Ordnern und Dateien im S3-Bucket, deren vollständiger Pfadname mit dem angegebenen Präfix beginnt.

      Beispiel: Sie speichern Sicherungsdateien auf S3 in einem Unterordner namens 'backups' und haben mehrere Sätze von Sicherungsdateien, die sich jeweils in einem eigenen Verzeichnis befinden (gzip_backup1, gzip_backup2 usw.). Sie müssen nun das Präfix backups/gzip_backup1 angeben, um die Wiederherstellung mit den Dateien im Ordner gzip_backup1 durchzuführen.

6. Unter Engine-Optionen:

   a. Wählen Sie in Engine type (Engine-Typ) die Option Amazon Aurora aus.

   b. Wählen Sie für Version die Aurora MySQL-Engine-Version für Ihre wiederhergestellte DB-Instance aus.

7. Für die IAM-Rolle können Sie eine vorhandene IAM-Rolle auswählen.

8. (Optional) Sie können auch eine neue IAM-Rolle erstellen lassen, indem Sie Neue Rolle erstellen wählen. Wenn ja:

   a. Geben Sie den IAM-Rollennamen ein.

   b. Wählen Sie aus, ob Sie den Zugriff auf den KMS-Schlüssel zulassen möchten:

      - Wenn Sie die Sicherungsdateien nicht verschlüsselt haben, wählen Sie No (Nein) aus.
      - Wenn Sie die Sicherungsdateien mit AES-256 (SSE-S3) verschlüsselt haben, als Sie diese zu Amazon S3 hochgeladen haben, wählen Sie Nein aus. In diesem Fall werden die Daten automatisch entschlüsselt.
      - Wenn Sie die Sicherungsdateien beim Hochladen auf Amazon S3 mit serverseitiger Verschlüsselung AWS KMS (SSE-KMS) verschlüsselt haben, wählen Sie Ja. Wählen Sie dann den richtigen KMS-Schlüssel für AWS KMS key.

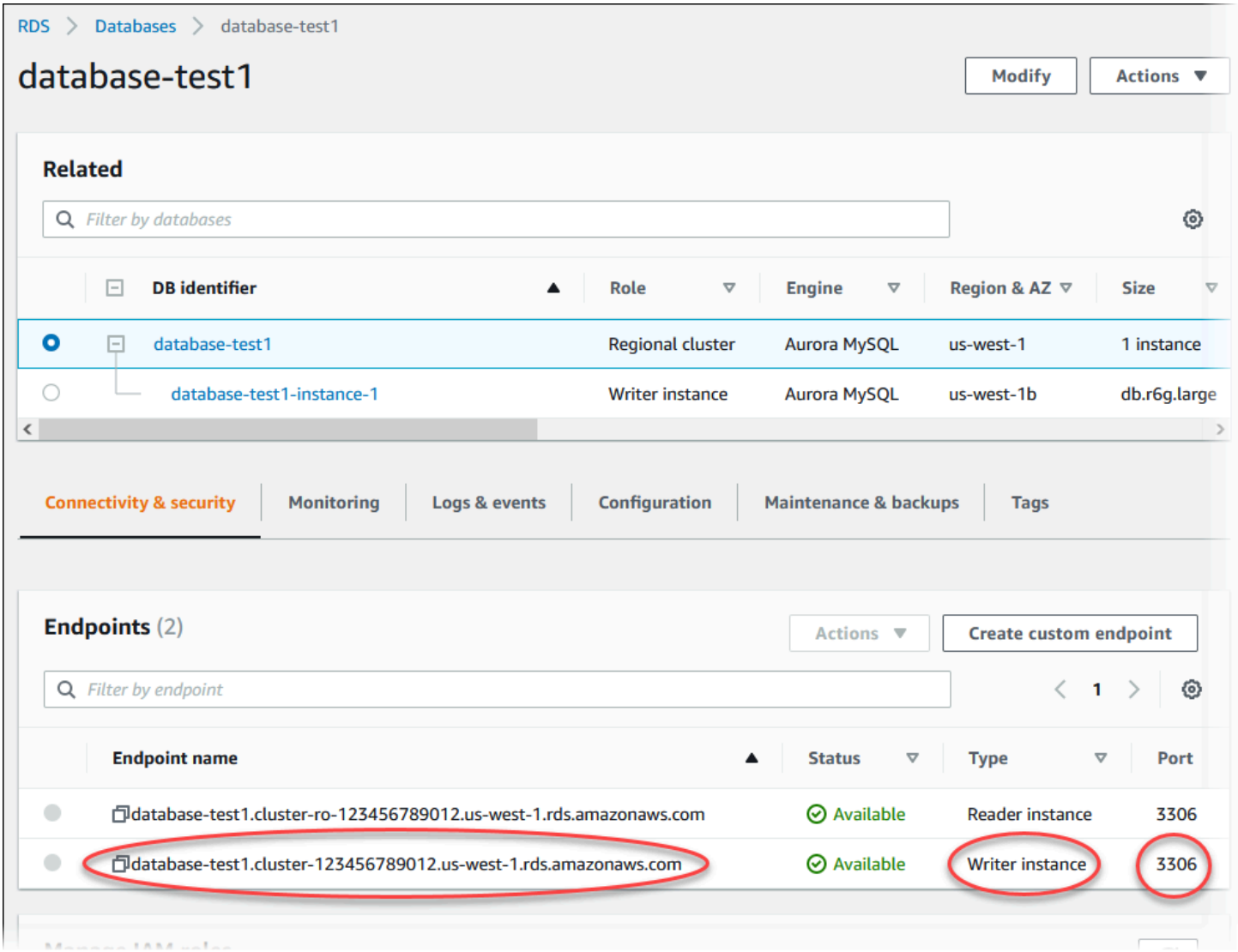
Das AWS-Managementkonsole erstellt eine IAM-Richtlinie, die es Aurora ermöglicht, die Daten zu entschlüsseln.

Weitere Informationen finden Sie unter Schützen von Daten mithilfe serverseitiger Verschlüsselung im Amazon S3-Entwicklerhandbuch.

9. Wählen Sie Einstellungen für Ihren DB-Cluster aus, z. B. die DB-Cluster-Speicherkonfiguration, die DB-Instance-Klasse, die DB-Cluster-ID und die Anmeldeinformationen. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.
10. Passen Sie zusätzliche Einstellungen für Ihren Aurora MySQL-DB-Cluster nach Bedarf an.
11. Wählen Sie Create database (Datenbank erstellen) , um Ihre Aurora-DB-Instance zu starten.

In der Amazon-RDS-Konsole wird die neue DB-Instance in der Liste der DB-Instances angezeigt. Die DB-Instance wird mit dem Status creating (Wird erstellt) angezeigt, bis sie erstellt wurde und einsatzbereit ist. Wenn sich der Status in available (Verfügbar) ändert, können Sie die Verbindung mit der primären Instance des DB-Clusters herstellen. Je nach Klasse und Speicherort der DB-Instance kann es einige Minuten dauern, bis sie verfügbar ist.

Um das neu erstellte Cluster anzuzeigen, wählen Sie die Ansicht Database (Datenbank) in der Amazon-RDS-Konsole und anschließend das DB-Cluster aus. Weitere Informationen finden Sie unter Anzeigen eines DB-Clusters von Amazon Aurora.



Notieren Sie den Port und den Writer-Endpunkt des DB-Clusters. Verwenden Sie den Writer-Endpunkt und den Port des DB-Clusters in den JDBC- und ODBC-Verbindungszeichenfolgen aller Anwendungen, die Lese- oder Schreibvorgänge ausführen.

Synchronisieren des Amazon Aurora MySQL-DB-Clusters mit der MySQL-Datenbank mittels Replikation

Damit es zu einer möglichst kurzen bzw. überhaupt keiner Unterbrechung während der Migration kommt, können Sie die in der MySQL-Datenbank durchgeführten Transaktionen mittels Replikation in das Aurora MySQL-DB-Cluster übernehmen. Durch die Replikation können die in der MySQL-Datenbank während der Migration durchgeführten Transaktionen in das DB-Cluster übernommen werden. Sobald das DB-Cluster auf dem Stand des MySQL-Masters ist, können Sie die Replikation anhalten und die Migration nach Aurora MySQL beenden.

Themen

- Konfigurieren der externen MySQL-Datenbank und des Aurora MySQL-DB-Clusters für die verschlüsselte Replikation
- Synchronisieren des Amazon Aurora MySQL-DB-Clusters mit der externen MySQL-Datenbank

Konfigurieren der externen MySQL-Datenbank und des Aurora MySQL-DB-Clusters für die verschlüsselte Replikation

Wenn Sie Daten sicher replizieren möchten, können Sie eine verschlüsselte Replikation verwenden.

> **Note**
>
> Wenn Sie keine verschlüsselte Replikation benötigen, können Sie diese Schritte überspringen und mit der Anleitung unter fortfahre Synchronisieren des Amazon Aurora MySQL-DB-Clusters mit der externen MySQL-Datenbank.

Folgende Voraussetzungen gelten für die Verwendung einer verschlüsselten Replikation:

- Secure Sockets Layer (SSL) muss in der externen MySQL-Primärdatenbank aktiviert sein.
- Ein Clientschlüssel und ein Clientzertifikat müssen für das Aurora MySQL-DB-Cluster vorbereitet werden.

Während der verschlüsselten Replikation dient das Aurora MySQL-DB-Cluster als Client für den MySQL-Datenbankserver. Die Zertifikate und Schlüssel für den Aurora MySQL-Client befinden sich in Dateien im PEM-Format.

So konfigurieren Sie die externe MySQL-Datenbank und das Aurora MySQL-DB-Cluster für die verschlüsselte Replikation

1. Stellen Sie sicher, dass alle Vorbereitungen für die verschlüsselte Replikation getroffen wurden:
   - Wenn SSL auf dem externen Server mit der MySQL-Primärdatenbank nicht aktiviert ist und Sie keinen Clientschlüssel und kein Clientzertifikat vorbereitet haben, aktivieren Sie SSL auf dem MySQL-Datenbankserver und generieren Sie den Clientschlüssel und das Clientzertifikat.
   - Wenn SSL auf die externen Primäre aktiviert ist, geben Sie einen Clientschlüssel und ein Clientzertifikat für das Aurora MySQL-DB-Cluster an. Wenn Sie diese Werte nicht haben,

erstellen Sie ein einen neuen Schlüssel und ein neues Zertifikat für das Aurora MySQL-DB-Cluster. Sie benötigen zur Signierung des Clientzertifikats den Zertifizierungsstellenschlüssel, den Sie zum Konfigurieren von SSL auf der externen MySQL-Primärdatenbank verwendet haben.

Weitere Informationen finden Sie unter Creating SSL Certificates and Keys Using openssl in der MySQL-Dokumentation.

Sie benötigen das Zertifizierungsstellenzertifikat, den Clientschlüssel und das Clientzertifikat.

2. Stellen Sie als Primärbenutzer über SSL eine Verbindung zum Aurora MySQL-DB-Cluster her.

   Informationen zum Herstellen einer Verbindung mit einem Aurora MySQL-DB-Cluster über SSL finden Sie unter TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

3. Führen Sie die gespeicherte Prozedur mysql.rds_import_binlog_ssl_material aus, um die SSL-Informationen in das Aurora MySQL-DB-Cluster zu importieren.

   Fügen Sie die Informationen aus den PEM-Dateien für das Aurora MySQL-DB-Cluster in die korrekte JSON-Nutzlast für den Parameter `ssl_material_value` ein.

   Im folgenden Beispiel werden SSL-Informationen in ein Aurora MySQL-DB-Cluster importiert. Der Code in den PEM-Dateien ist in der Regel länger als in diesem Beispiel.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
```

```
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

Weitere Informationen erhalten Sie unter mysql.rds_import_binlog_ssl_material und TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

> **Note**
>
> Nach dem Ausführen der Prozedur werden die SSL-Informationen in Dateien gespeichert. Um die Dateien später zu löschen, können Sie die mysql.rds_remove_binlog_ssl_material gespeicherte Prozedur ausführen.

Synchronisieren des Amazon Aurora MySQL-DB-Clusters mit der externen MySQL-Datenbank

Sie können das Amazon Aurora MySQL-DB-Cluster mittels Replikation mit der MySQL-Datenbank synchronisieren.

So synchronisieren Sie das Aurora MySQL-DB-Cluster mittels Replikation mit der MySQL-Datenbank

1. Stellen Sie sicher, dass die Datei "/etc/my.cnf" für die externe MySQL-Datenbank die relevanten Einträge enthält.

   Wenn keine verschlüsselte Replikation erforderlich ist, muss die externe MySQL-Datenbank mit aktivierten Binärprotokollen (binlogs) und deaktiviertem SSL gestartet werden. Dies sind die relevanten Einträge für unverschlüsselte Daten in der Datei "/etc/my.cnf".

   ```
   log-bin=mysql-bin
   server-id=2133421
   innodb_flush_log_at_trx_commit=1
   sync_binlog=1
   ```

   Wenn die verschlüsselte Replikation erforderlich ist, muss die externe MySQL-Datenbank mit aktivierten Binärprotokollen und aktiviertem SSL gestartet werden. Die Einträge in der Datei "/etc/my.cnf" müssen dann die Speicherorte der PEM-Dateien für den MySQL-Datenbankserver enthalten.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

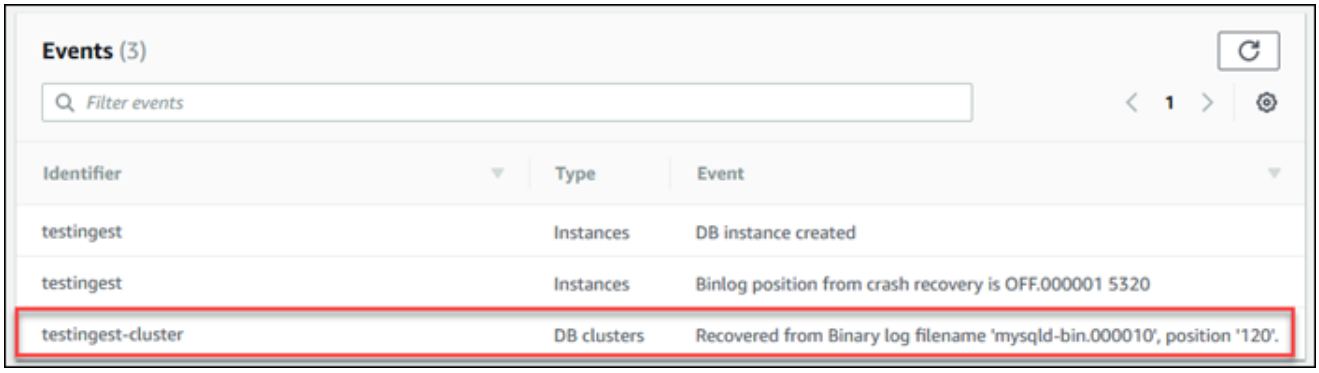
# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

Mit dem folgenden Befehl können Sie prüfen, ob SSL aktiviert ist.

```
mysql> show variables like 'have_ssl';
```

Die Ausgabe sollte in etwa wie folgt aussehen.

```
+~-~-~-
```



Verwenden Sie den AWS CLI

Sie können den Namen und die Position der Binlog-Datei auch mit dem Befehl AWS CLI describe-events abrufen. Im Folgenden wird ein `describe-events` Beispielbefehl gezeigt.

```
PROMPT> aws rds describe-events
```

Identifizieren Sie in der Ausgabe das Ereignis, das die Binärprotokoll-Position anzeigt.

3. Erstellen Sie bei bestehender Verbindung mit der externen MySQL-Datenbank einen Benutzer für die Replikation. Dieses Konto wird ausschließlich für die Replikation verwendet und muss auf Ihre Domäne beschränkt sein, um die Sicherheit zu erhöhen. Im Folgenden wird ein Beispiel gezeigt.

```
mysql> CREATE USER '<user_name>'@'<domain_name>' IDENTIFIED BY '<password>';
```

Der Benutzer benötigt die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE`. Gewähren Sie dem Benutzer diese Berechtigungen.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO
 '<user_name>'@'<domain_name>';
```

Wenn die Replikation verschlüsselt durchgeführt werden muss, fordern Sie für den Replikationsbenutzer eine SSL-Verbindung an. Sie können beispielsweise das folgende Statement verwenden, damit für das Benutzerkonto "encrypted_user" eine SSL-Verbindung *<user_name>*.

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

> **Note**
>
> Wenn `REQUIRE SSL` nicht angegeben wird, kann die Replikationsverbindung ohne entsprechende Meldung auf eine unverschlüsselte Verbindung zurückgesetzt werden.

4. Fügen Sie in der Amazon-RDS-Console die IP-Adresse des Servers, der die externe MySQL-Datenbank hostet, zur VPC-Sicherheitsgruppe für den Aurora MySQL-DB-Cluster hinzu. Weitere Informationen zum Ändern einer VPC-Sicherheitsgruppe finden Sie unter Sicherheitsgruppen für Ihre VPC im Amazon Virtual Private Cloud-Benutzerhandbuch.

   Sie müssen möglicherweise noch das lokale Netzwerk so konfigurieren, dass es Verbindungen von der IP-Adresse des Aurora MySQL-DB-Clusters zulässt, damit Sie mit der externen MySQL-Datenbank kommunizieren können. Verwenden Sie den Befehl `host`, um die IP-Adresse des Aurora MySQL-DB-Clusters herauszufinden.

   ```
   host <db_cluster_endpoint>
   ```

   Der Hostname ist der DNS-Name aus dem Aurora MySQL-DB-Cluster-Endpunkt.

5. Aktivieren Sie die Binärprotokollreplikation, indem Sie die gespeicherte Prozedur mysql.rds_reset_external_master (Aurora-MySQL-Version 2) oder mysql.rds_reset_external_source (Aurora-MySQL-Version 3) ausführen. Diese gespeicherte Prozedur hat die folgende Syntax.

   ```
   CALL mysql.rds_set_external_master (
     host_name
     , host_port
     , replication_user_name
     , replication_user_password
     , mysql_binary_log_file_name
     , mysql_binary_log_file_location
     , ssl_encryption
   );

   CALL mysql.rds_set_external_source (
   ```

```
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Informationen zu den Parametern finden Sie unter mysql.rds_reset_external_master (Aurora-MySQL-Version 2) und mysql.rds_reset_external_source (Aurora-MySQL-Version 3).

Verwenden Sie für `mysql_binary_log_file_name` und `mysql_binary_log_file_location` die Position im Ereignis Recovered from Binary log filename (Wiederhergestellt aus Binärprotokoll-Dateiname), die Sie zuvor notiert haben.

Wenn die Daten im Aurora MySQL-DB-Cluster nicht verschlüsselt sind, muss der Parameter `ssl_encryption` auf `0` festgelegt werden. Sind die Daten verschlüsselt, muss der Parameter `ssl_encryption` auf 1 gesetzt werden.

Im folgenden Beispiel wird die Prozedur für ein Aurora MySQL-DB-Cluster mit verschlüsselten Daten ausgeführt.

```
CALL mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);

CALL mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);
```

Diese gespeicherte Prozedur weist die Parameter zu, die das Aurora MySQL-DB-Cluster zum Herstellen einer Verbindung mit der externen MySQL-Datenbank und zum Lesen seines Binärprotokolls verwendet. Wenn die Daten verschlüsselt sind, werden auch das SSL-Zertifizierungsstellenzertifikat, das Clientzertifikat und der Clientschlüssel auf die lokale Festplatte heruntergeladen.

6. Starten Sie die Binärprotokollreplikation, indem Sie die gespeicherte Prozedur mysql.rds_start_replication ausführen.

```
CALL mysql.rds_start_replication;
```

7. Überwachen Sie den Rückstand des Aurora MySQL-DB-Clusters gegenüber der MySQL-Replikationsprimärdatenbank. Stellen Sie dazu eine Verbindung zum Aurora MySQL-DB-Cluster her und führen Sie den folgenden Befehl aus.

```
Aurora MySQL version 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

In der Befehlsausgabe wird im Feld `Seconds Behind Master` angezeigt, wie weit das Aurora MySQL-DB-Cluster hinter der MySQL-Primären zurückliegt. Wenn dieser Wert `0` (null) lautet, hat das Aurora MySQL-DB-Cluster den Stand der Primären erreicht und Sie können im nächsten Schritt die Replikation anhalten.

8. Stellen Sie eine Verbindung zur MySQL-Replikationsprimärdatenbank her und halten Sie die Replikation an. Rufen Sie dazu die gespeicherte Prozedur mysql.rds_stop_replication auf.

```
CALL mysql.rds_stop_replication;
```

Verkürzung der Zeit für die physische Migration zu Amazon Aurora MySQL

Mit den folgenden Datenbankänderungen können Sie den Vorgang der Migration einer Datenbank zu Amazon Aurora MySQL beschleunigen.

> **Important**
>
> Stellen Sie sicher, dass Sie diese Updates auf einer Kopie der Produktionsdatenbank und nicht auf der Produktionsdatenbank selbst durchführen. Anschließend können Sie die Kopie sichern und zu Ihrem Aurora MySQL DB-Cluster wiederherstellen, um Service-Unterbrechungen in der Produktionsdatenbank vermeiden.

Nicht unterstützte Tabellentypen

Aurora MySQL unterstützt nur die InnoDB-Engine für Datenbanktabellen. Wenn die Datenbank MyISAM-Tabellen enthält, müssen diese vor der Migration nach Aurora MySQL konvertiert werden. Während der Migration wird für die Konvertierung von MyISAM nach InnoDB zusätzlicher Speicherplatz benötigt.

Um das Risiko zu senken, dass der Speicherplatz nicht ausreicht, oder um den Migrationsvorgang zu beschleunigen, wandeln Sie alle MyISAM-Tabellen vor der Migration in InnoDB-Tabellen um. Die Größe der resultierenden InnoDB-Tabelle entspricht den Größenanforderungen von Aurora MySQL für diese Tabelle. Führen Sie zum Umwandeln einer MyISAM-Tabelle in eine InnoDB-Tabelle den folgenden Befehl aus:

```
ALTER TABLE schema.table_name engine=innodb, algorithm=copy;
```

Aurora MySQL unterstützt keine komprimierten Tabellen oder Seiten (d. h. mit `ROW_FORMAT=COMPRESSED` oder `COMPRESSION = {"zlib"|"lz4"}` erstellte Tabellen).

Um das Risiko zu senken, dass der Speicherplatz nicht ausreicht, oder um den Migrationsvorgang zu beschleunigen, erweitern Sie komprimierte Tabellen, indem Sie `ROW_FORMAT` auf `DEFAULT`, `COMPACT`, `DYNAMIC` oder `REDUNDANT` einstellen. Stellen Sie für komprimierte Seiten `COMPRESSION="none"` ein.

Weitere Informationen finden Sie unter InnoDB-Zeilenformate und InnoDB-Tabellen- und Seitenkomprimierung in der MySQL-Dokumentation.

Sie können mit dem folgenden SQL-Skript alle Datenbanktabellen der MySQL-DB-Instance auflisten, die MyISAM-Tabellen oder komprimierte Tabellen sind.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Aurora MySQL.
-- It must be run from an account that has read permission for the
```

```
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
```

```
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

Benutzerkonten mit nicht unterstützten Rechten

Benutzerkonten mit Rechten, die von Aurora MySQL nicht unterstützt werden, werden ohne die nicht unterstützten Rechte importiert. Die Liste der unterstützten Rechte finden Sie unter Rollenbasiertes Berechtigungsmodell.

Sie können die folgende SQL-Abfrage in Ihrer Quelldatenbank ausführen, um die Benutzerkonten aufzulisten, die nicht unterstützte Rechte haben.

```
SELECT
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Dynamische Rechte in Aurora MySQL Version 3

Dynamische Rechte werden nicht importiert. Aurora MySQL Version 3 unterstützt die folgenden dynamischen Rechte.

```
'APPLICATION_PASSWORD_ADMIN',
'CONNECTION_ADMIN',
'REPLICATION_APPLIER',
'ROLE_ADMIN',
'SESSION_VARIABLES_ADMIN',
'SET_USER_ID',
'XA_RECOVER_ADMIN'
```

Das folgende Beispielskript gewährt den Benutzerkonten im Aurora MySQL-DB-Cluster die unterstützten dynamischen Rechte.

```
-- This script finds the user accounts that have Aurora MySQL supported dynamic
 privileges
-- and grants them to corresponding user accounts in the Aurora MySQL DB cluster.

/home/ec2-user/opt/mysql/8.0.26/bin/mysql -uusername -pxxxxx -P8026 -h127.0.0.1 -BNe
 "SELECT
  CONCAT('GRANT ', GRANTS, ' ON *.* TO ', GRANTEE ,';') AS grant_statement
  FROM (select GRANTEE, group_concat(privilege_type) AS GRANTS FROM
 information_schema.user_privileges
      WHERE privilege_type IN (
        'APPLICATION_PASSWORD_ADMIN',
        'CONNECTION_ADMIN',
        'REPLICATION_APPLIER',
        'ROLE_ADMIN',
        'SESSION_VARIABLES_ADMIN',
        'SET_USER_ID',
        'XA_RECOVER_ADMIN')
      AND GRANTEE NOT IN (\"'mysql.session'@'localhost'\",
\"'mysql.infoschema'@'localhost'\",\"'mysql.sys'@'localhost'\") GROUP BY GRANTEE)
      AS PRIVGRANTS; " | /home/ec2-user/opt/mysql/8.0.26/bin/mysql -u master_username -
p master_password -h DB_cluster_endpoint
```

Gespeicherte Objekte mit 'rdsadmin'@'localhost' als Definierer

Funktionen, Prozeduren, Ansichten, Ereignisse und Trigger mit `'rdsadmin'@'localhost'` als Definierer werden nicht importiert.

Mit dem folgenden SQL-Skript können die in Ihrer MySQL-Datenbank gespeicherten Objekte mit nicht unterstützten Definierern auflisten.

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.

SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
```

```
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

Logische Migration von MySQL zu Amazon Aurora MySQL mithilfe von mysqldump

Da Amazon Aurora MySQL eine MySQL-kompatible Datenbank ist, können Sie mit dem Dienstprogramm `mysqldump` Daten aus Ihrer MySQL-Datenbank oder mit dem Dienstprogramm `mariadb-dump` Daten aus Ihrer MariaDB-Datenbank in einen vorhandenen DB-Cluster von Aurora MySQL kopieren.

Informationen, wie Sie dies für sehr große MySQL- oder MariaDB-Datenbanken durchführen, finden Sie unter den folgenden Themen im Benutzerhandbuch zu Amazon Relational Database Service:

- MySQL – Importieren von Daten in eine Datenbank von Amazon RDS für MySQL mit reduzierter Ausfallzeit
- MariaDB – Importieren von Daten in eine Datenbank von Amazon RDS für MariaDB mit reduzierter Ausfallzeit

Informationen zu MySQL- oder MariaDB-Datenbanken mit kleineren Datenmengen finden Sie unter den folgenden Themen im Benutzerhandbuch zu Amazon Relational Database Service:

- MySQL – Importieren von Daten aus einer externen MySQL-Datenbank in eine DB-Instance von Amazon RDS für MySQL
- MariaDB – Importieren von Daten aus einer externen MariaDB-Datenbank in eine DB-Instance von Amazon RDS für MariaDB

Migrieren von Daten aus einer DB-Instance von RDS für MySQL in einen DB-Cluster von Amazon Aurora MySQL

Sie können Daten aus einer DB-Instance von RDS für MySQL in einen DB-Cluster von Amazon Aurora MySQL migrieren (kopieren).

Themen

- Migrieren eines RDS für MySQL-Snapshots zu Aurora
- Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat)

> **Note**
>
> Da Amazon Aurora MySQL mit MySQL kompatibel ist, können Sie Daten aus einer MySQL-Datenbank migrieren, indem Sie eine Replikation zwischen Ihrer MySQL-Datenbank und einem Amazon Aurora MySQL-DB-Cluster einrichten. Weitere Informationen finden Sie unter Replikation mit Amazon Aurora.

Migrieren eines RDS für MySQL-Snapshots zu Aurora

Sie können einen DB-Snapshot einer RDS-MySQL-DB-Instance migrieren, um einen Aurora-MySQL-DB-Cluster zu erstellen. Der neue Aurora-MySQL-DB-Cluster wird mit den Daten der ursprünglichen RDS für MySQL-DB-Instance gefüllt. Der DB-Snapshot muss von einer DB-Instance von Amazon RDS erstellt worden sein, die eine mit Aurora MySQL kompatible MySQL-Version ausführt.

Sie können einen manuell oder automatisch erstellten DB-Snapshot migrieren. Nachdem der DB-Cluster erstellt wurde, können Sie optional Aurora-Replikate erstellen.

> **Note**
>
> Sie können eine MySQL-DB-Instance zu einem Aurora MySQL-DB-Cluster auch migrieren, indem Sie zunächst eine Aurora Read Replica der MySQL-Quell-DB-Instance erstellen. Weitere Informationen finden Sie unter Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat).

Sie können von einigen älteren Versionen von MySQL 8.0, einschließlich 8.0.11, 8.0.13 und 8.0.15, nicht zu Aurora MySQL Version 3.05 und höher migrieren. Wir empfehlen, vor der Migration auf MySQL Version 8.0.28 zu aktualisieren.

Gehen Sie wie folgt vor:

1. Bestimmen Sie die Menge an Speicher, die für den Aurora MySQL-DB-Cluster bereitgestellt werden soll. Weitere Informationen finden Sie unter Wie viel Speicherplatz benötige ich?
2. Verwenden Sie die Konsole zum Erstellen des Snapshots in der AWS-Region, in der sich die Amazon-RDS-MySQL-Instance befindet. Informationen zum Erstellen von DB-Snapshots finden Sie unter Erstellen eines DB-Snapshots.
3. Wenn sich der DB-Snapshot nicht in derselben AWS-Region wie der DB-Cluster befindet, kopieren Sie ihn mit der Amazon-RDS-Konsole in diese AWS-Region. Informationen zum Kopieren von DB-Snapshots finden Sie unter Kopieren eines DB-Snapshots.
4. Verwenden Sie die Konsole, um den DB-Snapshot zu migrieren und einen Aurora MySQL-DB-Cluster mit denselben Datenbanken wie die ursprüngliche MySQL-DB-Instance zu erstellen.

⚠ Warning

Amazon RDS beschränkt jedes AWS-Konto auf jeweils eine Snapshot-Kopie in jede AWS-Region.

Wie viel Speicherplatz benötige ich?

Wenn Sie den Snapshot einer MySQL-DB-Instance zu einem Aurora MySQL-DB-Cluster migrieren, verwendet Aurora ein Amazon Elastic Block Store (Amazon EBS)-Volume, um die Daten aus dem Snapshot vor der Migration zu formatieren. In manchen Fällen ist zusätzlicher Speicherplatz erforderlich, um die Daten für die Migration zu formatieren.

Tabellen, die keine MyISAM-Tabellen und nicht komprimiert sind, können bis zu 16 TB groß sein. Wenn Sie MyISAM-Tabellen besitzen, muss Aurora zusätzlichen Speicherplatz im Volume verwenden, um die Tabellen so zu konvertieren, dass sie mit Aurora MySQL kompatibel sind. Wenn Sie komprimierte Tabellen besitzen, muss Aurora zusätzlichen Speicherplatz im Volume verwenden, um diese Tabellen zu erweitern, bevor sie im Aurora-Cluster-Volume gespeichert werden. Aufgrund dieses zusätzlichen Platzbedarfs sollten Sie sicherstellen, dass keine der MyISAM-Tabellen und

der komprimierten Tabellen, die von Ihrer MySQL-DB-Instance migriert werden, die Größe von 8 TB übersteigt.

Reduzieren des erforderlichen Speicherplatzes für die Migration von Daten zu Amazon Aurora MySQL

Sie können bei Bedarf das Datenbankschema vor dem Migrieren nach Amazon Aurora ändern. Eine solche Änderung kann in den folgenden Fällen hilfreich sein:

- Sie möchten die Migration beschleunigen.
- Sie wissen nicht genau, wie viel Speicherplatz bereitgestellt werden muss.
- Ein Migrationsversuch ist fehlgeschlagen, weil nicht genügend Speicherplatz verfügbar war.

Sie können die folgenden Änderungen vornehmen, um das Migrieren einer Datenbank zu Amazon Aurora zu optimieren.

> **Important**
>
> Führen Sie diese Aktualisierungen für die neue DB-Instance durch, die aus einem Snapshot einer Produktionsdatenbank wiederhergestellt wurde, statt eine Produktions-Instance zu verwenden. Anschließend können Sie die Daten aus dem Snapshot der neuen DB-Instance in den Aurora-DB-Cluster migrieren, um Service-Unterbrechungen bei der Produktionsdatenbank zu vermeiden.

Tabellentyp	Einschränkung oder Richtlinie
MyISAM-Tabellen	Aurora MySQL unterstützt ausschließlich InnoDB-Tabellen. Wenn die Datenbank MyISAM-Tabellen enthält, müssen diese vor der Migration nach Aurora MySQL konvertiert werden. Während der Migration wird für die Konvertierung von MyISAM nach InnoDB zusätzlicher Speicherplatz benötigt. Um das Risiko zu senken, dass der Speicherplatz nicht ausreicht , oder um den Migrationsvorgang zu beschleunigen, wandeln Sie alle MyISAM-Tabellen vor der Migration in InnoDB-Tabellen um. Die Größe der resultierenden InnoDB-Tabelle entspricht

Tabellentyp	Einschränkung oder Richtlinie
	den Größenanforderungen von Aurora MySQL für diese Tabelle. Führen Sie zum Umwandeln einer MyISAM-Tabelle in eine InnoDB-Tabelle den folgenden Befehl aus: `alter table <schema>.<table_name> engine=innodb, algorithm=copy;`
Komprimierte Tabellen	Aurora MySQL unterstützt keine komprimierten Tabellen (d. h. mit erstellte Tabelle `ROW_FORMAT=COMPRESSED`). Um das Risiko zu senken, dass der Speicherplatz nicht ausreicht , oder um den Migrationsvorgang zu beschleunigen, erweitern Sie komprimierte Tabellen, indem Sie `ROW_FORMAT` auf `DEFAULT`, `COMPACT`, `DYNAMIC` oder `REDUNDANT` einstellen. Weitere Informationen finden Sie unter InnoDB row formats in der MySQL-Dokumentation.

Sie können mit dem folgenden SQL-Skript alle Datenbanktabellen der MySQL-DB-Instance auflisten, die MyISAM-Tabellen oder komprimierte Tabellen sind.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
```

```
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

Das Skript produziert eine Ausgabe wie im folgenden Beispiel. Das Beispiel zeigt zwei Tabellen, die vom MyISAM- in das InnoDB-Format umgewandelt werden müssen. Außerdem wird die ungefähre Größe jeder Tabelle in Megabyte (MB) angezeigt.

```
+---------------------------------+------------------+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+---------------------------------+------------------+
```

```
| test.name_table                 |          2102.25 |
| test.my_table                   |            65.25 |
+---------------------------------+------------------+
2 rows in set (0.01 sec)
```

Migrieren eines DB-Snapshots von RDS für MySQL zu einem Aurora-MySQL-DB-Cluster

Sie können einen DB-Snapshot einer DB-Instance von RDS für MySQL mit der AWS-Managementkonsole oder der AWS CLI migrieren, um einen Aurora-MySQL-DB-Cluster zu erstellen. Der neue Aurora-MySQL-DB-Cluster wird mit den Daten der ursprünglichen RDS für MySQL-DB-Instance gefüllt. Informationen zum Erstellen von DB-Snapshots finden Sie unter Erstellen eines DB-Snapshots.

Wenn sich der DB-Snapshot nicht in der AWS-Region befindet, in der Sie die Daten speichern möchten, kopieren Sie den DB-Snapshot in die betreffende AWS-Region. Informationen zum Kopieren von DB-Snapshots finden Sie unter Kopieren eines DB-Snapshots.

Konsole

Wenn Sie den DB-Snapshot mithilfe der AWS-Managementkonsole migrieren, nimmt die Konsole die erforderlichen Aktionen vor, um ausschließlich den DB-Cluster zu erstellen.

Sie können sich auch dafür entscheiden, dass Ihr neuer Aurora MySQL DB-Cluster im Ruhezustand verschlüsselt wird, indem Sie eine AWS KMS key.

So migrieren Sie einen MySQL -DB-Snapshot mithilfe der AWS-Managementkonsole:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Starten Sie die Migration entweder von der MySQL DB-Instance oder vom Snapshot aus:

   Um die Migration von der DB-Instance aus zu starten:

   1. Wählen Sie im Navigationsbereich Databases (Datenbanken) und anschließend die MySQL-DB-Instance aus.
   2. Wählen Sie unter Actions (Aktionen) die Option Migrate latest snapshot (Aktuellen Snapshot migrieren) aus.

   Um die Migration vom Snapshot aus zu starten:

1. Klicken Sie auf Snapshots (Snapshots).
2. Wählen Sie auf der Seite Snapshots (Snapshots) den Snapshot aus, den Sie zu einem Aurora MySQL-DB-Cluster migrieren möchten.
3. Wählen Sie die Option Snapshot-Aktionen und anschließend Snapshot migrieren.

Die Seite Datenbank migrieren wird angezeigt.

3. Legen Sie auf der Seite Migrate Database (Datenbank migrieren) Folgendes fest:

- Migrate to DB Engine (Zu DB-Engine migrieren): Wählen Sie `aurora` aus.
- DB Engine Version (DB-Engine-Version): Wählen Sie die DB-Engine-Version für das Aurora MySQL-DB-Cluster aus.
- DB-Instance-Klasse: Wählen Sie eine DB-Instance-Klasse, die über ausreichend Speicher und Kapazität für Ihre Datenbank verfügt, z. B. `db.r3.large`. Aurora-Cluster-Volumes nehmen automatisch an Größe zu, wenn die Datenmenge in Ihrer Datenbank zunimmt. Ein Aurora-Cluster-Volume kann auf eine maximale Größe von 128 tebibytes (TiB) anwachsen. Sie müssen daher nur eine DB-Instance-Klasse auswählen, die den aktuellen Speicheranforderungen entspricht. Weitere Informationen finden Sie unter Übersicht über Amazon-Aurora-Speicher.
- DB Instance Identifier (DB-Instance-Kennung): Geben Sie einen Namen für den DB-Cluster ein. Dieser muss im Konto für die ausgewählte AWS-Region eindeutig sein. Dieser Bezeichner wird in den Endpunktadressen für die Instances im DB-Cluster verwendet. Sie können den Namen aussagekräftiger machen, indem Sie Angaben wie AWS-Region und DB-Engine hinzufügen, z. B. **`aurora-cluster1`**.

  Für den DB-Instance-Bezeichner gelten folgende Einschränkungen:

  - Sie darf 1 bis 63 alphanumerische Zeichen oder Bindestriche enthalten.
  - Das erste Zeichen muss ein Buchstabe sein.
  - Er darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.
  - Es muss für alle DB-Instances pro AWS-Konto und AWS-Region eindeutig sein.
- Virtual Private Cloud (VPC): Wenn bereits eine VPC vorhanden ist, können Sie diese VPC mit dem Aurora MySQL-DB-Cluster verwenden, indem Sie die VPC-ID auswählen, beispielsweise `vpc-a464d1c1`. Weitere Informationen zum Erstellen einer VPC finden Sie unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).

Andernfalls können Sie von Aurora eine neue VPC erstellen lassen, indem Sie auf Neue VPC erstellen klicken.

- DB-Subnetzgruppe: Wenn bereits eine Subnetzgruppe vorhanden ist, können Sie diese Subnetzgruppe mit dem Aurora-MySQL-DB-Cluster verwenden, indem Sie die Subnetzgruppen-ID auswählen, beispielsweise `gs-subnet-group1`.

  Andernfalls können Sie von Aurora eine neue Subnetzgruppe erstellen lassen, indem Sie auf Neue Subnetzgruppe erstellen klicken.
- Öffentliche Zugänglichkeit: Wählen Sie Nein aus, um anzugeben, dass nur Ressourcen innerhalb Ihrer VPC auf Instances im DB-Cluster zugreifen dürfen. Wählen Sie Ja aus, um anzugeben, dass Ressourcen im öffentlichen Netzwerk auf Instances im DB-Cluster zugreifen dürfen. Die Standardeinstellung lautet Ja.

  > **Note**
  >
  > Der Produktions-DB-Cluster muss sich nicht unbedingt in einem öffentlichen Subnetz befinden, da nur die Anwendungsserver Zugriff auf ihren DB-Cluster benötigen. Wenn sich das DB-Cluster nicht in einem öffentlichen Subnetz befinden muss, legen Sie Publicly Accessible (Öffentlich zugänglich) auf No (Nein) fest.

- Availability Zone (Availability Zone): Wählen Sie die Availability Zone aus, in der die primäre Instance für Ihr Aurora-MySQL-DB-Cluster gehostet werden soll. Wenn die Availability Zone durch Aurora ausgewählt werden soll, wählen Sie No Preference (Keine Präferenz) aus.
- Database Port (Datenbankport): Geben Sie den Standardport ein, der beim Verbinden der Instances im Aurora MySQL-DB-Cluster verwendet werden soll. Der Standardwert ist `3306`.

  > **Note**
  >
  > Möglicherweise lässt die Firewall eines Unternehmens den Zugriff auf Standardports, wie z. B. den MySQL-Standardport 3306, nicht zu. Geben Sie in diesem Fall einen Port ein, der von der Firewall Ihres Unternehmens zugelassen wird. Sie benötigen diesen Port-Wert später, wenn Sie eine Verbindung mit dem Aurora MySQL-DB-Cluster herstellen.

- Verschlüsselung: Wählen Sie Verschlüsselung aktivieren aus, um Ihren neuen Aurora MySQL-DB-Cluster im Ruhezustand zu verschlüsseln. Wenn Sie Verschlüsselung aktivieren wählen, müssen Sie einen KMS-Schlüssel als Wert AWS KMS key wählen.

  Wenn Ihr DB-Snapshot nicht verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster in Ruhe verschlüsselt wird.

  Wenn Ihr DB-Snapshot verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster in Ruhe mit dem angegebenen Verschlüsselungsschlüssel verschlüsselt wird. Sie können den vom DB-Snapshot verwendeten Verschlüsselungsschlüssel oder einen anderen Schlüssel angeben. Aus einem verschlüsselten DB-Snapshot kann kein unverschlüsselter DB-Cluster erstellt werden.
- Auto Minor Version Upgrade (Automatisches Unterversions-Upgrade): Diese Einstellung ist für Aurora MySQL-DB-Cluster nicht relevant.

  Weitere Informationen über Engine-Updates für Aurora MySQL finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.

4. Wählen Sie Migrate (Migrieren) aus, um den DB-Snapshot zu migrieren.
5. Klicken Sie auf Instances und danach auf das Pfeilsymbol, um die DB-Cluster-Details anzuzeigen und den Fortschritt der Migration zu überwachen. Auf der Detailseite wird der Cluster-Endpunkt zum Herstellen einer Verbindung mit der primären Instance des DB-Clusters angezeigt. Weitere Informationen zum Herstellen einer Verbindung mit einem Aurora-MySQL-DB-Cluster finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

AWS CLI

Sie können einen Aurora-DB-Cluster aus einem DB-Snapshot einer RDS für MySQL-DB-Instance erstellen, indem Sie den `restore-db-cluster-from-snapshot`-Befehl mit folgenden Parametern verwenden:

- `--db-cluster-identifier` – Der Name des zu erstellenden DB-Clusters.
- `--engine aurora-mysql` – Für einen mit MySQL 5.7 oder 8.0 kompatiblen DB-Cluster
- `--kms-key-id` – DerAWS KMS key zur optionalen Verschlüsselung des DB-Clusters, je nachdem, ob Ihr DB-Snapshot verschlüsselt ist.
  - Wenn Ihr DB-Snapshot nicht verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster in Ruhe verschlüsselt wird. Andernfalls wird Ihr DB-Cluster nicht verschlüsselt.

- Wenn Ihr DB-Snapshot verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster in Ruhe mit dem angegebenen Verschlüsselungsschlüssel verschlüsselt wird. Ansonsten wird Ihr DB-Cluster im Ruhezustand mit dem Verschlüsselungsschlüssel für den DB-Snapshot verschlüsselt.

  > **Note**
  >
  > Aus einem verschlüsselten DB-Snapshot kann kein unverschlüsselter DB-Cluster erstellt werden.

- `--snapshot-identifier` – Der Amazon-Ressourcenname (ARN) des zu migrierenden DB-Snapshots. Weitere Informationen zu Amazon-RDS-ARNs finden Sie unter Amazon Relational Database Service (Amazon RDS).

Wenn Sie den DB-Snapshot mit dem Befehl `RestoreDBClusterFromSnapshot` migrieren, erstellt der Befehl sowohl den DB-Cluster als auch die primäre Instance.

In diesem Beispiel erstellen Sie ein mit MySQL 5.7 kompatibles DB-Cluster mit dem Namen *`mydbcluster`* aus einem DB-Snapshot, dessen ARN auf *`mydbsnapshotARN`* festgelegt wurde.

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Für Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

In diesem Beispiel erstellen Sie ein mit MySQL 5.7 kompatibles DB-Cluster mit dem Namen *`mydbcluster`* aus einem DB-Snapshot, dessen ARN auf *`mydbsnapshotARN`* festgelegt wurde.

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Für Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat)

Aurora verwendet die binäre Protokoll-Replizierungsfunktionalität der MySQL-DB-Engine, um für eine MySQL-Quell-DB-Instance einen speziellen Typ von DB-Cluster zu erstellen, der als „Aurora Read Replica“ bezeichnet wird. Aktualisierungen, die an der RDS-für-MySQL-DB-Quell-Instance vorgenommen werden, werden asynchron in diese Aurora Read Replica repliziert.

Es wird empfohlen, diese Funktionalität zu verwenden, um eine Migration aus einer RDS-für-MySQL-DB-Instance zu einem Aurora-MySQL-DB-Cluster durchzuführen, indem Sie eine Aurora Read Replica Ihrer MySQL-DB-Quell-Instance erstellen. Wenn die Replikationsverzögerung zwischen der RDS-für-MySQL-DB-Instance und der Aurora Read Replica 0 beträgt, können Sie die Clientanwendungen für den Zugriff auf die Aurora Read Replica konfigurieren und dann die Replikation stoppen, damit die Aurora Read Replica als eigenständiger Aurora MySQL-DB-Cluster verwendet wird. Beachten Sie, dass die Migration sehr lange dauern kann. Pro Tebibyte (TiB) Daten werden mehrere Stunden benötigt.

Eine Liste der Regionen, in denen Aurora erhältlich ist, finden Sie unter Amazon Aurora in der Allgemeine AWS-Referenz.

Wenn Sie eine Aurora Read Replica einer RDS-für-MySQL-DB-Instance erstellen, wird von Amazon RDS automatisch ein DB-Snapshot der RDS für MySQL-DB-Quell-Instance erstellt (privat für Amazon RDS und kostenlos). Anschließend migriert Amazon RDS die Daten vom DB-Snapshot zum Aurora-Lesereplikat. Nach dem Migrieren der Daten des DB-Snapshots zum neuen Aurora-MySQL-DB-Cluster startet Amazon RDS die Replikation zwischen Ihrer RDS-für-MySQL-DB-Instance und dem Aurora MySQL-DB-Cluster. Wenn die RDS-für-MySQL-DB-Instance Tabellen enthält, die andere Speicher-Engines als InnoDB oder ein komprimiertes Zeilenformat verwenden, können Sie die Erstellung einer Aurora Read Replica beschleunigen, indem Sie diese Tabellen für die Verwendung der InnoDB-Speicher-Engine und des dynamischen Zeilenformats konfigurieren, bevor Sie die Aurora Read Replica erstellen. Weitere Informationen zum Kopieren eines MySQL-DB-Snapshots zu einem Aurora MySQL-DB-Cluster finden Sie unter Migrieren von Daten aus einer DB-Instance von RDS für MySQL in einen DB-Cluster von Amazon Aurora MySQL.

Sie können nur eine Aurora-Read Replica für eine RDS-für-MySQL-DB-Instance erstellen.

> **Note**
>
> Aufgrund von Funktionsunterschieden zwischen Aurora-MySQL und der MySQL-Datenbank-Engine-Version der als Replikationsprimäre dienenden RDS-für-MySQL-DB-Instance kann es

bei der Replikation zu Problemen kommen. Bei Problemen können Sie Fragen im Amazon-RDS-Community-Forum stellen oder sich an AWS -Support wenden.
Sie können keine Aurora Read Replica erstellen, wenn Ihre RDS-für-MySQL-DB-Instance bereits die Quelle für eine regionsübergreifende Read Replica enthält.
Sie können von einigen älteren Versionen von RDS für MySQL 8.0, einschließlich 8.0.11, 8.0.13 und 8.0.15, nicht zu Aurora MySQL Version 3.05 und höher migrieren. Wir empfehlen, vor der Migration auf RDS für MySQL Version 8.0.28 zu aktualisieren.

Weitere Informationen zu MySQL-Lesereplikaten finden Sie unter Arbeiten mit Read Replicas von MariaDB-, MySQL- und PostgreSQL-DB-Instances.

Erstellen einer Aurora Read Replica

Sie können eine Aurora Read Replica für eine RDS-für-MySQL-DB-Instance über die Konsole, die AWS CLI oder die RDS-API erstellen.

Konsole

So erstellen Sie eine Aurora Read Replica einer RDS-für-MySQL-DB-Instance

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie die MySQL DB-Instance aus, die Sie als Quelle für Ihr Aurora Read Replica verwenden möchten.
4. Wählen Sie für Actions (Aktionen) Create Aurora read replica (Aurora Read Replica erstellen) aus.
5. Legen Sie die in der folgenden Tabelle beschriebenen DB-Cluster-Einstellungen für die Aurora-Read Replica fest.

Option	Beschreibung
DB-Instance-Klasse	Wählen Sie eine DB-Instance-Klasse aus, die die Verarbeitungs- und Speicheranforderungen der primären Instance im DB-Cluster definiert. Weitere Informationen zu den Optionen für DB-Instance-Klasse

Option	Beschreibung
	n finden Sie unter DB-Instance-Klassen von Amazon Aurora.
Multi-AZ-Bereitstellung	Wählen Sie Replika in einer anderen Zone erstellen aus, um eine Standby Replica des neuen DB-Cluste rs in einer anderen Availability Zone für eine Failover-Unterstützung in der AWS-Zielregion zu erstellen. Weitere Informationen über die Multi-AZ-Bereitstellung finden Sie unter Regionen und Availability Zones.
DB-Instance-Kennung	Geben Sie einen Namen für die primäre Instance des DB-Clusters der Aurora-Read Replica ein. Dieser Bezeichner wird in der Endpunktadresse für die primäre Instance des neuen DB-Clusters verwendet. Für den DB-Instance-Bezeichner gelten folgende Einschränkungen: • Sie darf 1 bis 63 alphanumerische Zeichen oder Bindestriche enthalten. • Das erste Zeichen muss ein Buchstabe sein. • Er darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten. • Sie muss bei allen DB-Instances für jedes AWS-Konto und jede AWS-Region eindeutig sein. Da das Aurora Read Replica-DB-Cluster aus einem Snapshot der DB-Quell-Instance erstellt wird, sind der Master-Benutzername und das Master-Passwort für die Aurora Read Replica mit dem Master-Benutzernam en und Master-Passwort für die DB-Quell-Instance identisch.

Option	Beschreibung
Virtual Private Cloud (VPC)	Wählen Sie die VPC aus, die als Host für das DB-Cluster dienen soll. Wählen Sie Create a new VPC (Neue VPC erstellen) aus, wenn Aurora eine VPC für Sie erstellen soll. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.
DB-Subnetzgruppe	Wählen Sie die DB-Subnetzgruppe aus, die für den DB-Cluster verwendet werden soll. Wählen Sie Create a new DB subnet group (Neue DB-Subnetzgruppe erstellen), wenn Aurora eine DB-Subnetzgruppe für Sie erstellen soll. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.
Öffentliche Zugänglichkeit	Wählen Sie `Yes` aus, um dem DB-Cluster eine öffentliche IP-Adresse zu geben. Wählen Sie andernfalls `No` aus. Die Instances in Ihrem DB-Cluster können eine Mischung aus öffentlichen und privaten DB-Instances sein. Weitere Informationen darüber, wie Sie den öffentlichen Zugriff für Instances deaktivieren, finden Sie unter Ausblenden einer DB-Clusters in einer VPC vor dem Internet.
Availability Zone	Legen Sie fest, ob Sie eine bestimmte Availability Zone angeben möchten. Weitere Informationen über Availability Zones finden Sie unter Regionen und Availability Zones.
VPC-Sicherheitsgruppe (Firewall)	Wählen Sie Neue VPC-Sicherheitsgruppe erstellen aus, damit Aurora eine VPC-Sicherheitsgruppe für Sie erstellt. Wählen Sie Vorhandene VPC-Sicherheitsgruppen auswählen aus, um eine oder mehrere VPC-Sicherheitsgruppen auszuwählen und den Netzwerkzugriff auf das DB-Cluster zu schützen. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.

Option	Beschreibung
Datenbankport	Geben Sie den Port an, über den Anwendungen und Dienstprogramme auf die Datenbank zugreifen können. Aurora-MySQL;-DB-Cluster verwenden standardmäßig den MySQL-Standardport 3306. Die Firewalls einiger Unternehmen blockieren Verbindungen mit diesem Port. Wenn die Firewall Ihres Unternehmens den Standardport blockiert, wählen Sie einen anderen Port für den neuen DB-Cluster aus.
DB-Parametergruppe	Wählen Sie eine DB-Parametergruppe für den Aurora MySQL DB-Cluster aus. Aurora verfügt über eine DB-Standardparametergruppe, die Sie verwenden können, oder Sie können Ihre eigene DB-Parame tergruppe erstellen. Weitere Informationen zu DB-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.
DB-Cluster-Parametergruppe	Wählen Sie eine DB-Cluster-Parametergruppe für den Aurora-MySQL-DB-Cluster aus. Aurora verfügt über eine DB-Cluster-Standardparametergruppe, die Sie verwenden können, oder Sie können Ihre eigene DB-Cluster-Parametergruppe erstellen. Weitere Informati onen zu DB-Cluster-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Option	Beschreibung
Verschlüsselung	Wählen Sie Verschlüsselung deaktivieren aus, wenn das neue Aurora-DB-Cluster nicht verschlüsselt werden soll. Wählen Sie Verschlüsselung aktivieren aus, wenn das neue Aurora-DB-Cluster im Ruhezustand verschlüsselt werden soll. Wenn Sie Verschlüsselung aktivieren wählen, müssen Sie einen KMS-Schlüssel als Wert AWS KMS key wählen. Wenn Ihre MySQL-DB-Instance nicht verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster im Ruhezustand verschlüsselt wird. Wenn Ihre MySQL-DB-Instance verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster in Ruhe mit dem angegebenen Verschlüsselungsschlüssel verschlüsselt wird. Sie können den von der MySQL DB-Instance verwendeten Verschlüsselungsschlüssel oder einen anderen Schlüssel angeben. Aus einer verschlüsselten MySQL-DB-Instance kann kein unverschlüsselter DB-Cluster erstellt werden.
Priorität	Wählen Sie eine Failover-Priorität für den DB-Cluster aus. Wenn Sie keinen Wert auswählen, wird als Standard tier-1 eingestellt. Diese Priorität bestimmt die Reihenfolge, in der Aurora-Replikate bei der Wiederherstellung nach einem Ausfall der primären Instance hochgestuft werden. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.
Aufbewahrungszeitraum für Backups	Wählen Sie den Aufbewahrungszeitraum (1 bis 35 Tage) für Sicherungskopien der Datenbank in Aurora aus. Sicherungskopien können für sekundengenaue Point-in-Time-Wiederherstellungen (PITR) der Datenbank verwendet werden.

Option	Beschreibung
Verbesserte Überwachung	Wählen Sie Erweiterte Überwachung aktivieren aus, um die Erfassung von Metriken in Echtzeit für das Betriebssystem zu aktivieren, in dem Ihr DB-Cluster ausgeführt wird. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).
Überwachungsrolle	Nur verfügbar, wenn Verbesserte Überwachung auf Erweiterte Überwachung aktivieren gesetzt ist. Wählen Sie die IAM-Rolle aus, die Sie erstellt haben, damit Aurora für Sie mit Amazon CloudWatch Logs kommunizieren kann, oder wählen Sie Standard aus, damit Aurora eine Rolle namens `rds-monitoring-role` für Sie erstellt. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).
Granularität	Nur verfügbar, wenn Erweiterte Überwachung auf Erweiterte Überwachung aktivieren gesetzt ist. Mit ihr können Sie die Zeitspanne zwischen den Erfassungen der Kennzahlen des DB-Clusters in Sekunden festlegen.
Kleinere Versions-Updates automatisch aktivieren	Diese Einstellung wird für Aurora MySQL-DB-Cluster nicht verwendet. Weitere Informationen über Engine-Updates für Aurora MySQL finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.
Wartungsfenster	Wählen Sie Fenster auswählen aus und geben Sie den wöchentlichen Zeitraum an, in dem Systemwartungen durchgeführt werden können. Sie können auch Keine Präferenz auswählen, damit Aurora einen Zeitraum nach dem Zufallsprinzip auswählt.

6. Wählen Sie Read Replica erstellen aus.

AWS CLI

Um eine Aurora Read Replica aus einer RDS-für-MySQL-Quell-DB-Instance zu erstellen, verwenden Sie die AWS CLI-Befehle `create-db-cluster` und `create-db-instance`, um einen neuen Aurora-MySQL-DB-Cluster zu erstellen. Geben Sie beim Aufrufen des Befehls `create-db-cluster` mit dem Parameter `--replication-source-identifier` den Amazon-Ressourcennamen (ARN) der gewünschten MySQL-DB-Instance an. Weitere Informationen zu Amazon-RDS-ARNs finden Sie unter Amazon Relational Database Service (Amazon RDS).

Geben Sie nicht den Masterbenutzernamen, das Masterpasswort oder den Datenbanknamen an, da die Aurora-Read Replica diese Werte aus der MySQL-DB-Instance übernimmt.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Für Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Wenn Sie die Konsole verwenden, um eine Aurora Read Replica zu erstellen, erstellt Aurora automatisch die primäre Instance für die Aurora Read Replica Ihres DB-Clusters. Wenn Sie die AWS CLI zum Erstellen einer Aurora-Read Replica verwenden, müssen Sie die primäre Instance explizit für Ihren DB-Cluster anlegen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance.

Sie können eine primäre Instance für Ihr DB-Cluster erstellen, indem Sie den `create-db-instance`-Befehl AWS CLI mit folgenden Parametern verwenden.

- `--db-cluster-identifier`

Der Name Ihres DB-Clusters.

- `--db-instance-class`

  Der Name der DB-Instance-Klasse, die für Ihre primäre Instance verwendet werden soll.

- `--db-instance-identifier`

  Der Name Ihrer primären Instance.

- `--engine aurora`

In diesem Beispiel erstellen Sie eine primäre Instance mit dem Namen *myreadreplicainstance* für den DB-Cluster mit dem Namen *myreadreplicacluster* unter Verwendung der DB-Instance-Klasse, die in *myinstanceclass* angegeben ist.

Example

Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
    --engine aurora
```

Für Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier myreadreplicacluster ^
    --db-instance-class myinstanceclass ^
    --db-instance-identifier myreadreplicainstance ^
    --engine aurora
```

RDS-API

Um eine Aurora-Read Replica aus einer RDS-für-MySQL-DB-Quell-Instance zu erstellen, verwenden Sie die Amazon-RDS-API-Befehle `CreateDBCluster` und `CreateDBInstance`, um einen neuen Aurora-DB-Cluster und eine neue primäre Instance zu erstellen. Geben Sie nicht den Masterbenutzernamen, das Masterpasswort oder den Datenbanknamen an, da die Aurora Read Replica diese Werte aus der RDS-für-MySQL-DB-Instance übernimmt.

Sie können einen neuen Aurora-DB-Cluster für eine Aurora Read Replica aus einer RDS-für-MySQL-DB-Quell-Instance erstellen, indem Sie den Amazon-RDS-API-Befehl `CreateDBCluster` mit folgenden Parametern verwenden:

- `DBClusterIdentifier`

  Name des zu erstellenden DB-Clusters.

- `DBSubnetGroupName`

  Der Name der DB-Subnetzgruppe, die mit diesem DB-Cluster verknüpft werden soll.

- `Engine=aurora`
- `KmsKeyId`

  Die AWS KMS key, mit der der DB-Cluster optional verschlüsselt wird, je nachdem, ob Ihre MySQL-DB-Instance verschlüsselt ist.

  - Wenn Ihre MySQL-DB-Instance nicht verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster im Ruhezustand verschlüsselt wird. Ansonsten wird Ihr DB-Cluster im Ruhezustand mit dem Standard-Verschlüsselungsschlüssel für Ihr Konto verschlüsselt.
  - Wenn Ihre MySQL-DB-Instance verschlüsselt ist, geben Sie einen Verschlüsselungsschlüssel an, damit Ihr DB-Cluster in Ruhe mit dem angegebenen Verschlüsselungsschlüssel verschlüsselt wird. Ansonsten wird Ihr DB-Cluster im Ruhezustand mit dem Verschlüsselungsschlüssel für die MySQL-DB-Instance verschlüsselt.

  > **Note**
  >
  > Aus einer verschlüsselten MySQL-DB-Instance kann kein unverschlüsselter DB-Cluster erstellt werden.

- `ReplicationSourceIdentifier`

  Der Amazon Resource Name (ARN) für die Quell-MySQL DB-Instance. Weitere Informationen zu Amazon-RDS-ARNs finden Sie unter Amazon Relational Database Service (Amazon RDS).

- `VpcSecurityGroupIds`

  Die Liste der EC2-VPC-Sicherheitsgruppen, die mit diesem DB-Cluster verknüpft werden sollen.

In diesem Beispiel erstellen Sie ein DB-Cluster namens *myreadreplicacluster* aus einer MySQL-DB-Quell-Instance mit einem auf *mysqlmasterARN* festgelegten ARN, verbunden mit einer DB-Subnetzgruppe namens *mysubnetgroup* und einer VPC-Sicherheitsgruppe namens *mysecuritygroup*.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBCluster
    &DBClusterIdentifier=myreadreplicacluster
    &DBSubnetGroupName=mysubnetgroup
    &Engine=aurora
    &ReplicationSourceIdentifier=mysqlprimaryARN
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &VpcSecurityGroupIds=mysecuritygroup
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
    &X-Amz-Date=20150927T164851Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

Wenn Sie die Konsole verwenden, um eine Aurora Read Replica zu erstellen, erstellt Aurora automatisch die primäre Instance für die Aurora Read Replica Ihres DB-Clusters. Wenn Sie die AWS CLI zum Erstellen einer Aurora-Read Replica verwenden, müssen Sie die primäre Instance explizit für Ihren DB-Cluster anlegen. Die primäre Instance ist die erste in einem DB-Cluster erstellte Instance.

Sie können eine primäre Instance für Ihr DB-Cluster erstellen, indem Sie den Amazon-RDS-API-Befehl `CreateDBInstance` mit folgenden Parametern verwenden:

- `DBClusterIdentifier`

  Der Name Ihres DB-Clusters.
- `DBInstanceClass`

  Der Name der DB-Instance-Klasse, die für Ihre primäre Instance verwendet werden soll.
- `DBInstanceIdentifier`

  Der Name Ihrer primären Instance.

- `Engine=aurora`

In diesem Beispiel erstellen Sie eine primäre Instance mit dem Namen *`myreadreplicainstance`* für den DB-Cluster mit dem Namen *`myreadreplicacluster`* unter Verwendung der DB-Instance-Klasse, die in *`myinstanceclass`* angegeben ist.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBInstance
    &DBClusterIdentifier=myreadreplicacluster
    &DBInstanceClass=myinstanceclass
    &DBInstanceIdentifier=myreadreplicainstance
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-09-01
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
    &X-Amz-Date=20140424T194844Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

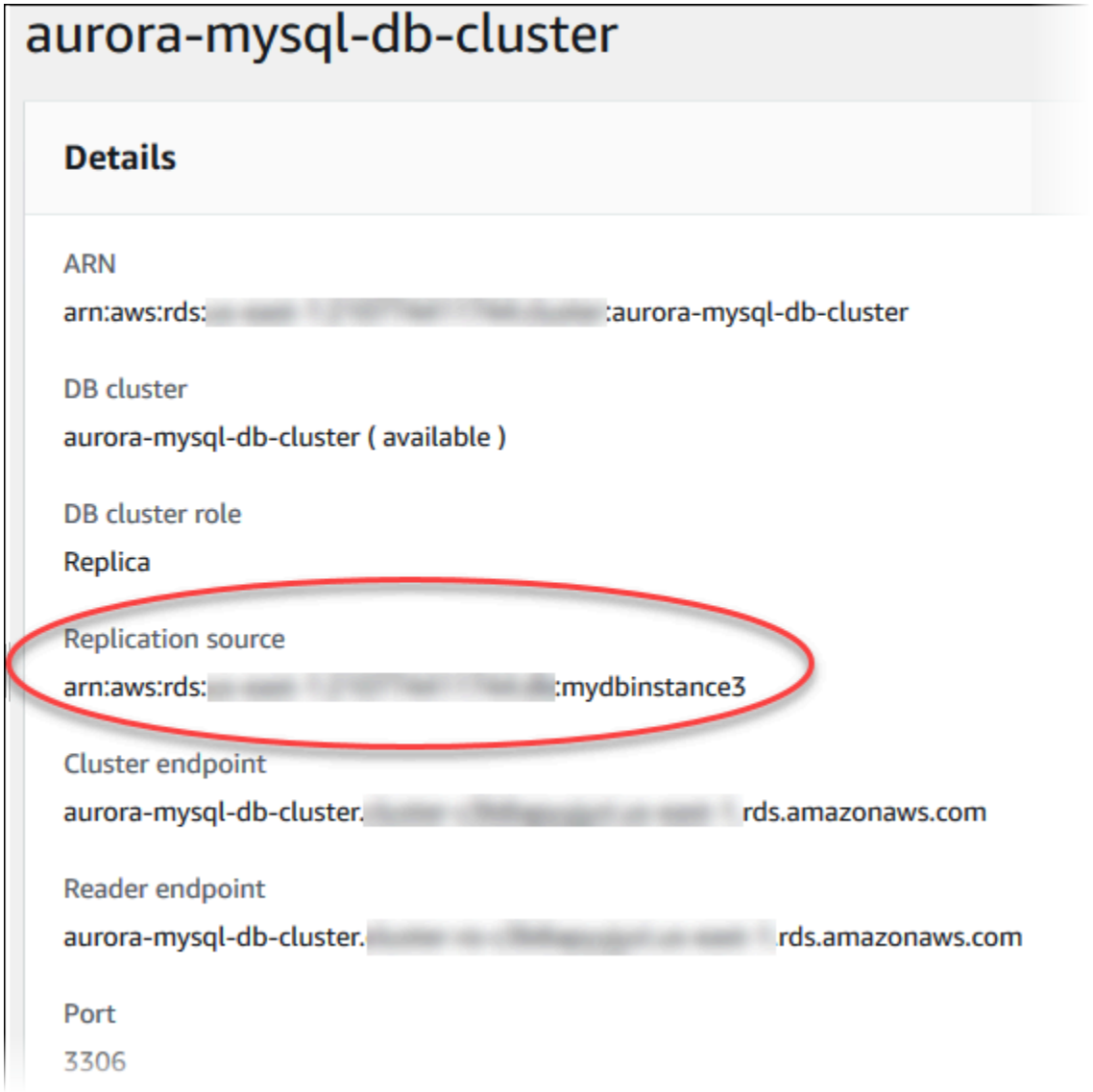
Anzeigen einer Aurora-Read Replica

Sie können die MySQL-Aurora MySQL-Replikationsbeziehungen für Ihre Aurora-MySQL-DB-Cluster über die AWS-Managementkonsole oder die AWS CLI anzeigen.

Konsole

So zeigen Sie die MySQL-DB-Primär-Instance für eine Aurora-Read Replica an:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie das DB-Cluster für die Aurora Read Replica aus, um Details anzuzeigen. Die Informationen zur MySQL-DB-Primäre-Instance werden im Feld Replication source (Replikationsquelle) angezeigt.



AWS CLI

Um die MySQL-Aurora MySQL-Replikationsbeziehungen für Ihre Aurora-MySQL-DB-Cluster über die AWS CLI anzuzeigen, verwenden Sie die Befehle `describe-db-clusters` und `describe-db-instances`.

Um zu ermitteln, welche MySQL-DB-Instance die Primär-Instance ist, verwenden Sie `describe-db-clusters` und geben die Cluster-ID der Aurora Read Replica für die Option `--db-cluster-identifier` an. Der ARN der als primäre Replikation verwendeten DB-Instance wird in der Befehlsausgabe mit dem Element `ReplicationSourceIdentifier` angezeigt.

Um zu ermitteln, welches DB-Cluster die Aurora Read Replica ist, verwenden Sie `describe-db-instances` und geben die Instance-ID der MySQL-DB-Instance für die Option `--db-instance-identifier` an. Informationen zur DB-Cluster-ID für die Aurora Read Replica finden Sie im Element `ReadReplicaDBClusterIdentifiers` in der Ausgabe.

Example

Für Linux, macOS oder Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
    --db-instance-identifier mysqlprimary
```

Für Windows:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
    --db-instance-identifier mysqlprimary
```

Hochstufen einer Aurora Read Replica

Nach Abschluss der Migration können Sie die Aurora Read Replica über die AWS-Managementkonsole oder die AWS CLI zu einem eigenständigen DB-Cluster hochstufen.

Anschließend können Sie Ihre Clientanwendungen für den Zugriff auf den Endpunkt der Aurora Read Replica konfigurieren. Weitere Informationen zu den Aurora-Endpunkten finden Sie unter Amazon-Aurora-Endpunktverbindungen. Die Hochstufung sollte ziemlich schnell erfolgen. Während des Vorgangs sind Lese- und Schreibzugriffe auf die Aurora-Read Replica möglich. Sie können allerdings in dieser Zeit weder die MySQL-DB-Primär-Instance löschen noch die Verknüpfung zwischen der DB-Instance und der Aurora-Read Replica aufheben.

Halten Sie vor der Hochstufung der Aurora Read Replica alle Schreibtransaktionen zur MySQL-DB-Quell-Instance an und warten Sie anschließend, bis die Replikationsverzögerung der Aurora Read Replica 0 erreicht. Sie können die Replikatverzögerung für eine Aurora-Read Replica anzeigen,

indem Sie den Befehl `SHOW SLAVE STATUS` (Aurora-MySQL-Version 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) auf Ihrem Aurora-Lesereplikat aufrufen. Überprüfen Sie den Wert Sekunden hinter Master.

Schreibzugriffe auf die Aurora-Read Replica sollten erst durchgeführt werden, nachdem die Schreibtransaktionen zur Primären angehalten wurden und die Replikationsverzögerung 0 beträgt. Wenn Sie zur Aurora Read Replica schreiben, bevor die Replikationsverzögerung null erreicht, und Tabellen geändert werden, die auch im primären MySQL geändert werden, könnte die Replikation zu Aurora unterbrochen werden. Falls dies geschieht, müssen Sie die Aurora-Read Replica löschen und erneut erstellen.

Konsole

So stufen Sie eine Aurora-Read Replica zu einem Aurora-DB-Cluster hoch:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie den DB-Cluster für das Aurora Read Replica aus.
4. Wählen Sie für Actions (Aktionen) Promote (Hochstufen) aus.
5. Klicken Sie auf Promote Read Replica) Read Replica hochstufen.

Bestätigen Sie nach der Werbung, dass das Hochstufen abgeschlossen wurde, indem Sie das folgende Verfahren anwenden.

Um zu bestätigen, dass das Aurora Read Replica hochgestuft wurde

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Events.
3. Stellen Sie auf der Seite Ereignisse sicher, dass für den Cluster, den Sie hochgestuft haben, ein `Promoted Read Replica cluster to a stand-alone database cluster` Ereignis vorliegt.

Nach Abschluss der Hochstufung wird die Verknüpfung zwischen der MySQL-DB-Primär-Instance und der Aurora-Read Replica aufgehoben. Sie können dann die DB-Instance bei Bedarf löschen.

AWS CLI

Um ein Aurora-Lesereplikat zu einem eigenständigen DB-Cluster hochzustufen, verwenden Sie den `promote-read-replica-db-cluster` AWS CLI-Befehl .

Example

Für Linux, macOS oder Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Für Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

Verwalten von Amazon Aurora MySQL

In den folgenden Abschnitten wird das Verwalten eines Amazon-Aurora-MySQL-DB-Clusters erläutert.

Themen

- Verwalten von Performance und Skalierung für Amazon Aurora MySQL
- Rückverfolgen eines Aurora-DB-Clusters
- Testen von Amazon Aurora MySQL unter Verwendung von Fehlersimulationsabfragen
- Ändern von Tabellen in Amazon Aurora mithilfe von Fast DDL
- Anzeigen des Volume-Status für einen Aurora MySQL-DB-Cluster

Verwalten von Performance und Skalierung für Amazon Aurora MySQL

Skalierung von Aurora MySQL-DB-Instances

Sie können Aurora MySQL-DB-Instances auf zwei Arten skalieren, durch Skalierung der Instance oder durch Skalierung der Lesevorgänge. Weitere Informationen zur Lese-Skalierung finden Sie unter Skalierung von Lesevorgängen.

Sie können Ihr Aurora-MySQL-DB-Cluster skalieren, indem Sie die DB-Instance-Klasse für jede DB-Instance im DB-Cluster ändern. Aurora MySQL unterstützt mehrere DB-Instance-Klassen, die für Aurora optimiert sind. Verwenden Sie keine db.t2- oder db.t3-Instance-Klassen für größere Aurora-Cluster mit einer Größe von mehr als 40 TB. Die Spezifikationen der von Aurora MySQL unterstützten DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter Verwendung von T-Instance-Klassen für Entwicklung und Tests.

Maximale Verbindungen zu einer Aurora MySQL-DB-Instance

Die maximale Anzahl der zulässigen Verbindungen mit einer Aurora MySQL-DB-Instance wird durch den Parameter `max_connections` in der Instance-Ebenen-Parametergruppe für die DB-Instance festgelegt.

Die folgende Tabelle listet den Standardwert von `max_connections` für jede DB-Instance auf, die für Aurora MySQL verfügbar ist. Sie können die maximal zulässige Anzahl von Verbindungen mit Ihrer Aurora MySQL-DB-Instance erhöhen, indem Sie die Instance auf eine DB-Instance-Klasse mit mehr Arbeitsspeicher skalieren oder einen größeren Wert für den Parameter `max_connections` in der DB-Parametergruppe für Ihre Instance festlegen, wobei ein Wert von bis zu 16 000 möglich ist.

> **Tip**
>
> Wenn Ihre Anwendungen häufig Verbindungen öffnen und schließen oder langlebige Verbindungen in großer Zahl offen lassen, empfehlen wir Ihnen, Amazon-RDS-Proxy zu verwenden. RDS-Proxy ist ein vollständig verwalteter, hochverfügbarer Datenbank-Proxy, der Datenbankverbindungen sicher und effizient per Verbindungspooling freigibt. Weitere Informationen zu RDS-Proxy finden Sie unter Amazon-RDS-Proxy für Aurora.

Weitere Informationen dazu, wie Aurora Serverless v2-Instances diesen Parameter behandeln, finden Sie unter Maximale Anzahl der Verbindungen für Aurora Serverless v2.

Instance class	Vorgabewert max_connections		
db.t2.small	45		
db.t2.Medium	90		
db.t3.small	45		
db.t3.medium	90		
db.t3.large	135		
db.t4g.medium	90		

Instance class	Vorgabewert max_connections		
db.t4g.large	135		
db.r3.large	1000		
db.r3.xlarge	2000		
db.r3.2xlarge	3000		
db.r3.4xlarge	4000		
db.r3.8xlarge	5000		
db.r4.large	1000		
db.r4.xlarge	2000		
db.r4.2xlarge	3000		
db.r4.4xlarge	4000		
db.r4.8xlarge	5000		
db.r4.16xlarge	6 000		
db.r5.large	1000		
db.r5.xlarge	2000		
db.r5.2xlarge	3 000		
db.r5.4xlarge	4000		
db.r5.8xlarge	5000		
db.r5.12xlarge	6 000		
db.r5.16xlarge	6 000		

Instance class	Vorgabewert max_connections		
db.r5.24xlarge	7000		
db.r6g.large	1000		
db.r6g.xlarge	2000		
db.r6g.2xlarge	3 000		
db.r6g.4xlarge	4000		
db.r6g.8xlarge	5000		
db.r6g.12xlarge	6 000		
db.r6g.16xlarge	6 000		
db.r6i.large	1000		
db.r6i.xlarge	2000		
db.r6i.2xlarge	3 000		
db.r6i.4xlarge	4000		
db.r6i.8xlarge	5000		
db.r6i.12xlarge	6 000		
db.r6i.16xlarge	6 000		
db.r6i.24xlarge	7000		
db.r6i.32xlarge	7000		
db.r7g.large	1000		
db.r7g.xlarge	2000		

Instance class	Vorgabewert max_connections
db.r7g.2xlarge	3 000
db.r7g.4xlarge	4000
db.r7g.8xlarge	5000
db.r7g.12xlarge	6 000
db.r7g.16xlarge	6 000
db.r7i.large	1000
db.r7i.xlarge	2000
db.r7i.2xlarge	3 000
db.r7i.4xlarge	4000
db.r7i.8xlarge	5000
db.r7i.12xlarge	6 000
db.r7i.16xlarge	6 000
db.r7i.24xlarge	7000
db.r7i.48xlarge	8000
db.r8g.large	1000
db.r8g.xlarge	2000
db.r8g.2xlarge	3 000
db.r8g.4xlarge	4000
db.r8g.8xlarge	5000

Instance class	Vorgabewert max_connections
db.r8g.12xlarge	6 000
db.r8g.16xlarge	6 000
db.r8g.24xlarge	7000
db.r8g.48xlarge	8000
db.x2g.large	2000
db.x2g.xlarge	3 000
db.x2g.2xlarge	4000
db.x2g.4xlarge	5000
db.x2g.8xlarge	6 000
db.x2g.12xlarge	7000
db.x2g.16xlarge	7000

Tip

Die `max_connections`-Parameterberechnung verwendet die Protokollbasis 2 (unterschiedlich zum natürlichen Logarithmus) und den `DBInstanceClassMemory`-Wert in Byte für die ausgewählte Instance-Klasse von Aurora MySQL. Der Parameter akzeptiert nur Ganzzahlwerte, wobei Dezimalstellen bei Berechnungen gekürzt werden. Die Formel implementiert Verbindungslimits wie folgt:

- Inkrement von 1 000 Verbindungen für größere R3-, R4- und R5-Instances
- Inkrement von 45 Verbindungen für T2- und T3-Instance-Speichervarianten

Beispiel: Für db.r6g.large berechnet die Formel zwar 1 069,2, das System implementiert aber 1 000, um konsistente inkrementelle Muster aufrechtzuerhalten.

Wenn Sie eine neue Parametergruppe erstellen, um einen eigenen Standardwert für den Verbindungsgrenzwert zu erstellen, werden Sie feststellen, dass der Standard-Verbindungsgrenzwert mithilfe einer auf dem Wert `DBInstanceClassMemory` basierenden Formel abgeleitet wird. Wie in der vorhergehenden Tabelle dargestellt, erzeugt die Formel Verbindungsbegrenzungen, die bei einer Verdoppelung des Speichers über die progressiv größeren R3-, R4- und R5-Instances um jeweils 1000 und bei unterschiedlichen Speichergrößen von T2- und T3-Instances um 45 erhöht werden.

Weitere Informationen darüber, wie `DBInstanceClassMemory` berechnet wird, finden Sie unter Festlegen von DB-Parametern.

Aurora MySQL- und RDS für MySQL-DB-Instances weisen unterschiedliche Speicher-Overhead-Mengen auf. Daher kann der `max_connections`-Wert für Aurora MySQL- und RDS für MySQL-DB-Instances, die dieselbe Instance-Klasse verwenden, unterschiedlich sein. Die Werte in der Tabelle gelten nur für Aurora MySQL-DB-Instances.

> **Note**
>
> Die deutlich niedrigeren Verbindungsgrenzwerte für T2- und T3-Instances basieren darauf, dass diese Instances-Klassen bei Aurora nur für Entwicklungs- und Testszenarien und nicht für Produktionsworkloads vorgesehen sind.

Für Systeme, die Standardwerte für andere größere Arbeitsspeicherverbraucher verwenden, beispielsweise Pufferpool und Abfragezwischenspeicher, werden die Standard-Verbindungsgrenzwerte angepasst. Wenn Sie diese anderen Einstellungen für Ihr Cluster ändern, sollten Sie auch den Verbindungsgrenzwert ändern, um die Zu- oder Abnahme des verfügbaren Arbeitsspeichers in den DB-Instances zu berücksichtigen.

Temporäre Speicherlimits für Aurora MySQL

Aurora MySQL speichert Tabellen und Indizes im Aurora Speichersubsystem. Aurora MySQL verwendet separaten temporären oder lokalen Speicher für nicht persistente temporäre Dateien und temporäre Nicht-InnoDB-Tabellen. Zum lokalen Speicher gehören auch Dateien, die

für Zwecke wie das Sortieren großer Datensätze während der Abfrageverarbeitung oder für Indexerstellungsvorgänge verwendet werden. Er umfasst keine temporären InnoDB-Tabellen.

Weitere Informationen zu temporären Tabellen in Aurora MySQL Version 3 finden Sie unter Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3. Weitere Informationen zu temporären Tabellen in Version 2 finden Sie unter Temporäres Tabellenverhalten in Aurora-MySQL-Version 2.

Die Daten und temporären Dateien auf diesen Volumes gehen beim Starten und Stoppen der DB-Instance sowie beim Austausch des Hosts verloren.

Diese lokalen Speicher-Volumes werden von Amazon Elastic Block Store (EBS) gestützt und können durch Einsatz einer größeren DB-Instance-Klasse erweitert werden. Weitere Informationen über Speicher finden Sie unter Amazon-Aurora-Speicher.

Lokaler Speicher wird auch für den Import von Daten aus Amazon S3 mit `LOAD DATA FROM S3` oder `LOAD XML FROM S3` und für den Export von Daten zu S3 mit SELECT INTO OUTFILE S3 verwendet. Weitere Informationen zum Import aus und Export zu S3 finden Sie nachfolgend:

- Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket
- Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket

Aurora MySQL verwendet separaten permanenten Speicher für Fehlerprotokolle, allgemeine Protokolle, langsame Abfrageprotokolle und Prüfprotokolle für die meisten DB-Instance-Klassen von Aurora MySQL (ausgenommen Instance-Klassentypen mit Spitzenlastleistung wie db.t2, db.t3 und db.t4g). Die Daten auf diesem Volume werden beim Starten und Stoppen der DB-Instance sowie beim Austausch des Hosts beibehalten.

Dieses permanente Speicher-Volume wird ebenfalls von Amazon EBS unterstützt und hat eine feste Größe entsprechend der DB-Instance-Klasse. Eine Erweiterung durch die Verwendung einer größeren DB-Instance-Klasse ist nicht möglich.

Die folgende Tabelle zeigt die maximale Menge an temporärem und permanentem Speicher, die für jede DB-Instance-Klasse von Aurora MySQL verfügbar ist. Informationen zur Unterstützung der DB-Instance-Klasse für Aurora finden Sie unter DB-Instance-Klassen von Amazon Aurora.

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer/lokaler Speicher (GiB)	Zusätzlicher maximal verfügbarer Speicher für Protokolldateien (GiB)
db.x2g.16xlarge	1280	500
db.x2g.12xlarge	960	500
db.x2g.8xlarge	640	500
db.x2g.4xlarge	320	500
db.x2g.2xlarge	160	60
db.x2g.xlarge	80	60
db.x2g.large	40	60
db.r8g.48xlarge	3840	500
db.r8g.24xlarge	1920	500
db.r8g.16xlarge	1280	500
db.r8g.12xlarge	960	500
db.r8g.8xlarge	640	500
db.r8g.4xlarge	320	500
db.r8g.2xlarge	160	60
db.r8g.xlarge	80	60
db.r8g.large	32	60
db.r7i.48xlarge	3840	500
db.r7i.24xlarge	1920	500
db.r7i.16xlarge	1280	500

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer/lokaler Speicher (GiB)	Zusätzlicher maximal verfügbarer Speicher für Protokolldateien (GiB)
db.r7i.12xlarge	960	500
db.r7i.8xlarge	640	500
db.r7i.4xlarge	320	500
db.r7i.2xlarge	160	60
db.r7i.xlarge	80	60
db.r7i.large	32	60
db.r7g.16xlarge	1280	500
db.r7g.12xlarge	960	500
db.r7g.8xlarge	640	500
db.r7g.4xlarge	320	500
db.r7g.2xlarge	160	60
db.r7g.xlarge	80	60
db.r7g.large	32	60
db.r6i.32xlarge	2560	500
db.r6i.24xlarge	1920	500
db.r6i.16xlarge	1280	500
db.r6i.12xlarge	960	500
db.r6i.8xlarge	640	500
db.r6i.4xlarge	320	500

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer/lokaler Speicher (GiB)	Zusätzlicher maximal verfügbarer Speicher für Protokolldateien (GiB)
db.r6i.2xlarge	160	60
db.r6i.xlarge	80	60
db.r6i.large	32	60
db.r6g.16xlarge	1280	500
db.r6g.12xlarge	960	500
db.r6g.8xlarge	640	500
db.r6g.4xlarge	320	500
db.r6g.2xlarge	160	60
db.r6g.xlarge	80	60
db.r6g.large	32	60
db.r5.24xlarge	1920	500
db.r5.16xlarge	1280	500
db.r5.12xlarge	960	500
db.r5.8xlarge	640	500
db.r5.4xlarge	320	500
db.r5.2xlarge	160	60
db.r5.xlarge	80	60
db.r5.large	32	60
db.r4.16xlarge	1280	500

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer/lokaler Speicher (GiB)	Zusätzlicher maximal verfügbarer Speicher für Protokolldateien (GiB)
db.r4.8xlarge	640	500
db.r4.4xlarge	320	500
db.r4.2xlarge	160	60
db.r4.xlarge	80	60
db.r4.large	32	60
db.t4g.large	32	–
db.t4g.medium	32	–
db.t3.large	32	–
db.t3.medium	32	–
db.t3.small	32	–
db.t2.medium	32	–
db.t2.small	32	–

Important

Diese Werte stellen die theoretische maximale Menge an freiem Speicher auf jeder DB-Instance dar. Der tatsächliche lokale Speicher, der Ihnen zur Verfügung steht, ist möglicherweise niedriger. Aurora verwendet einen lokalen Speicher für seine Verwaltungsprozesse und die DB-Instance verwendet einen lokalen Speicher, noch bevor Sie Daten laden. Sie können den für eine bestimmte DB-Instance verfügbaren temporären Speicher mit der `FreeLocalStorage`CloudWatch-Metrik überwachen, die unter CloudWatch Amazon-Metriken für Amazon Aurora beschrieben wird. Sie können die Menge des freien Speichers zur Zeit überprüfen. Sie können auch die Menge des freien Speichers im Laufe der Zeit abzeichnen. Die Überwachung des freien Speichers im Laufe der Zeit

hilft Ihnen festzustellen, ob der Wert steigt oder sinkt, oder um die Mindest-, Maximal- oder Durchschnittswerte zu ermitteln.
(Dies gilt nicht für Aurora Serverless v2.)

Rückverfolgen eines Aurora-DB-Clusters

Mit Amazon Aurora MySQL-kompatible Edition können Sie einen DB-Cluster auf einen bestimmten Zeitpunkt zurückverfolgen, ohne die Daten aus einer Sicherung wiederherstellen zu müssen.

Inhalt

- Übersicht zur Rückverfolgung
  - Rückverfolgungsfenster
  - Rückverfolgungszeit
  - Einschränkungen der Rückverfolgung
- Verfügbarkeit von Regionen und Versionen
- Überlegungen zum Upgrade für rückverfolgungsfähige Cluster
- Konfigurieren der Rückverfolgung eines DB-Clusters von Aurora MySQL
- Durchführen einer Rückverfolgung für einen DB-Cluster von Aurora MySQL
- Überwachen der Rückverfolgung für einen DB-Cluster von Aurora MySQL
- Abonnieren eines Rückverfolgungsereignisses mit der Konsole
- Abrufen vorhandener Rückverfolgungen
- Deaktivieren der Rückverfolgung für einen DB-Cluster von Aurora MySQL

Übersicht zur Rückverfolgung

Durch die Rückverfolgung wird der DB-Cluster auf den angegebenen Zeitpunkt "zurückgespult". Die Rückverfolgung ersetzt nicht das Sichern des DB-Clusters, damit dieser mit dem Stand eines bestimmten Zeitpunkts wiederhergestellt werden kann. Gegenüber herkömmlichen Sicherungen und Wiederherstellungen bietet die Rückverfolgung jedoch einige Vorteile:

- Sie können Fehler einfach rückgängig machen. Wenn Sie versehentlich eine destruktive Aktion ausführen – beispielsweise eine DELETE-Anweisung ohne WHERE-Klausel –, können Sie den DB-Cluster bei minimaler Unterbrechung der Service-Bereitstellung auf den Zeitpunkt vor der destruktiven Aktion rückverfolgen.

- Sie können einen DB-Cluster schnell rückverfolgen. Das Wiederherstellen eines DB-Clusters auf den Status eines bestimmten Zeitpunkts startet einen neuen DB-Cluster und stellt diesen aus den Sicherungsdaten oder einem DB-Cluster-Snapshot wieder her. Dies kann einige Stunden dauern. Die Rückverfolgung eines DB-Clusters macht keinen neuen DB-Cluster erforderlich. Zudem erfolgt das "Zurückspulen" innerhalb weniger Minuten.
- Sie können frühere Datenänderungen untersuchen. Sie können einen DB-Cluster mehrfach in der Zeit zurück- und vorspulen, um zu ermitteln, wann eine bestimmte Datenänderung vorgenommen wurde. Sie können beispielsweise einen DB-Cluster drei Stunden zurück und anschließend eine Stunde vorspulen. In diesem Fall liegt der Rückverfolgungszeitpunkt zwei Stunden vor der Originalzeit.

> **Note**
>
> Weitere Informationen zum Wiederherstellen eines DB-Cluster für einen bestimmten Zeitpunkt finden Sie unter Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters.

Rückverfolgungsfenster

Für die Rückverfolgung gilt ein Zielfenster für die Rückverfolgung und ein Ist-Rückverfolgungsfenster:

- Das Zielfenster für die Rückverfolgung gibt den Zeitraum an, über den die Rückverfolgung des DB-Clusters möglich sein soll. Wenn Sie die Rückverfolgung aktivieren, geben Sie ein Zielfenster für die Rückverfolgung an. Sie können beispielsweise ein Zielfenster von 24 Stunden für die Rückverfolgung angeben, wenn die Rückverfolgung des DB-Clusters über einen Tag möglich sein soll.
- Das tatsächliche Rückverfolgungsfenster bezeichnet den tatsächlichen Zeitraum, über den die Rückverfolgung des DB-Clusters möglich ist. Diese Dauer kann kürzer als die des Zielfensters für die Rückverfolgung sein. Das tatsächliche Rückverfolgungsfenster basiert auf dem Workload und dem für Informationen zu Datenbankänderungen (die sogenannten Änderungsdatensätze) verfügbaren Arbeitsspeicher.

Wenn Sie ein Aurora-DB-Cluster mit aktivierter Rückverfolgung aktualisieren, generieren Sie Änderungsdatensätze. Aurora bewahrt Änderungsdatensätze für das Zielfenster für die Rückverfolgung auf und Sie zahlen für die Speicherung einen Stundensatz. Die Anzahl gespeicherter

Änderungsdatensätze hängt vom Zielfenster für die Rückverfolgung und von der Workload auf dem DB-Cluster ab. Die Workload bezeichnet die Anzahl der Änderungen, die Sie in einem gegebenen Zeitraum am DB-Cluster vornehmen. Bei starker Workload werden mehr Änderungsdatensätze im Zielfenster für die Rückverfolgung gespeichert als bei geringerer Workload.

Das Zielfenster für die Rückverfolgung bezeichnet also die maximale Dauer, über die eine Rückverfolgung des DB-Clusters möglich sein soll. In den meisten Fällen ist eine Rückverfolgung über die von Ihnen angegebene Maximaldauer möglich. In einigen Fällen kann der DB-Cluster aber nicht genug Änderungsdatensätze speichern, um eine Rückverfolgung über diese Maximaldauer zu ermöglichen, das Ist-Rückverfolgungsfenster ist in einer solchen Situation kleiner als das Zielfenster. Normalerweise ist das Ist-Rückverfolgungsfenster kleiner als das Zielfenster, wenn die Workload auf dem DB-Cluster extrem hoch ist. Wenn das Ist-Rückverfolgungsfenster kleiner als das Zielfenster ist, senden wir eine Benachrichtigung.

Wenn die Rückverfolgung für einen DB-Cluster aktiviert ist und Sie eine im DB-Cluster gespeicherte Tabelle löschen, bewahrt Aurora die betreffende Tabelle in den Änderungsdatensätzen für die Rückverfolgung auf. Dies ermöglicht Ihnen die Rückkehr zu einem Zeitpunkt vor dem Löschen der Tabelle. Wenn das Zielfenster für die Rückverfolgung nicht genug Platz zum Speichern der Tabelle bietet, wird die Tabelle schließlich aus den Änderungsdatensätzen für die Rückverfolgung entfernt.

Rückverfolgungszeit

Die Rückverfolgung durch Aurora erfolgt immer auf einen Zeitpunkt, der für den DB-Cluster konsistent ist. Dadurch wird das Auftreten noch nicht eingetragener Transaktionen nach Abschluss der Rückverfolgung verhindert. Wenn Sie eine Zeit für eine Rückverfolgung angeben, wählt Aurora automatisch den nächstgelegenen konsistenten Zeitpunkt. Dieser Ansatz bedeutet, dass der abgeschlossene Backtrack möglicherweise nicht genau der von Ihnen angegebenen Zeit entspricht, aber Sie können die genaue Zeit für einen Backtrack mithilfe des describe-db-cluster-backtracks AWS CLI-Befehls ermitteln. Weitere Informationen finden Sie unter Abrufen vorhandener Rückverfolgungen.

Einschränkungen der Rückverfolgung

Die folgenden Einschränkungen gelten für die Rückverfolgung:

- Die Rückverfolgung ist nur für DB-Cluster verfügbar, die mit aktivierter Rückverfolgungsfunktion erstellt wurden. Sie können einen DB-Cluster nicht modifizieren, um die Rückverfolgungsfunktion zu aktivieren. Sie können die Rückverfolgungsfunktion aktivieren, wenn Sie einen neuen DB-Cluster erstellen oder einen Snapshot eines DB-Clusters wiederherstellen.

- Die Obergrenze für das Zielfenster für die Rückverfolgung beträgt 72 Stunden.
- Die Rückverfolgung betrifft den gesamten DB-Cluster. Sie können also beispielsweise nicht eine einzelne Tabelle oder eine einzelne Datenaktualisierung verfolgen.
- Sie können keine regionsübergreifenden Read Replicas aus einem Cluster mit Rückverfolgungsmöglichkeit erstellen, aber Sie können trotzdem die Replikation von Binärprotokollen (Binlog) auf dem Cluster aktivieren. Wenn Sie versuchen, einen DB-Cluster mit aktivierter Binärprotokollierung rückzuverfolgen, tritt üblicherweise ein Fehler auf, sofern die Rückverfolgung nicht erzwungen wird. Jeder Versuch, einen Backtrack zu erzwingen, macht nachgeschaltete Read Replicas kaputt und beeinträchtigt andere Vorgänge, wie z. B. Bereitstellungen. blue/green
- Sie können einen Datenbankklon nicht auf einen Zeitpunkt zurückverfolgen, der vor der Erstellung des Datenbankklons liegt. Sie können aber die ursprüngliche Datenbank auf einen Zeitpunkt vor der Erstellung des Klons zurückverfolgen. Weitere Informationen zum Klonen von Datenbanken erhalten Sie unter Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster.
- Bei der Rückverfolgung kommt es zu einer kurzen Betriebsunterbrechung der DB-Instance. Sie müssen Anwendungen vor dem Starten einer Rückverfolgung stoppen oder anhalten, um sicherzustellen, dass keine neuen Lese- oder Schreibanforderungen ergehen. Während der Rückverfolgung hält Aurora die Datenbank an, schließt alle offenen Verbindungen und verwirft noch nicht durchgeführte Lese- und Schreibvorgänge. Dann wird der Abschluss der Rückverfolgung abgewartet.
- Sie können keinen regionsübergreifenden Snapshot eines Clusters mit Backtrack-Aktivierung in einer AWS Region wiederherstellen, die Backtracking nicht unterstützt.
- Wenn Sie ein direktes Upgrade für einen Cluster mit Rückverfolgungsmöglichkeit von Aurora MySQL Version 2 auf Version 3 durchführen, können Sie nicht auf einen Zeitpunkt vor der Durchführung des Upgrades zurückgehen.

Verfügbarkeit von Regionen und Versionen

Die Rückverfolgung ist für Aurora PostgreSQL nicht verfügbar.

Im Folgenden sind die unterstützten Engines und die Regionsverfügbarkeit für die Rückverfolgung mit Aurora MySQL aufgeführt.

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
USA Ost (Nord-Virginia)	Alle Versionen	Alle Versionen
USA Ost (Ohio)	Alle Versionen	Alle Versionen
USA West (Nordkalifornien)	Alle Versionen	Alle Versionen
USA West (Oregon)	Alle Versionen	Alle Versionen
Afrika (Kapstadt)	–	–
Asia Pacific (Hongkong)	–	–
Asien-Pazifik (Jakarta)	–	–
Asien-Pazifik (Malaysia)	–	–
Asien-Pazifik (Melbourne)	–	–
Asien-Pazifik (Mumbai)	Alle Versionen	Alle Versionen
Asien-Pazifik (Neuseeland)	–	–
Asien-Pazifik (Osaka)	Alle Versionen	Version 2.07.3 und höher
Asien-Pazifik (Seoul)	Alle Versionen	Alle Versionen

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Asien-Pazifik (Singapur)	Alle Versionen	Alle Versionen
Asien-Pazifik (Sydney)	Alle Versionen	Alle Versionen
Asien-Pazifik (Taipeh)	–	–
Asien-Pazifik (Thailand)	–	–
Asien-Pazifik (Tokio)	Alle Versionen	Alle Versionen
Kanada (Zentral)	Alle Versionen	Alle Versionen
Kanada West (Calgary)	–	–
China (Peking)	–	–
China (Ningxia)	–	–
Europa (Frankfurt)	Alle Versionen	Alle Versionen
Europa (Irland)	Alle Versionen	Alle Versionen
Europa (London)	Alle Versionen	Alle Versionen
Europa (Milan)	–	–
Europa (Paris)	Alle Versionen	Alle Versionen
Europa (Spain)	–	–
Europa (Stockholm)	–	–

Region	Aurora-MySQL-Version 3	Aurora-MySQL-Version 2
Europa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Mexiko (Zentral)	–	–
Middle East (Bahrain)	–	–
Naher Osten (VAE)	–	–
Südamerika (São Paulo)	–	–
AWS GovCloud (US-Ost)	–	–
AWS GovCloud (US-West)	–	–

Überlegungen zum Upgrade für rückverfolgungsfähige Cluster

Sie können ein Upgrade für einen DB-Cluster mit Rückverfolgungsmöglichkeit von Aurora MySQL Version 2 auf Version 3 durchführen, da alle Nebenversionen von Aurora MySQL Version 3 für die Rückverfolgung unterstützt werden.

Konfigurieren der Rückverfolgung eines DB-Clusters von Aurora MySQL

Um die Rückverfolgungsfunktion zu verwenden, müssen Sie die Rückverfolgung aktivieren und ein Zielfenster für die Rückverfolgung angeben. Andernfalls wird die Rückverfolgung deaktiviert.

Geben Sie für das Zielfenster für die Rückverfolgung an, für wie lange Sie Ihre Datenbank mit dieser Funktion zurückverfolgen können möchten. Aurora versucht, genügend Änderungsaufzeichnungen aufzubewahren, um dieses Zeitfenster zu unterstützen.

Konsole

Sie können die Konsole verwenden, um die Rückverfolgung beim Erstellen eines DB-Clusters zu konfigurieren. Sie können auch einen DB-Cluster ändern, um das Rückverfolgungsfenster für einen rückverfolgungsfähigen Cluster zu ändern. Wenn Sie die Rückverfolgung für einen Cluster vollständig deaktivieren, indem Sie das Rückverfolgungsfenster auf 0 setzen, können Sie die Rückverfolgung für diesen Cluster nicht erneut aktivieren.

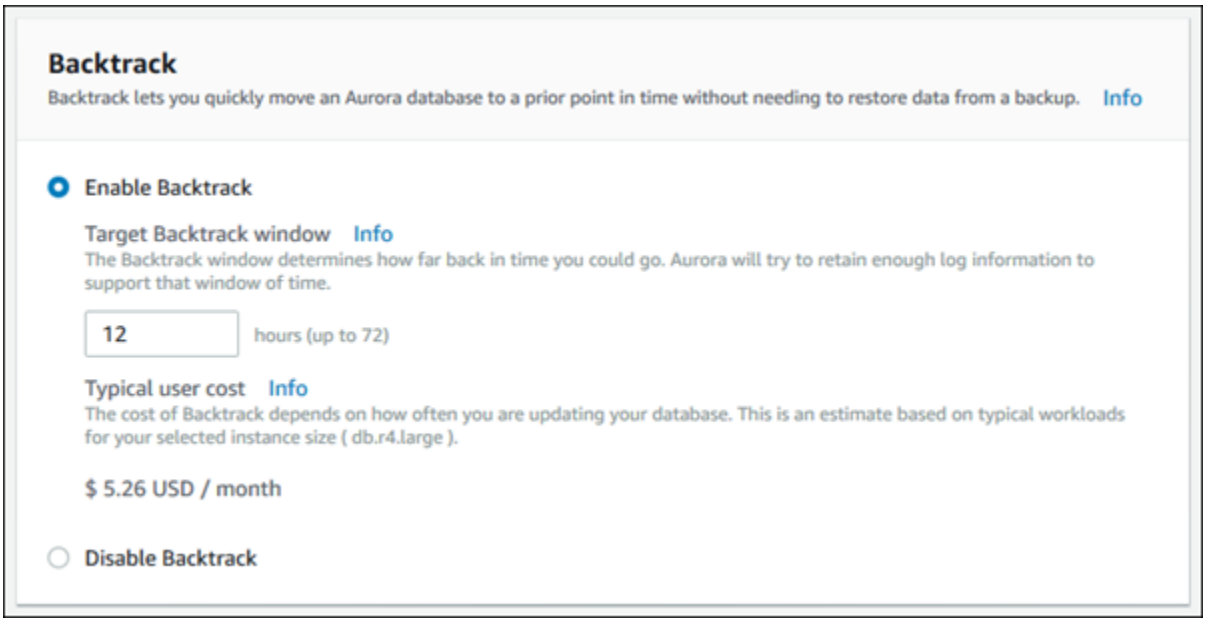
Themen

- Konfigurieren der Rückverfolgung mit der Konsole beim Erstellen eines DB-Clusters
- Konfigurieren der Rückverfolgung mit der Konsole beim Ändern eines DB-Clusters

Konfigurieren der Rückverfolgung mit der Konsole beim Erstellen eines DB-Clusters

Wenn Sie ein neues Aurora MySQL-DB-Cluster erstellen, wird die Rückverfolgung konfiguriert, indem Sie Enable Backtrack (Rückverfolgung aktivieren) wählen und für Target Backtrack window (Zielfenster für die Rückverfolgung) im Bereich Backtrack (Rückverfolgung) einen Wert größer als 0 eingeben.

Befolgen Sie die Anweisungen unter , um einen DB-Cluster zu erstelle Erstellen eines Amazon Aurora-DB Clusters. Die folgende Abbildung zeigt den Bereich Backtrack (Rückverfolgung).



Wenn Sie einen DB-Cluster erstellen, verfügt Aurora über keinerlei Daten zur Workload des DB-Clusters. Deshalb können die Kosten für den neuen DB-Cluster nicht geschätzt werden. Stattdessen

stellt die Konsole einen typischen Wert für die Benutzerkosten für das angegebene Zielfenster für die Rückverfolgung basierend auf einer typischen Workload dar. Diese typischen Kosten dienen also nur als Hinweis auf die möglichen Kosten, die durch die Verwendung der Rückverfolgungsfunktion entstehen können.

> **Important**
>
> Ihre tatsächlichen Kosten können niedriger sein, weil sie von der Workload des DB-Clusters abhängig sind.

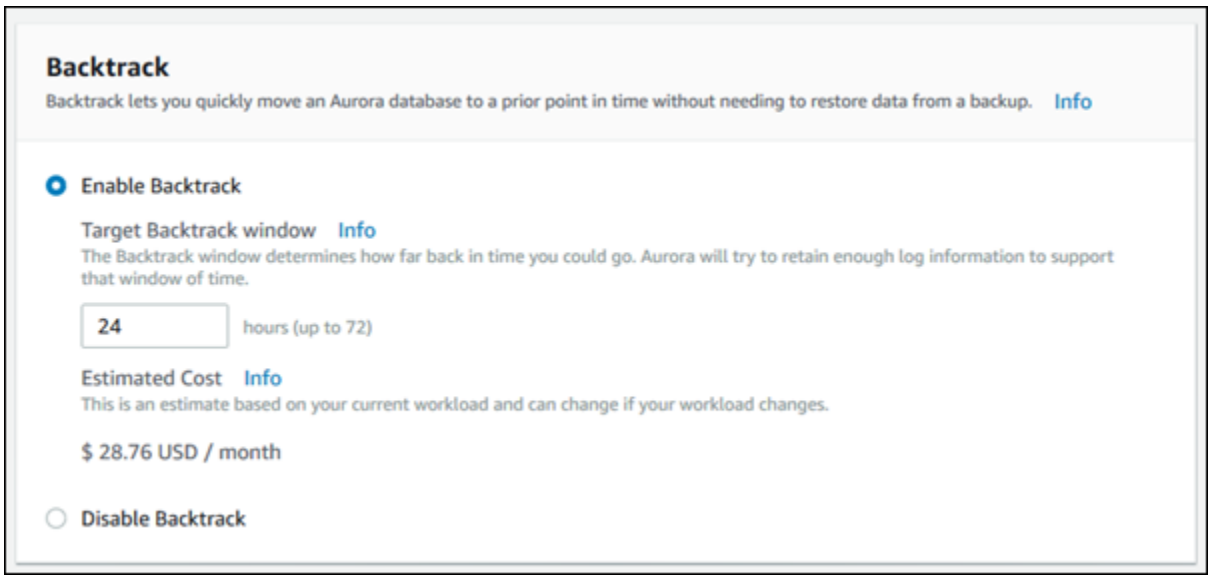
Konfigurieren der Rückverfolgung mit der Konsole beim Ändern eines DB-Clusters

Sie können die Rückverfolgung für einen DB-Cluster mit der Konsole ändern.

> **Note**
>
> Derzeit können Sie die Rückverfolgungsfunktion nur für einen DB-Cluster ändern, für den die Funktion „Backtrack“ aktiviert ist. Der Abschnitt Backtrack wird nicht für einen DB-Cluster angezeigt, der mit deaktivierter Backtrack-Funktion erstellt wurde oder wenn die Backtrack-Funktion für den DB-Cluster deaktiviert wurde.

Ändern Sie die Rückverfolgung für einen DB-Cluster mit der Konsole wie folgt:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie den zu ändernden Cluster und dann die Option Modify (Ändern) aus.
4. Ändern Sie für das Target Backtrack window (Zielfenster für die Rückverfolgung) den Zeitraum, über den die Rückverfolgung möglich sein soll. Die Obergrenze ist 72 Stunden.



Die Konsole zeigt die geschätzten Kosten für die von Ihnen angegebene Größe des Zeitfensters basierend auf der Workload des DB-Clusters in der Vergangenheit an:

- Wenn die Rückverfolgung für das DB-Cluster deaktiviert wurde, basiert die Kostenschätzung auf der Metrik `VolumeWriteIOPS` für das DB-Cluster in Amazon CloudWatch.
- Wenn die Rückverfolgung zuvor für das DB-Cluster aktiviert war, basiert die Kostenschätzung auf der Metrik `BacktrackChangeRecordsCreationRate` für das DB-Cluster in Amazon CloudWatch.

5. Klicken Sie auf Weiter.
6. Wählen Sie für Einplanung von Änderungen eine der folgenden Optionen:
   - Apply during the next scheduled maintenance window (Im nächsten geplanten Wartungsfenster anwenden): Die Änderung für Target Backtrack window (Zielfenster für die Rückverfolgung) soll erst im nächsten Wartungsfenster angewendet werden.
   - Apply immediately (Sofort anwenden): Die Änderung für Target Backtrack window (Zielfenster für die Rückverfolgung) soll so bald wie möglich angewendet werden.
7. Wählen Sie Modify Cluster (Cluster ändern) aus.

AWS CLI

Wenn Sie ein neues Aurora MySQL-DB-Cluster über den AWS-CLI-Befehl create-db-cluster erstellen, wird die Rückverfolgung konfiguriert, wenn Sie für `--backtrack-window` einen Wert größer als 0

angeben. Der Wert `--backtrack-window` gibt das Zielfenster für die Rückverfolgung an. Weitere Informationen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Sie können den Wert für `--backtrack-window` auch mit den folgenden AWS-CLI-Befehlen angeben:

- modify-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Das folgende Verfahren beschreibt, wie Sie das Zielfenster für die Rückverfolgung für einen DB-Cluster mit der änder AWS CLI.

Ändern Sie das Zeitfenster für die Rückverfolgung für einen DB-Cluster mit der wie folgt AWS CLI

- Rufen Sie den AWS-CLI-Befehl modify-db-cluster auf und geben Sie folgende Werte an:
  - `--db-cluster-identifier`: Name des DB-Clusters.
  - `--backtrack-window`: maximaler Zeitraum in Sekunden für die Rückverfolgung des DB-Clusters.

  Im folgenden Beispiel wird für das Zielfenster für die Rückverfolgung von `sample-cluster` ein Wert von einem Tag (86 400 Sekunden) festgelegt.

  Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 86400
```

  Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 86400
```

> **Note**
>
> Derzeit kann die Rückverfolgung nur für einen DB-Cluster aktiviert werden, der mit aktivierter Rückverfolgungsfunktion erstellt wurde.

RDS-API

Wenn Sie einen neuen Aurora MySQL-DB-Cluster über die Amazon-RDS-API-Operation CreateDBCluster erstellen, wird die Rückverfolgung konfiguriert, wenn Sie für `BacktrackWindow` einen Wert größer als 0 angeben. Der Wert `BacktrackWindow` gibt das Zielfenster für die Rückverfolgung für den mit dem Wert `DBClusterIdentifier` angegebenen DB-Cluster an. Weitere Informationen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Sie können den Wert für `BacktrackWindow` auch mittels der folgenden API-Operationen angeben:

- ModifyDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

> **Note**
>
> Derzeit kann die Rückverfolgung nur für einen DB-Cluster aktiviert werden, der mit aktivierter Rückverfolgungsfunktion erstellt wurde.

Durchführen einer Rückverfolgung für einen DB-Cluster von Aurora MySQL

Sie können einen DB-Cluster auf einen bestimmten Zeitpunkt im Zielfenster für die Rückverfolgung rückverfolgen. Wenn der Zeitstempel für die Rückverfolgung nicht vor dem frühestmöglichen Rückverfolgungszeitpunkt und auch nicht in der Zukunft liegt, wird der DB-Cluster auf den betreffenden Zeitpunkt rückverfolgt.

Andernfalls tritt üblicherweise ein Fehler auf. Wenn Sie versuchen, einen DB-Cluster mit aktivierter Binärprotokollierung rückzuverfolgen, tritt üblicherweise ein Fehler auf, sofern Sie nicht das Erzwingen der Rückverfolgung veranlasst haben. Das Erzwingen einer Rückverfolgung kann zu Problemen mit anderen Operationen führen, die Binärprotokollierung verwenden.

> **Important**
>
> Die Rückverfolgung erzeugt keine binlog-Einträge für die von der Funktion vorgenommenen Änderungen. Wenn Sie die Binärprotokollierung für den DB-Cluster aktiviert haben, ist die Rückverfolgung möglicherweise nicht mit der binlog-Implementierung kompatibel.

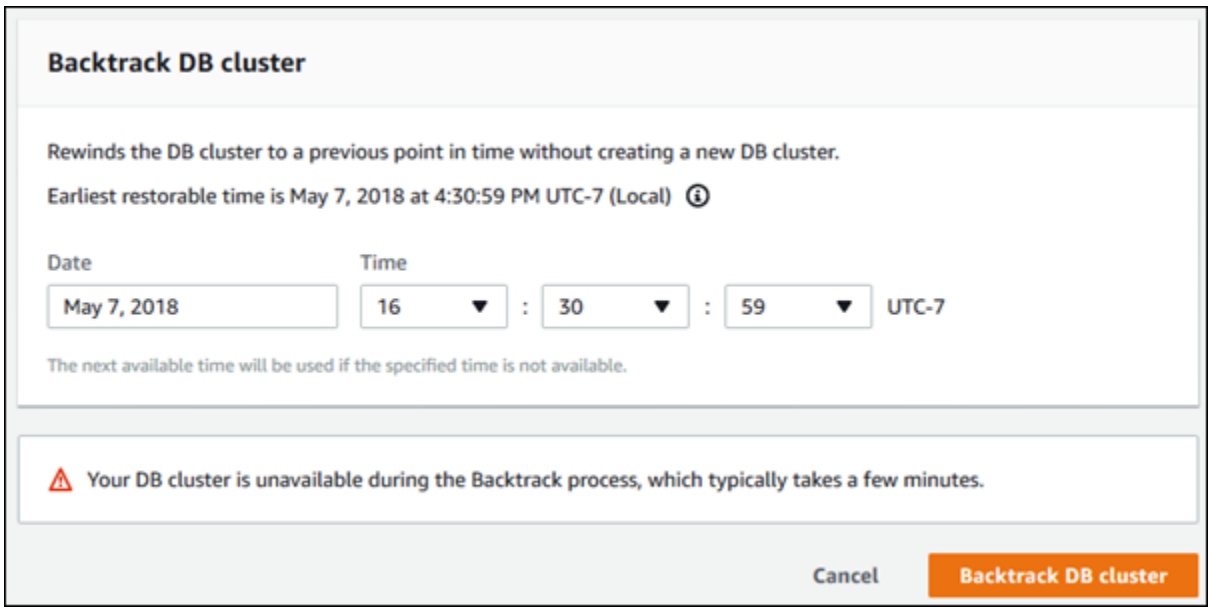
> **Note**
>
> Bei Datenbankklonen können Sie den DB-Cluster auf einen Zeitpunkt rückverfolgen, der vor dem Zeitpunkt der Erstellung des Klons liegt. Weitere Informationen zum Klonen von Datenbanken erhalten Sie unter Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster.

Konsole

Das folgende Verfahren beschreibt, wie Sie eine Rückverfolgung für einen DB-Cluster mit der Konsole durchführen.

Führen Sie eine Rückverfolgung mit der Konsole wie folgt durch:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Instances aus.
3. Wählen Sie die primäre Instance für den DB-Cluster aus, den Sie rückverfolgen möchten.
4. Wählen Sie unter Actions (Aktionen) die Option Backtrack DB cluster (DB-Cluster rückverfolgen) aus.
5. Geben Sie auf der Seite Backtrack DB cluster (DB-Cluster rückverfolgen) den Zeitstempel ein, zu dem der DB-Cluster rückverfolgt werden soll.



6. Wählen Sie Backtrack DB cluster (DB-Cluster rückverfolgen) aus.

AWS CLI

Im folgenden Verfahren wird beschrieben, wie ein DB-Cluster mit der rückverfolgt wir AWS CLI.

Rückverfolgen Sie einen DB-Cluster mit der wie folgt AWS CLI

- Rufen Sie den AWS-CLI-Befehl backtrack-db-cluster auf und stellen Sie die folgenden Werte bereit:
  - `--db-cluster-identifier`: Name des DB-Clusters.
  - `--backtrack-to` – Der Zeitstempel im ISO 8601-Format, zu dem das DB-Cluster zurückverfolgt werden soll.

  Das folgende Beispiel führt eine Rückverfolgung des DB-Clusters `sample-cluster` auf den 19. März 2018 um 10:00 Uhr durch.

  Für Linux, macOS oder Unix:

```
aws rds backtrack-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-to 2018-03-19T10:00:00+00:00
```

Für Windows:

```
aws rds backtrack-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-to 2018-03-19T10:00:00+00:00
```

RDS-API

Um ein DB-Cluster über die Amazon-RDS-API zurückzuverfolgen, verwenden Sie die Operation BacktrackDBCluster. Diese Operation nimmt die Rückverfolgung des mit `DBClusterIdentifier` angegebenen DB-Clusters auf den angegebenen Zeitpunkt vor.

Überwachen der Rückverfolgung für einen DB-Cluster von Aurora MySQL

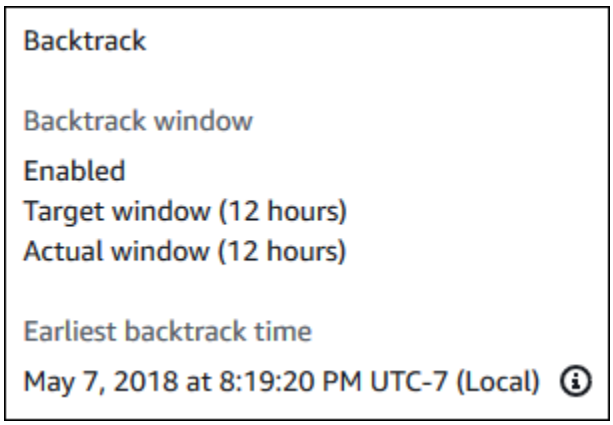
Sie können Rückverfolgungsinformationen für einen DB-Cluster anzeigen sowie Rückverfolgungskennzahlen überwachen.

Konsole

Zeigen Sie mit der Konsole Rückverfolgungsinformationen für einen DB-Cluster an und überwachen Sie die Rückverfolgungskennzahlen wie folgt:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie den Namen des DB-Clusters, um die zugehörigen Informationen anzuzeigen.

   Die Rückverfolgungsinformationen befinden sich im Bereich Backtrack (Rückverfolgung).



Wenn die Rückverfolgung aktiviert ist, sind folgende Informationen verfügbar:

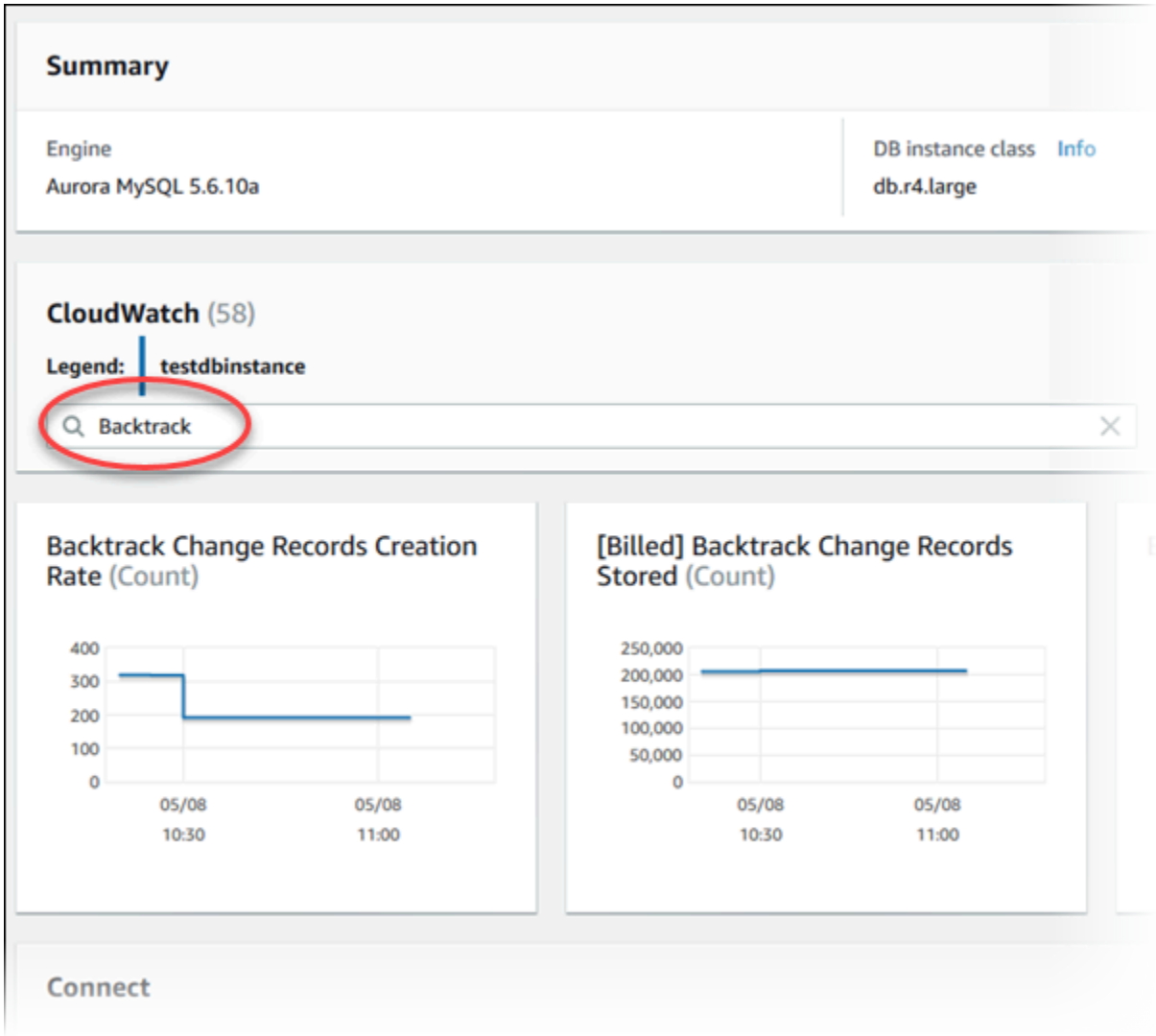
- Target window (Zielfenster): der zurzeit für das Zielfenster für die Rückverfolgung angegebene Zeitraum. Das Zielfenster bezeichnet die maximale Dauer, für die eine Rückverfolgung durchgeführt werden kann, sofern genug Speicher verfügbar ist.
- Actual window (Tatsächliches Fenster): der tatsächliche Zeitraum, über den Sie die Rückverfolgung ausführen können; dieser kann kleiner als das Zielfenster für die Rückverfolgung sein. Das Ist-Rückverfolgungsfenster basiert auf der Workload und dem für die Aufnahme von Änderungsdatensätzen für die Rückverfolgung verfügbaren Speicher.
- Earliest backtrack time (Früheste Rückverfolgungszeit): der früheste Rückverfolgungszeitpunkt, der für das DB-Cluster möglich ist. Sie können einen DB-Cluster nicht auf eine vor dem angezeigten Zeitpunkt liegende Zeit rückverfolgen.

4. Zeigen Sie die Rückverfolgungskennzahlen für den DB-Cluster wie folgt an:

   a. Wählen Sie im Navigationsbereich Instances aus.

   b. Wählen Sie den Namen der primären Instance des DB-Clusters, um die Details anzuzeigen.

   c. Geben Sie im Bereich CloudWatch (CloudWatch) den Wert **`Backtrack`** in das Feld CloudWatch (CloudWatch) ein, um nur die Rückverfolgungsmetriken anzuzeigen.



Die folgenden Kennzahlen werden angezeigt:

- Backtrack Change Records Creation Rate (Count) (Erstellungsrate (Anzahl) der Änderungsdatensätze für die Rückverfolgung): Diese Metrik gibt die Anzahl der Änderungsdatensätze für die Rückverfolgung an, die im Verlauf von fünf Minuten für das DB-Cluster erstellt wurden. Sie können diese Kennzahl heranziehen, um die Rückverfolgungskosten für das Zielfenster für die Rückverfolgung zu schätzen.
- [Billed] Backtrack Change Records Stored (Count) ([Berechnete] für die Rückverfolgung gespeicherte Änderungsdatensätze (Anzahl)): Diese Metrik gibt die tatsächliche Anzahl der Änderungsdatensätze für die Rückverfolgung an, die vom DB-Cluster verwendet werden.
- Backtrack Window Actual (Minutes) (Tatsächliches Rückverfolgungsfenster (Minuten)): Diese Metrik gibt an, ob es eine Differenz zwischen dem Zielfenster für die

Rückverfolgung und dem tatsächlichen Rückverfolgungsfenster gibt. Wenn das Zielfenster für die Rückverfolgung beispielsweise 2 Stunden (120 Minuten) lang ist und diese Kennzahl angibt, dass das Ist-Rückverfolgungsfenster nur 100 Minuten lang ist, ist das Ist-Rückverfolgungsfenster kleiner als das Zielfenster.

- Backtrack Window Alert (Count) (Rückverfolgungsfenster-Alarm (Anzahl)): Diese Metrik gibt an, wie oft das tatsächliche Rückverfolgungsfenster über einen bestimmten Zeitraum kleiner als das Zielfenster für die Rückverfolgung ist.

> **Note**
>
> Die folgenden Kennzahlen können hinter der aktuellen Zeit zurückliegen:
>
> - Erstellungsrate (Anzahl) der Änderungsdatensätze für die Rückverfolgung
> - [Billed] Backtrack Change Records Stored (Count) ([Berechnete] für die Rückverfolgung gespeicherte Änderungsdatensätze (Anzahl))

AWS CLI

Das folgende Verfahren beschreibt, wie Sie Rückverfolgungsinformationen für einen DB-Cluster mit anzeige AWS CLI.

Zeigen Sie Rückverfolgungsinformationen für einen DB-Cluster mit der wie folgt an AWS CLI

- Rufen Sie den AWS-CLI-Befehl describe-db-clusters auf und stellen Sie die folgenden Werte bereit:
  - `--db-cluster-identifier`: Name des DB-Clusters.

  Das folgende Beispiel listet Rückverfolgungsinformationen für au `sample-cluster`.

  Für Linux, macOS oder Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

  Für Windows:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

RDS-API

Um die Rückverfolgungsinformationen für einen DB-Cluster über die Amazon-RDS-API anzuzeigen, verwenden Sie die Aktion DescribeDBClusters. Diese Operation gibt Rückverfolgungsinformationen für den mit dem Wert `DBClusterIdentifier` angegebenen DB-Cluster zurück.

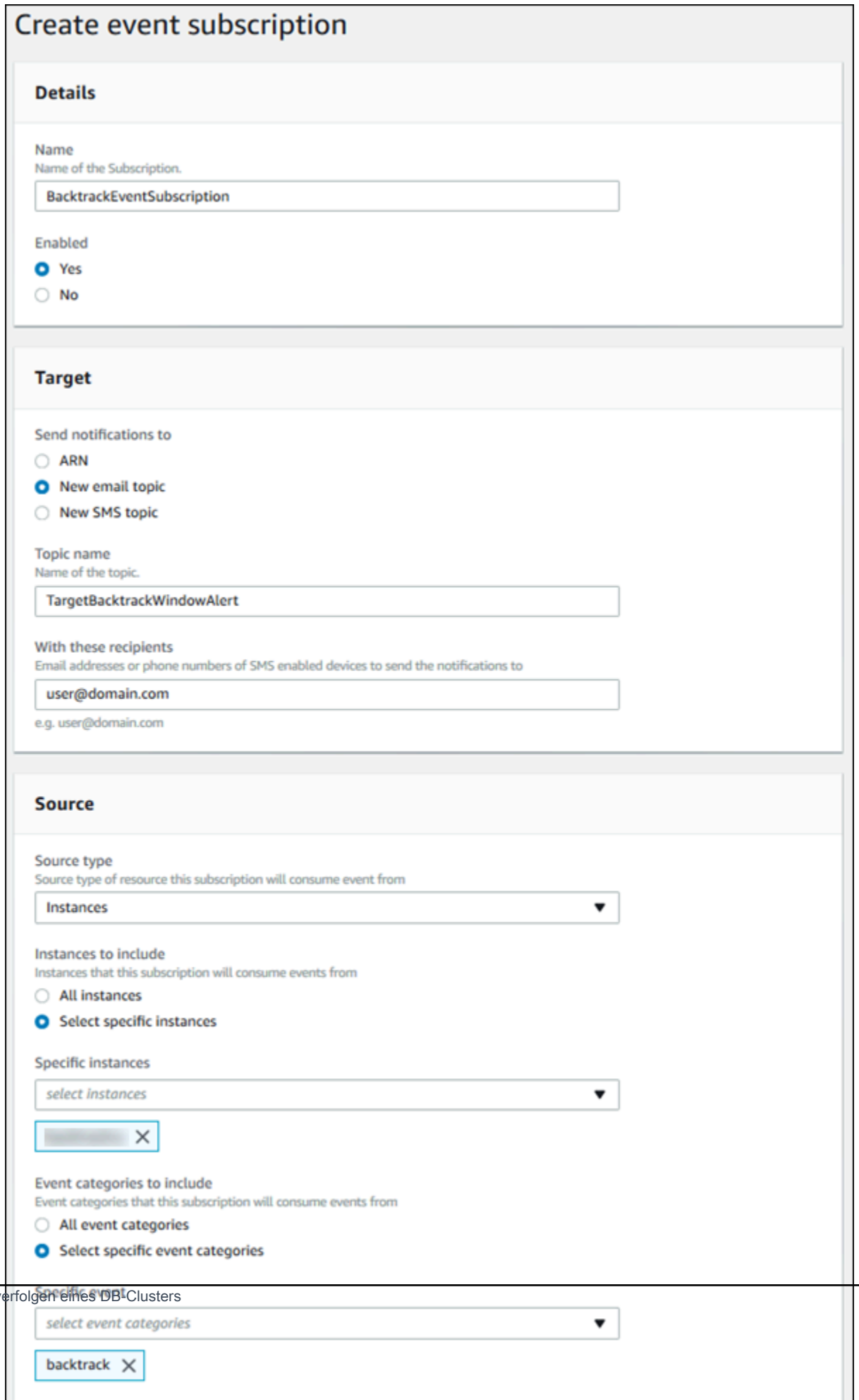
Abonnieren eines Rückverfolgungsereignisses mit der Konsole

Das folgende Verfahren beschreibt, wie Sie ein Rückverfolgungsereignis mit der Konsole abonnieren. Das Ereignis sendet eine E-Mail oder SMS, wenn das Ist-Rückverfolgungsfenster kürzer als das Zielfenster für die Rückverfolgung ist.

Zeigen Sie Rückverfolgungsinformationen mit der Konsole wie folgt an:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Event subscriptions (Ereignisabonnements) aus.
3. Wählen Sie Create event subscription (Ereignisabonnement erstellen) aus.
4. Geben Sie in das Feld Name (Name) einen Namen für das Ereignisabonnement ein und stellen Sie sicher, dass für Enabled (Aktiviert) der Wert Yes (Ja) ausgewählt ist.
5. Wählen Sie im Bereich Target (Ziel) die Option New email topic (Neues E-Mail-Thema) aus.
6. Geben Sie unter Topic name (Themenname) einen Namen für das Thema und unter With these recipients (Mit diesen Empfängern) die E-Mail-Adressen oder Telefonnummern der Benachrichtigungsempfänger ein.
7. Wählen Sie im Bereich Source (Quelle) für Source type (Quellentyp) den Wert Instances (Instances) aus.
8. Wählen Sie für Instances to include (Einzuschließende Instances) den Wert Select specific instances (Bestimmte Instances auswählen) und dann die DB-Instance aus.
9. Wählen Sie für Event categories to include (Einzuschließende Ereigniskategorien) den Wert Select specific event categories (Bestimmte Ereigniskategorien auswählen) und dann Backtrack (Rückverfolgen) aus.

Die Seite sollte der folgenden Seite ähneln.


Create event subscription
Details
Name
Name of the Subscription.
BacktrackEventSubscription
Enabled
Yes
No
Target
Send notifications to
ARN
New email topic
New SMS topic
Topic name
Name of the topic.
TargetBacktrackWindowAlert
With these recipients
Email addresses or phone numbers of SMS enabled devices to send the notifications to
user@domain.com
e.g. user@domain.com
Source
Source type
Source type of resource this subscription will consume event from
Instances
Instances to include
Instances that this subscription will consume events from
All instances
Select specific instances
Specific instances
select instances
Event categories to include
Event categories that this subscription will consume events from
All event categories
Select specific event categories
select event categories
backtrack

10. Wählen Sie Create aus.

Abrufen vorhandener Rückverfolgungen

Sie können Informationen über existierende Rückverfolgungen für einen DB-Cluster abrufen. Zu diesen Informationen gehören die eindeutige ID der Rückverfolgung, Ausgangs- und Zielzeitpunkt der Rückverfolgung, Datum und Zeitpunkt der Anforderung der Rückverfolgung und der aktuelle Status der Rückverfolgung.

> **Note**
>
> Derzeit können vorhandene Rückverfolgungen nicht mit der Konsole abgerufen werden.

AWS CLI

Das folgende Verfahren beschreibt, wie Sie vorhandene Rückverfolgungen für einen DB-Cluster mit der abrufe AWS CLI.

Um vorhandene Backtracks mit dem abzurufen AWS CLI

- Rufen Sie den describe-db-cluster-backtracks AWS CLI-Befehl auf und geben Sie die folgenden Werte an:
  - `--db-cluster-identifier`: Name des DB-Clusters.

  Im folgenden Beispiel werden die für vorhandenen Rückverfolgungen abgerufe `sample-cluster`.

  Für Linux, macOS oder Unix:

```
aws rds describe-db-cluster-backtracks \
    --db-cluster-identifier sample-cluster
```

  Für Windows:

```
aws rds describe-db-cluster-backtracks ^
```

```
    --db-cluster-identifier sample-cluster
```

RDS-API

Um Informationen über die Backtracks für einen DB-Cluster mithilfe der Amazon RDS-API abzurufen, verwenden Sie den Vorgang Describe DBCluster Backtracks. Diese Operation gibt Informationen über Rückverfolgungen für den mit dem Wert `DBClusterIdentifier` angegebenen DB-Cluster zurück.

Deaktivieren der Rückverfolgung für einen DB-Cluster von Aurora MySQL

Sie können die Rückverfolgungsfunktion für einen DB-Cluster deaktivieren.

Konsole

Sie können die Rückverfolgung für einen DB-Cluster mit der Konsole deaktivieren. Nachdem Sie die Rückverfolgung für einen Cluster vollständig deaktiviert haben, können Sie sie für diesen Cluster nicht erneut aktivieren.

Deaktivieren Sie die Rückverfolgungsfunktion für einen DB-Cluster mit der Konsole wie folgt:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie den zu ändernden Cluster und dann die Option Modify (Ändern) aus.
4. Wählen Sie im Bereich Backtrack (Rückverfolgung) die Option Disable Backtrack (Rückverfolgung deaktivieren) aus.
5. Klicken Sie auf Weiter.
6. Wählen Sie für Einplanung von Änderungen eine der folgenden Optionen:
   - Apply during the next scheduled maintenance window (Während des nächsten geplanten Wartungsfensters anwenden): Die Änderung soll erst während des nächsten Wartungsfensters ausgeführt werden.
   - Apply immediately (Sofort anwenden): Die Änderung soll sobald wie möglich angewendet werden.
7. Wählen Sie Modify Cluster (Cluster ändern) aus.

AWS CLI

Sie können die Rückverfolgungsfunktion für ein DB-Cluster über die AWS CLI deaktivieren, indem Sie das Zielfenster für die Rückverfolgung auf `0` (null) festlegen. Nachdem Sie die Rückverfolgung für einen Cluster vollständig deaktiviert haben, können Sie sie für diesen Cluster nicht erneut aktivieren.

Ändern Sie das Zeitfenster für die Rückverfolgung für einen DB-Cluster mit der wie folgt AWS CLI

- Rufen Sie den AWS-CLI-Befehl modify-db-cluster auf und geben Sie folgende Werte an:
  - `--db-cluster-identifier`: Name des DB-Clusters.
  - `--backtrack-window` – Geben Sie `0` an, um die Rückverfolgung zu deaktivieren.

  Im folgenden Beispiel wird die Rückverfolgungsfunktion für `sample-cluster` deaktiviert, indem `--backtrack-window` der Wert `0` zugewiesen wird.

  Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 0
```

  Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 0
```

RDS-API

Um die Rückverfolgungsfunktion für einen DB-Cluster über die Amazon-RDS-API zu deaktivieren, verwenden Sie die Aktion ModifyDBCluster. Weisen Sie `BacktrackWindow` den Wert `0` (Null) zu und geben Sie den DB-Cluster im Wert `DBClusterIdentifier` an. Nachdem Sie die Rückverfolgung für einen Cluster vollständig deaktiviert haben, können Sie sie für diesen Cluster nicht erneut aktivieren.

Testen von Amazon Aurora MySQL unter Verwendung von Fehlersimulationsabfragen

Sie können die Fehlertoleranz Ihres DB-Clusters von Aurora MySQL testen, indem Sie Fehlersimulationsabfragen verwenden. Fehlersimulationsabfragen werden als SQL-Befehle an eine Amazon Aurora-Instance ausgegeben. Sie ermöglichen Ihnen, eines der folgenden Fehlerereignisse geplant zu simulieren:

- Einen Ausfall einer Writer- oder Reader-DB-Instance
- Ein Ausfall eines Aurora-Replikats
- Einen Festplattenfehler
- Festplattenüberlastung

Wenn eine Fehlersimulationsabfrage einen Absturz angibt, erzwingt sie einen Absturz der DB-Instance von Aurora MySQL. Die anderen Fehler-Injection-Abfragen erzeugen Simulationen von Ausfallereignissen, lösen aber keine Ereignisse aus. Wenn Sie eine Fehlersimulationsabfrage senden, geben Sie auch einen Zeitraum vor, in dem die Simulation ablaufen soll.

Sie können eine Fehlersimulationsabfrage an eine Ihrer Aurora Replica-Instances senden, indem Sie eine Verbindung mit dem Endpunkt der Aurora Replica herstellen. Weitere Informationen finden Sie unter Amazon-Aurora-Endpunktverbindungen.

Zum Ausführen von Fehlersimulationsabfragen sind alle Master-Benutzerrechte erforderlich. Weitere Informationen finden Sie unter Berechtigungen von Hauptbenutzerkonten.

Testen eines Instance-Ausfalls

Sie können den Absturz einer Amazon-Aurora-Instance erzwingen, indem Sie die Fehlersimulationsabfrage `ALTER SYSTEM CRASH` verwenden.

Für diese Fehlersimulationsabfrage tritt kein Failover auf. Wenn Sie einen Failover testen möchten, können Sie die Instance-Operation Failover für Ihren DB-Cluster in der RDS-Konsole ausführen oder den AWS CLI-Befehl failover-db-cluster bzw. die RDS-API-Operation FailoverDBCluster verwenden.

Syntax

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

Optionen

Diese Fehlersimulationsabfrage löst einen der folgenden Ausfalltypen aus:

- **`INSTANCE`** – Es wird ein Absturz der PostgreSQL-kompatiblen Datenbank für die Amazon-Aurora-Instance simuliert.
- **`DISPATCHER`** – Es wird ein Absturz des Dispatchers auf Schreiber-Instance für das Aurora-DB-Cluster simuliert. Der Dispatcher schreibt Updates zum Cluster-Volume für ein Amazon Aurora-DB-Cluster.
- **`NODE`** – Es wird ein Absturz sowohl der MySQL-kompatiblen Datenbank als auch des Dispatchers für die Amazon-Aurora-Instance simuliert. Für diese Fehlersimulationsabfrage wird auch der Cache gelöscht.

Der Standard-Ausfalltyp ist `INSTANCE`.

Testen eines Aurora-Replikatausfalls

Sie können den Ausfall eines Aurora-Replikats mittels der Fehlersimulationsabfrage `ALTER SYSTEM SIMULATE READ REPLICA FAILURE` simulieren.

Der Ausfall eines Aurora Replica blockiert alle Anfragen an einen Aurora Replica oder alle Aurora Replicas im DB-Cluster für einen bestimmten Zeitabschnitt. Wenn dieser abgelaufen ist, werden die betroffenen Aurora-Replikate automatisch mit der Schreiber-Instance synchronisiert.

Syntax

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
    [ TO ALL | TO "replica name" ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Optionen

Diese Fehlersimulationsabfrage verwendet die folgenden Parameter:

- **`percentage_of_failure`** – Der Prozentsatz der Anfragen, die während des Ausfallereignisses blockiert werden sollen. Dieser Wert kann ein Duplikat zwischen 0 und 100 sein. Wenn Sie 0 festlegen, werden keine Anfragen blockiert. Wenn 100 festgelegt wird, werden alle Anfragen blockiert.

- Ausfalltyp – Der zu simulierende Ausfalltyp. Legen Sie `TO ALL` fest, um Ausfälle für alle Aurora-Replikate im DB-Cluster zu simulieren. Geben Sie `TO` und den Namen eines Aurora-Replikats an, um einen Ausfall des einzelnen Aurora-Replikats zu simulieren. Der Standard-Ausfalltyp ist `TO ALL`.
- **`quantity`** – Der Zeitraum, über den der Ausfall des Aurora-Replikats simuliert werden soll. Das Intervall wird als eine Menge gefolgt von einer Zeiteinheit angegeben. Die Simulation wird für die Dauer der angegebenen Einheit auftreten. Beispielsweise ergibt `20 MINUTE` eine Simulation, die 20 Minuten lang ausgeführt wird.

> **Note**
>
> Seien Sie vorsichtig bei der Angabe des Zeitintervalls für das Ausfallereignis in Ihrem Aurora-Replikat. Wenn Sie ein zu langes Zeitintervall festlegen und Ihre Schreiber-Instance eine große Datenmenge während des Ausfallereignisses schreibt, könnte Ihr Aurora-DB-Cluster annehmen, dass Ihr Aurora-Replikat ausgefallen ist und es ersetzen.

Testen eines Festplattenausfalls

Sie können einen Datenträgerausfall für ein Aurora-DB-Cluster mittels der Fehlersimulationsabfrage `ALTER SYSTEM SIMULATE DISK FAILURE` simulieren.

Während einer Simulation eines Festplattenausfalls markiert der Aurora-DB-Cluster zufällige Festplattensegmente als fehlerhaft. Anfragen an diese Segmente werden für die Dauer der Simulation blockiert.

Syntax

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Optionen

Diese Fehlersimulationsabfrage verwendet die folgenden Parameter:

- **`percentage_of_failure`** – Der Prozentsatz des Datenträgers, der während des Ausfallereignisses als ausgefallen markiert werden soll. Dieser Wert kann ein Duplikat zwischen 0

und 100 sein. Wenn Sie 0 festlegen, wird kein Segment der Festplatte als fehlerhaft markiert. Wenn Sie 100 festlegen, wird die gesamte Festplatte als fehlerhaft markiert.

- **`DISK index`** – Ein spezifischer logischer Datenblock, für den das Ausfallereignis simuliert werden soll. Wenn Sie den verfügbaren Bereich an logischen Datenblocks überschreiten, erhalten Sie eine Fehlermeldung, in der Ihnen der maximal festlegbare Indexwert mitgeteilt wird. Weitere Informationen finden Sie unter Anzeigen des Volume-Status für einen Aurora MySQL-DB-Cluster.
- **`NODE index`** – Ein spezifischer Speicherknoten, für den das Ausfallereignis simuliert werden soll. Wenn Sie den verfügbaren Bereich an Speicherknoten überschreiten, erhalten Sie eine Fehlermeldung, in der Ihnen der maximal festlegbare Indexwert mitgeteilt wird. Weitere Informationen finden Sie unter Anzeigen des Volume-Status für einen Aurora MySQL-DB-Cluster.
- **`quantity`** – Der Zeitraum, über den der Datenträgerausfall simuliert werden soll. Das Intervall wird als eine Menge gefolgt von einer Zeiteinheit angegeben. Die Simulation wird für die Dauer der angegebenen Einheit auftreten. Beispielsweise ergibt `20 MINUTE` eine Simulation, die 20 Minuten lang ausgeführt wird.

Testen einer Festplattenüberlastung

Sie können einen Datenträgerausfall für ein Aurora-DB-Cluster mittels der Fehlersimulationsabfrage `ALTER SYSTEM SIMULATE DISK CONGESTION` simulieren.

Während einer Simulation einer Festplattenüberlastung markiert der Aurora-DB-Cluster zufällige Festplattensegmente als überlastet. Anfragen an diese Segmente werden für die Dauer der angegebenen minimalen und maximalen Verzögerungszeit während der Simulation verzögert.

Syntax

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
    BETWEEN minimum AND maximum MILLISECONDS
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Optionen

Diese Fehlersimulationsabfrage verwendet die folgenden Parameter:

- **`percentage_of_failure`** – Der Prozentsatz des Datenträgers, der während des Ausfallereignisses als überlastet markiert werden soll. Dieser Wert kann ein Duplikat zwischen

0 und 100 sein. Wenn Sie 0 festlegen, wird kein Segment der Festplatte als überlastet markiert. Wenn Sie 100 festlegen, wird die gesamte Festplatte als überlastet markiert.

- **`DISK index`** oder **`NODE index`** – Ein spezifischer Datenträger oder Knoten, für den das Ausfallereignis simuliert werden soll. Wenn Sie den verfügbaren Bereich der Indizes für die Festplatten oder den Knoten überschreiten, erhalten Sie eine Fehlermeldung, in der Ihnen der maximal festlegbare Indexwert mitgeteilt wird.
- **`minimum`** und **`maximum`** – Der Mindest- und Höchstwert in Millisekunden für die Überlastungsverzögerung. Festplattensegmente, die als überlastet markiert sind, werden innerhalb des Bereichs zwischen der minimalen und maximalen Anzahl an Millisekunden während der Simulation verzögert.
- **`quantity`** – Der Zeitraum, über den die Datenträgerüberlastung simuliert werden soll. Das Intervall wird als eine Menge gefolgt von einer Zeiteinheit angegeben. Die Simulation wird für die Dauer der angegebenen Zeiteinheit ausgeführt. Beispielsweise ergibt `20 MINUTE` eine Simulation, die 20 Minuten lang ausgeführt wird.

Ändern von Tabellen in Amazon Aurora mithilfe von Fast DDL

Amazon Aurora umfasst Optimierungen, um einen `ALTER TABLE`-Vorgang nahezu sofort auszuführen. Der Vorgang schließt ab, ohne dass ein Kopieren der Tabelle erforderlich ist und ohne eine materielle Auswirkung auf andere DML-Statements zu haben. Da kein temporärer Speicher für eine Tabellenkopie benötigt wird, haben DDL-Statements praktische Vorteile, sogar für große Tabellen auf kleinen Instance-Klassen.

Aurora MySQL Version 3 ist mit der MySQL 8.0-Funktion namens Instant DDL kompatibel. Aurora MySQL Version 2 verwendet eine andere Implementierung namens Fast DDL.

Themen

- Sofortige DDL (Aurora MySQL Version 3)
- Fast DDL (Aurora MySQL Version 2)

Sofortige DDL (Aurora MySQL Version 3)

Die von Aurora MySQL Version 3 durchgeführte Optimierung zur Verbesserung der Effizienz einiger DDL-Vorgänge wird als Instant DDL bezeichnet.

Aurora MySQL Version 3 ist mit der Instant DDL von Community MySQL 8.0 kompatibel. Sie führen einen sofortigen DDL-Vorgang durch, indem Sie die Klausel `ALGORITHM=INSTANT` mit der `ALTER TABLE`-Anweisung verwenden. Weitere Informationen zu Syntax und Verwendung von Instant DDL finden Sie unter ALTER TABLE und Online DDL Operations in der MySQL-Dokumentation.

Die folgenden Beispiele veranschaulichen die Instant-DDL-Funktion. Die `ALTER TABLE`-Anweisungen fügen Spalten hinzu und ändern die Standardspaltenwerte. Die Beispiele umfassen sowohl reguläre als auch virtuelle Spalten sowie reguläre und partitionierte Tabellen. Bei jedem Schritt können Sie die Ergebnisse durch die Ausgabe von `SHOW CREATE TABLE`- und `DESCRIBE`-Anweisungen anzeigen.

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM =
 INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM
 = INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t3 (a INT, b INT) PARTITION BY LIST(a)(
    ->   PARTITION mypart1 VALUES IN (1,3,5),
    ->   PARTITION MyPart2 VALUES IN (2,4,6)
    -> );
```

```
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
 ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
    ->   (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
    ->   PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
    ->   PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->       PARTITION p0 VALUES LESS THAN (1990),
    ->       PARTITION p1 VALUES LESS THAN (2000),
    ->       PARTITION p2 VALUES LESS THAN MAXVALUE
    ->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

Fast DDL (Aurora MySQL Version 2)

Schnelle DDL in Aurora MySQL ist eine Optimierung, mit der die Leistung bestimmter Schemaänderungen, wie das Hinzufügen oder Löschen von Spalten, verbessert werden soll, indem die Ausfallzeit und der Ressourcenverbrauch reduziert werden. Dadurch können diese Operationen im Vergleich zu herkömmlichen DDL-Methoden effizienter ausgeführt werden.

> **Important**
>
> Zurzeit müssen Sie den Aurora-Labormodus aktivieren, um schnelle DDL verwenden zu können. Weitere Informationen zum Aktivieren des Labormodus finden Sie unter Amazon Aurora MySQL-Labor-Modus.

Die Optimierung für schnelle DDL wurde ursprünglich im Labormodus für Aurora-MySQL-Version 2 eingeführt, um die Effizienz bestimmter DDL-Operationen zu verbessern. In Aurora-MySQL-Version 3 wurde der Labormodus eingestellt und schnelle DDL wurde durch die MySQL-8.0-Funktion für sofortige DDL ersetzt.

In MySQL wirken sich viele Data Definition Language (DDL)-Vorgänge signifikant auf die Leistungsfähigkeit aus.

Nehmen wir beispielsweise an, dass Sie eine `ALTER TABLE`-Operation verwenden, um eine Spalte zu einer Tabelle hinzuzufügen. Abhängig vom angegebenen Algorithmus für den Vorgang kann der Vorgang Folgendes beinhalten:

- Erstellen einer vollen Kopie für die Tabelle
- Erstellen einer temporären Tabelle für die Verarbeitung simultaner Data Manipulation Language (DML)-Vorgänge
- Erneutes Aufbauen aller Indizes für die Tabelle
- Anwenden von Tabellensperren beim Vornehmen von simultanen DML-Änderungen
- Verlangsamen des simultanen DML-Durchsatzes

Diese Leistungsbeeinträchtigung kann in Umgebungen mit großen Tabellen oder hohem Transaktionsvolumen eine besondere Herausforderung darstellen. Schnelle DDL trägt dazu bei, diese Herausforderungen zu minimieren, indem Schemaänderungen optimiert werden, was schnellere und weniger ressourcenintensive Operationen ermöglicht.

Schnelle DDL-Beschränkungen

Aktuell hat Fast DDL folgende Einschränkungen:

- Sie unterstützt nur das Hinzufügen von löschbaren Spalten, ohne Standardwerte, bis zum Ende einer bestehenden Tabelle.
- Schnelle DDL funktioniert nicht für partitionierte Tabellen.
- Schnelle DDL funktioniert nicht für InnoDB-Tabellen, die das Zeilenformat REDUNDANT verwenden.
- Schnelle DDL funktioniert nicht für Tabellen mit Volltextsuchindizes.
- Wenn die maximal mögliche Datensatzgröße für die DDL-Operation zu groß ist, wird Fast DDL nicht verwendet. Eine Datensatzgröße ist zu groß, wenn sie größer als die Hälfte der Seitengröße

ist. Die maximale Größe eines Datensatzes wird berechnet, indem die maximale Größe aller Spalten addiert wird. Bei Spalten variabler Größe werden nach InnoDB-Standards externe Bytes für die Berechnung nicht berücksichtigt.

Schnelle DDL-Syntax

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

Dieses Statement verwendet die folgenden Optionen:

- **`tbl_name`** — Der Name der Tabelle, die geändert werden soll.
- **`col_name`** — Der Name der Spalte, die hinzugefügt werden soll.
- **`col_definition`** — Die Definition der Spalte, die hinzugefügt werden soll.

> **Note**
>
> Sie müssen eine löschbare Spaltendefinition ohne einen Standardwert festlegen. Andernfalls kann Fast DDL nicht verwendet werden.

Fast DDL-Beispiele

Die folgenden Beispiele veranschaulichen die Beschleunigung von Fast-DDL-Operationen. Im ersten SQL-Beispiel werden `ALTER TABLE`-Anweisungen für eine große Tabelle ausgeführt, ohne Fast DDL zu verwenden. Diese Operation benötigt beträchtliche Zeit. Ein CLI-Beispiel zeigt, wie Fast DDL für den Cluster aktiviert wird. Dann führt ein anderes SQL-Beispiel dieselben `ALTER TABLE` Anweisungen für eine identische Tabelle aus. Mit aktiviertem Fast DDL ist der Betrieb sehr schnell.

In diesem Beispiel wird die `ORDERS` Tabelle aus dem TPC-H-Benchmark verwendet, die 150 Millionen Zeilen enthält. Dieser Cluster verwendet absichtlich eine relativ kleine Instance-Klasse, um zu demonstrieren, wie lange `ALTER TABLE`-Anweisungen dauern können, wenn Sie Fast DDL nicht verwenden können. Im Beispiel wird ein Klon der Originaltabelle erstellt, der identische Daten enthält. Eine Überprüfung der Einstellung für `aurora_lab_mode` bestätigt, dass der Cluster Fast DDL nicht verwenden kann, da der Labormodus nicht aktiviert ist. Dann brauchen `ALTER TABLE ADD COLUMN` Anweisungen beträchtliche Zeit, um am Ende der Tabelle neue Spalten hinzuzufügen.

```
mysql> create table orders_regular_ddl like orders;
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+------------------+
| @@aurora_lab_mode |
+------------------+
|                0 |
+------------------+

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

In diesem Beispiel wird die gleiche Vorbereitung einer großen Tabelle wie im vorherigen Beispiel durchgeführt. Sie können den Labor-Modus jedoch nicht einfach innerhalb einer interaktiven SQL-Sitzung aktivieren. Diese Einstellung muss in einer benutzerdefinierten Parametergruppe aktiviert sein. Dazu müssen Sie die `mysql` Sitzung verlassen und einige AWS CLI-Befehle ausführen oder die AWS-Managementkonsole verwenden.

```
mysql> create table orders_fast_ddl like orders;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

Das Aktivieren des Labor-Modus für den Cluster erfordert einige Arbeiten an einer Parametergruppe. In diesem Beispiel für eine AWS CLI wird eine Clusterparametergruppe verwendet, um sicherzustellen, dass alle DB-Instances im Cluster den gleichen Wert für die Einstellung des Labor-Modus verwenden.

```
$ aws rds create-db-cluster-parameter-group \
  --db-parameter-group-family aurora5.7 \
    --db-cluster-parameter-group-name lab-mode-enabled-57 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
```

```
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].[ParameterName,ParameterValue]' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 0
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-57"
}

# Assign the custom parameter group to the cluster that's going to use Fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
  --db-cluster-parameter-group-name lab-mode-enabled-57
{
  "DBClusterIdentifier": "tpch100g",
  "DBClusterParameterGroup": "lab-mode-enabled-57",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2",
  "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
  "DBInstanceIdentifier": "instance-2020-12-22-5208",
  "DBInstanceStatus": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
  --query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-57

$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 1
```

Das folgende Beispiel zeigt die verbleibenden Schritte, nachdem die Änderung der Parametergruppe wirksam wird. Es testet die Einstellung für `aurora_lab_mode`, um sicherzustellen, dass der Cluster Fast DDL verwenden kann. Dann führt es `ALTER TABLE` Anweisungen aus, um Spalten am Ende

einer anderen großen Tabelle hinzuzufügen. Dieses Mal werden die Anweisungen sehr schnell beendet.

```
mysql> select @@aurora_lab_mode;
+------------------+
| @@aurora_lab_mode |
+------------------+
|                1 |
+------------------+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (0.40 sec)
```

Anzeigen des Volume-Status für einen Aurora MySQL-DB-Cluster

In Amazon Aurora besteht ein DB-Cluster-Volume aus einer Sammlung von logischen Blöcken. Jeder von ihnen stellt 10 Gigabyte des zugeteilten Arbeitsspeichers bereit. Diese Blöcke werden als Schutzgruppen bezeichnet.

Die Daten in den einzelnen Schutzgruppen werden über sechs physische Speichereinheiten, so genannte Speicherknoten, repliziert. Diese Speicherknoten werden in drei Availability Zones (AZs) in der AWS-Region zugeteilt, in der sich das DB-Cluster befindet. Jeder Speicherknoten wiederum besteht aus einem oder mehreren logischen Datenblöcken für das DB-Cluster-Volume. Weitere Informationen zu Schutzgruppen und Speicherknoten finden Sie unter Introducing the Aurora Storage Engine im AWS Database Blog.

Sie können den Ausfall eines gesamten Speicherknotens oder den eines einzelnen Datenblocks innerhalb eines Speicherknotens simulieren. Verwenden Sie hierfür die Fehlersimulationsanweisung `ALTER SYSTEM SIMULATE DISK FAILURE`. Sie geben für die Anweisung den Indexwert eines spezifischen logischen Datenblocks oder Speicherknotens an. Wenn Sie jedoch einen Indexwert angeben, der größer als die Anzahl der vom DB-Cluster-Volume verwendeten logischen Datenblöcke oder Speicherknoten ist, gibt die Anweisung einen Fehler zurück. Weitere Informationen über Fehlersimulationsabfragen finden Sie unter Testen von Amazon Aurora MySQL unter Verwendung von Fehlersimulationsabfragen.

Sie können diesen Fehler vermeiden, indem Sie die Anweisung `SHOW VOLUME STATUS` verwenden. Die Anweisung gibt zwei Serverstatusvariablen zurück, `Disks` und `Nodes`. Diese Variablen stellen

jeweils die gesamte Anzahl an logischen Datenblocks und Speicherknoten für das DB-Cluster-Volume dar.

Syntax

```
SHOW VOLUME STATUS
```

Beispiel

Das folgende Beispiel zeigt ein typisches Ergebnis von SHOW VOLUME STATUS.

```
mysql> SHOW VOLUME STATUS;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Disks         | 96    |
| Nodes         | 74    |
+---------------+-------+
```

Optimieren von Aurora MySQL

Warteereignisse und Thread-Zustände sind wichtige Optimierungs-Tools für Aurora MySQL. Wenn Sie herausfinden, warum Sitzungen auf Ressourcen warten und was sie tun, können Sie Engpässe besser reduzieren. Anhand der Informationen in diesem Abschnitt können Sie mögliche Ursachen und Abhilfemaßnahmen ermitteln.

Amazon DevOps Guru für RDS kann proaktiv feststellen, ob in Ihren Aurora-MySQL-Datenbanken problematische Bedingungen vorliegen, die später zu größeren Problemen führen könnten. Amazon DevOps Guru für RDS veröffentlicht eine Erklärung und Empfehlungen für Korrekturmaßnahmen in einem proaktiven Einblick. Dieser Abschnitt enthält Einblicke in Bezug auf häufig auftretende Probleme.

> **Important**
>
> Die Warteereignisse und Thread-Zustände in diesem Abschnitt sind spezifisch für Aurora MySQL. Verwenden Sie die Informationen in diesem Abschnitt, um nur Amazon Aurora zu optimieren, nicht Amazon RDS for MySQL.
> Einige Warteereignisse in diesem Abschnitt haben keine Entsprechungen in den Open-Source-Versionen dieser Datenbank-Engines. Andere Warteereignisse haben dieselben Namen wie Ereignisse in Open-Source-Engines, verhalten sich jedoch anders. Beispielsweise funktioniert Amazon-Aurora-Speicher anders als Open-Source-Speicher, sodass speicherbezogene Warteereignisse auf unterschiedliche Ressourcenbedingungen hinweisen.

Themen

- Grundlegende Konzepte für Aurora-MySQL-Optimierung
- Optimieren von Aurora MySQL mit Warteereignissen
- Optimierung von Aurora MySQL mit Thread-Status
- Optimierung von Aurora MySQL mit proaktiven Einblicken von Amazon DevOps Guru

Grundlegende Konzepte für Aurora-MySQL-Optimierung

Bevor Sie Ihre Aurora-MySQL-Datenbank optimieren, stellen Sie sicher, dass Sie wissen, was Warteereignisse und Thread-Zustände sind und warum sie auftreten. Überprüfen Sie

auch die grundlegende Speicher- und Festplattenarchitektur von Aurora MySQL, wenn Sie die InnoDB-Speicher-Engine verwenden. Ein hilfreiches Architekturdiagramm finden Sie im MySQL-Referenzhandbuch.

Themen

- Aurora-MySQL-Warteereignisse
- Aurora-MySQL-Thread-Zustände
- Aurora-MySQL-Speicher
- Aurora-MySQL-Prozesse

Aurora-MySQL-Warteereignisse

Ein Warteereignis zeigt eine Ressource an, auf die eine Sitzung wartet. Das Warteereignis `io/socket/sql/client_connection` zeigt beispielsweise an, dass ein Thread gerade eine neue Verbindung verarbeitet. Zu den typischen Ressourcen, auf die eine Sitzung wartet, gehören die folgenden:

- Singlethread-Zugriff auf einen Puffer, beispielsweise wenn eine Sitzung versucht, einen Puffer zu ändern
- Eine Zeile, die derzeit von einer anderen Sitzung gesperrt ist
- Eine gelesene Datendatei
- Eine geschriebene Protokolldatei

Um beispielsweise eine Abfrage zu erfüllen, kann die Sitzung einen vollständigen Tabellenscan durchführen. Wenn sich die Daten noch nicht im Arbeitsspeicher befinden, wartet die Sitzung, bis die Datenträger-I/O abgeschlossen ist. Wenn die Puffer in den Speicher gelesen werden, muss die Sitzung möglicherweise warten, da andere Sitzungen auf dieselben Puffer zugreifen. Die Datenbank zeichnet die Wartezeiten unter Verwendung eines vordefinierten Warteereignisses auf. Diese Ereignisse sind in Kategorien eingeteilt.

Ein Wait-Ereignis allein zeigt kein Leistungsproblem an. Wenn sich beispielsweise die angeforderten Daten nicht im Speicher befinden, müssen die Daten von der Festplatte gelesen werden. Wenn eine Sitzung eine Zeile für eine Aktualisierung sperrt, wartet eine andere Sitzung darauf, dass die Zeile entsperrt wird, damit sie sie aktualisieren kann. Bei einem Commit muss gewartet werden, bis der Schreibvorgang in eine Protokolldatei abgeschlossen ist. Wartezeiten sind ein wesentlicher Bestandteil des normalen Funktionierens einer Datenbank.

Eine große Anzahl von Warteereignissen weist normalerweise auf ein Leistungsproblem hin. In solchen Fällen können Sie Warteereignisdaten verwenden, um zu bestimmen, wo die Sitzungen Zeit verbringen. Wenn beispielsweise ein Bericht, der normalerweise in Minuten ausgeführt wird, jetzt stundenlang ausgeführt wird, können Sie die Warteereignisse identifizieren, die am meisten zur Gesamtwartezeit beitragen. Wenn Sie die Ursachen für die häufigsten Warteereignisse ermitteln können, können Sie manchmal Änderungen vornehmen, die die Leistung verbessern. Wenn Ihre Sitzung beispielsweise auf eine Zeile wartet, die von einer anderen Sitzung gesperrt wurde, können Sie die Sperrsitzung beenden.

Aurora-MySQL-Thread-Zustände

Ein allgemeiner Thread-Zustand ist ein `State`-Wert, der der allgemeinen Abfrageverarbeitung zugeordnet ist. Der Thread-Zustand `sending data` gibt beispielsweise an, dass ein Thread Zeilen für eine Abfrage liest und filtert, um die richtige Ergebnismenge zu ermitteln.

Sie können Thread-Zustände verwenden, um Aurora MySQL auf ähnliche Weise zu optimieren, wie Sie Warteereignisse verwenden. Häufiges Vorkommen von `sending data` weist beispielsweise normalerweise darauf hin, dass eine Abfrage keinen Index verwendet. Weitere Informationen zu Thread-Zuständen finden Sie unter Allgemeine Thread-Zustände im MySQL-Referenzhandbuch.

Wenn Sie Performance Insights verwenden, gilt eine der folgenden Bedingungen:

- Performance Schema ist aktiviert – Aurora MySQL zeigt Warteereignisse anstelle des Thread-Zustands an.
- Performance-Schema ist nicht aktiviert – Aurora MySQL zeigt den Thread-Zustand an.

Es wird empfohlen, Performance Schema für die automatische Verwaltung zu konfigurieren. Das Leistungsschema bietet zusätzliche Einblicke und bessere Tools zur Untersuchung potenzieller Leistungsprobleme. Weitere Informationen finden Sie unter Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL.

Aurora-MySQL-Speicher

In Aurora MySQL sind die wichtigsten Speicherbereiche der Pufferpool und der Protokollpuffer.

Themen

- Puffer Pool

Puffer Pool

Der Pufferpool ist der gemeinsam genutzte Speicherbereich, in dem Aurora MySQL Tabellen- und Indexdaten zwischenspeichert. Abfragen können direkt aus dem Speicher auf häufig verwendete Daten zugreifen, ohne von der Festplatte zu lesen.

Der Pufferpool ist als verknüpfte Seitenliste strukturiert. Eine Seite kann mehrere Zeilen enthalten. Aurora MySQL verwendet einen LRU-Algorithmus (Least Latest Used), um Seiten aus dem Pool zu altern.

Weitere Informationen finden Sie unter Pufferpool im MySQL-Referenzhandbuch.

Aurora-MySQL-Prozesse

Aurora MySQL verwendet ein Prozessmodell, das sich stark von Aurora PostgreSQL unterscheidet.

Themen

- MySQL-Server (mysqld)
- Threads
- Thread-Pool

MySQL-Server (mysqld)

Der MySQL-Server ist ein einziger Betriebssystemprozess namens mysqld. Der MySQL-Server erzeugt keine zusätzlichen Prozesse. Daher verwendet eine Aurora-MySQL-Datenbank mysqld, um den größten Teil ihrer Arbeit zu erledigen.

Wenn der MySQL-Server startet, lauscht er auf Netzwerkverbindungen von MySQL-Clients. Wenn ein Client eine Verbindung zur Datenbank herstellt, öffnet mysqld einen Thread.

Threads

Verbindungsmanager-Threads verknüpfen jede Clientverbindung mit einem dedizierten Thread. Dieser Thread verwaltet die Authentifizierung, führt Anweisungen aus und gibt Ergebnisse an den Client zurück. Der Verbindungsmanager erstellt bei Bedarf neue Threads.

Der Thread-Cache ist die Menge der verfügbaren Threads. Wenn eine Verbindung endet, gibt MySQL den Thread an den Thread-Cache zurück, wenn der Cache nicht voll ist. Die `thread_cache_size`-Systemvariable bestimmt die Thread-Cache-Größe.

Thread-Pool

Der Thread-Pool besteht aus einer Reihe von Thread-Gruppen. Jede Gruppe verwaltet eine Reihe von Clientverbindungen. Wenn ein Client eine Verbindung mit der Datenbank herstellt, weist der Thread-Pool die Verbindungen Thread-Gruppen auf Round-Robin-Art zu. Der Thread-Pool trennt Verbindungen und Threads. Es gibt keine feste Beziehung zwischen Verbindungen und den Threads, die Anweisungen ausführen, die von diesen Verbindungen empfangen wurden.

Optimieren von Aurora MySQL mit Warteereignissen

Die folgende Tabelle fasst die Aurora-MySQL-Warteereignisse zusammen, die am häufigsten auf Leistungsprobleme hinweisen. Die folgenden Warteereignisse sind eine Teilmenge der Liste in Aurora-MySQL-Warteereignisse.

Warteereignis	Description
cpu	Dieses Ereignis tritt auf, wenn ein Thread in der CPU aktiv ist oder auf die CPU wartet.
io/aurora_redo_log_flush	Dieses Ereignis tritt ein, wenn eine Sitzung persistente Daten in den Aurora-Speicher schreibt.
io/aurora_respond_to_client	Das Ereignis tritt auf, wenn ein Thread darauf wartet, eine Ergebnismenge an einen Client zurückzugeben.
io/redo_log_flush	Dieses Ereignis tritt ein, wenn eine Sitzung persistente Daten in den Aurora-Speicher schreibt.
io/socket/sql/client_connection	Dieses Ereignis tritt auf, wenn ein Thread gerade eine neue Verbindung verarbeitet.
io/table/sql/handler	Dieses Ereignis tritt ein, wenn Arbeit an eine Speicher-Engine delegiert wurde.
synch/cond/innodb/row_lock_wait	Dieses Ereignis tritt auf, wenn eine Sitzung eine Zeile für eine Aktualisierung gesperrt hat

Warteereignis	Description
	und eine andere Sitzung versucht, dieselbe Zeile zu aktualisieren.
synch/cond/innodb/row_lock_wait_cond	Dieses Ereignis tritt auf, wenn eine Sitzung eine Zeile für eine Aktualisierung gesperrt hat und eine andere Sitzung versucht, dieselbe Zeile zu aktualisieren.
synch/cond/sql/MDL_context::COND_wait_status	Dieses Ereignis tritt ein, wenn Threads auf eine Tabellenmetadatensperre warten.
synch/mutex/innodb/aurora_lock_thread_slot_futex	Dieses Ereignis tritt auf, wenn eine Sitzung eine Zeile für eine Aktualisierung gesperrt hat und eine andere Sitzung versucht, dieselbe Zeile zu aktualisieren.
synch/mutex/innodb/buf_pool_mutex	Dieses Ereignis tritt ein, wenn ein Thread eine Sperre des InnoDB-Puffer-Pools für den Zugriff auf eine Seite im Speicher erworben hat.
synch/mutex/innodb/fil_system_mutex	Dieses Ereignis tritt auf, wenn eine Sitzung darauf wartet, auf den Tablespace-Speicherache zuzugreifen.
synch/mutex/innodb/trx_sys_mutex	Dieses Ereignis tritt auf, wenn eine hohe Datenbankaktivität mit einer großen Anzahl von Transaktionen besteht.
synch/sxlock/innodb/hash_table_locks	Dieses Ereignis tritt ein, wenn Seiten, die nicht im Pufferpool gefunden wurden, aus einer Datei gelesen werden müssen.
synch/mutex/innodb/temp_poolmanager_mutex	Dieses Ereignis tritt auf, wenn eine Sitzung darauf wartet, einen Mutex für die Verwaltung des Pools temporärer Sitzungs-Tablespaces zu erhalten.

cpu

Das cpu-Warteereignis tritt auf, wenn ein Thread in der CPU aktiv ist oder auf die CPU wartet.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Versionen 2 und 3

Kontext

Für jede vCPU kann eine Verbindung an dieser CPU arbeiten. In einigen Situationen ist die Anzahl der aktiven Verbindungen, die zum Ausführen bereit sind, höher als die Anzahl der vCPUs. Dieses Ungleichgewicht führt dazu, dass Verbindungen auf CPU-Ressourcen warten. Wenn die Anzahl der aktiven Verbindungen konstant höher bleibt als die Anzahl der vCPUs, erfährt Ihre Instance CPU-Konflikte. Der Konflikt bewirkt, dass das cpu-Warteereignis auftritt.

> **Note**
>
> Die Performance-Insights-Metrik für CPU ist `DBLoadCPU`. Der Wert für `DBLoadCPU` kann vom Wert für die CloudWatch-Metrik `CPUUtilization` abweichen. Die letztere Metrik wird vom HyperVisor für eine Datenbank-Instance gesammelt.

Performance-Insights-OS-Metriken liefern detaillierte Informationen zur CPU-Auslastung. Sie können beispielsweise die folgenden Metriken anzeigen:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`

- `os.cpuUtilization.idle.avg`

Performance Insights meldet die CPU-Auslastung durch die Datenbank-Engine als `os.cpuUtilization.nice.avg`.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

- Analytische Abfragen
- Hochgradig gleichzeitige Transaktionen
- Langlebige Transaktionen
- Ein plötzlicher Anstieg der Anzahl von Verbindungen, bekannt als Login-Sturm
- Eine Zunahme des Kontextwechsels

Aktionen

Wenn das `cpu`-Wait-Ereignis die Datenbankaktivität dominiert, weist dies nicht unbedingt auf ein Leistungsproblem hin. Reagieren Sie auf dieses Ereignis nur, wenn sich die Leistung verschlechtert.

Berücksichtigen Sie je nach Ursache des Anstiegs der CPU-Auslastung die folgenden Strategien:

- Erhöht die CPU-Kapazität des Hosts. Dieser Ansatz bietet normalerweise nur vorübergehende Erleichterung.
- Identifizieren Sie Top-Abfragen zur möglichen Optimierung.
- Leiten Sie ggf. einen schreibgeschützten Workload auf Reader-Knoten um.

Themen

- Identifizieren Sie die Sitzungen oder Abfragen, die das Problem verursachen
- Analysieren und optimieren Sie die hohe CPU-Workload

Identifizieren Sie die Sitzungen oder Abfragen, die das Problem verursachen

Um die Sitzungen und Abfragen zu finden, suchen Sie in der Top-SQL-Tabelle in Performance Insights nach den SQL-Anweisungen mit der höchsten CPU-Auslastung. Weitere Informationen finden Sie unter Analyse der Metriken mit dem Performance-Insights-Dashboard.

In der Regel verbrauchen ein oder zwei SQL-Anweisungen den Großteil der CPU-Zyklen. Konzentrieren Sie sich auf diese Anweisungen. Angenommen, Ihre DB-Instance verfügt über 2 vCPUs mit einer DB-Last von 3,1 durchschnittlichen aktiven Sitzungen (AAS), alle im CPU-Status. In diesem Fall ist Ihre Instance CPU-gebunden. Berücksichtigen Sie dabei die folgenden Strategien:

- Aktualisieren Sie auf eine größere Instance-Klasse mit mehr vCPUs.
- Stimmen Sie Ihre Abfragen auf eine geringere CPU-Last ein.

In diesem Beispiel haben die obersten SQL-Abfragen eine DB-Last von 1,5 AAS, alle im CPU-Status. Eine andere SQL-Anweisung hat eine Last von 0,1 im CPU-Status. Wenn Sie in diesem Beispiel die SQL-Anweisung mit der niedrigsten Last gestoppt haben, reduzieren Sie die Datenbanklast nicht wesentlich. Optimieren Sie jedoch die beiden Abfragen mit hoher Last, um doppelt so effizient zu sein, beseitigen Sie den CPU-Engpass. Wenn Sie die CPU-Last von 1,5 AAS um 50 Prozent reduzieren, verringert sich der AAS für jede Anweisung auf 0,75. Die gesamte für die CPU aufgewendete DB-Last beträgt jetzt 1,6 AAS. Dieser Wert liegt unter der maximalen vCPU-Linie von 2,0.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights. Siehe auch den AWS-Support-Artikel Wie kann ich eine hohe CPU-Auslastung auf meinen Amazon RDS for MySQL-Instances beheben und beheben?.

Analysieren und optimieren Sie die hohe CPU-Workload

Nachdem Sie die Abfrage oder Abfragen identifiziert haben, die die CPU-Auslastung erhöhen, können Sie sie entweder optimieren oder die Verbindung beenden. Das folgende Beispiel zeigt, wie Sie eine Verbindung beenden.

```
CALL mysql.rds_kill(processID);
```

Weitere Informationen finden Sie unter mysql.rds_kill.

Wenn Sie eine Sitzung beenden, löst die Aktion möglicherweise einen langen Rollback aus.

Befolgen Sie die Richtlinien für die Optimierung von Abfragen

Beachten Sie die folgenden Richtlinien, um Abfragen zu optimieren:

- Ausführen der `EXPLAIN`-Anweisung.

Dieser Befehl zeigt die einzelnen Schritte an, die beim Ausführen einer Abfrage beteiligt sind. Weitere Informationen finden Sie unter Optimierung von Abfragen mit EXPLAIN in der MySQL-Dokumentation.

- Ausführen der `SHOW PROFILE`-Anweisung.

  Verwenden Sie diese Anweisung, um Profildetails zu überprüfen, die die Ressourcennutzung für Anweisungen angeben können, die während der aktuellen Sitzung ausgeführt werden. Weitere Informationen finden Sie unter SHOW PROFILE-Anweisung in der MySQL-Dokumentation.

- Ausführen der `ANALYZE TABLE`-Anweisung.

  Verwenden Sie diese Anweisung, um die Indexstatistiken für die Tabellen zu aktualisieren, auf die die Abfrage mit hohem CPU-Verbrauch zugegriffen wird. Durch die Analyse der Anweisung können Sie dem Optimierer bei der Auswahl eines geeigneten Ausführungsplans helfen. Weitere Informationen finden Sie unter ANALYZE TABLE in der MySQL-Dokumentation.

Befolgen Sie die Richtlinien zur Verbesserung der CPU-Auslastung

Um die CPU-Auslastung in einer Datenbank-Instance zu verbessern, befolgen Sie die folgenden Richtlinien:

- Stellen Sie sicher, dass alle Abfragen die richtigen Indizes verwenden.
- Finden Sie heraus, ob Sie parallele Aurora-Abfragen verwenden können. Sie können diese Technik verwenden, um die CPU-Auslastung auf dem Hauptknoten zu reduzieren, indem Sie die Funktionsverarbeitung, die Zeilenfilterung und die Spaltenprojektion für die `WHERE`-Klausel herabstufen.
- Finden Sie heraus, ob die Anzahl der SQL-Ausführungen pro Sekunde die erwarteten Schwellenwerte erfüllt.
- Finden Sie heraus, ob Indexwartung oder neue Indexerstellung CPU-Zyklen in Anspruch nehmen, die von Ihrer Produktions-Workload benötigt werden Planen Sie Wartungsaktivitäten außerhalb der Spitzenzeiten.
- Finden Sie heraus, ob Sie die Partitionierung verwenden können, um den Abfragedatensatz zu reduzieren. Weitere Informationen finden Sie im Blogbeitrag So planen und optimieren Sie Amazon Aurora mit MySQL-Kompatibilität für konsolidierte Workloads.

Auf Verbindungsströme überprüfen

Ist die `DBLoadCPU`-Metrik nicht sehr hoch, aber die `CPUUtilization`-Metrik hoch, liegt die Ursache für die hohe CPU-Auslastung außerhalb der Datenbank-Engine. Ein klassisches Beispiel ist ein Verbindungssturm.

Prüfen Sie, ob die folgenden Bedingungen zutreffen:

- Sowohl die Performance-Insights-`CPUUtilization`-Metrik als auch die Amazon-CloudWatch-`DatabaseConnections`-Metrik sind gestiegen.
- Die Anzahl der Threads in der CPU ist größer als die Anzahl der vCPUs.

Wenn die vorhergehenden Bedingungen zutreffen, sollten Sie erwägen, die Anzahl der Datenbankverbindungen zu verringern. Sie können beispielsweise einen Verbindungspool wie RDS-Proxy verwenden. Informationen zu den bewährten Methoden für eine effektive Verbindungsverwaltung und -skalierung finden Sie im Whitepaper Handbuch von Amazon Aurora MySQL DBA für Verbindungsverwaltung.

io/aurora_redo_log_flush

Dieses `io/aurora_redo_log_flush`-Ereignis tritt ein, wenn eine Sitzung persistente Daten in den Amazon-Aurora-Speicher schreibt.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 2

Kontext

Das `io/aurora_redo_log_flush`-Ereignis ist für eine Schreiboperation mit Ein-/Ausgabe (I/O) in Aurora MySQL vorgesehen.

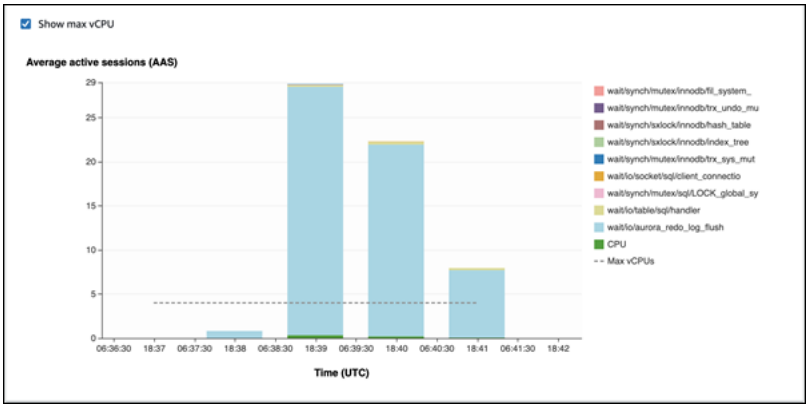
> **Note**
>
> In Version 3 von Aurora MySQL heißt dieses Warteereignis io/redo_log_flush.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

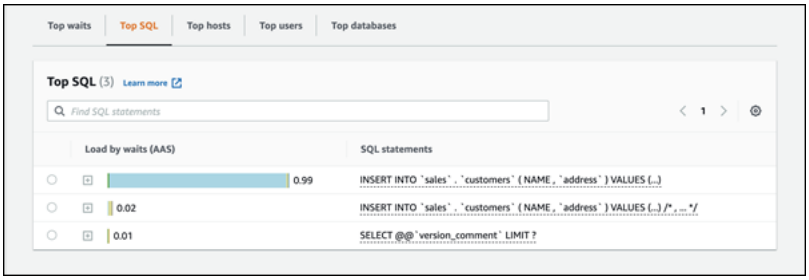
Aus Gründen der Datenpersistenz erfordern Commits ein dauerhaftes Schreiben in einen stabilen Speicher. Wenn die Datenbank zu viele Commits ausführt, gibt es ein Warteereignis für die I/O-Schreiboperation, das `io/aurora_redo_log_flush`-Warteereignis.

In den folgenden Beispielen werden 50 000 Datensätze mithilfe der db.r5.xlarge DB-Instance-Klasse in einen Aurora-MySQL-DB-Cluster eingefügt:

- Im ersten Beispiel fügt jede Sitzung zeilenweise 10 000 Datensätze ein. Wenn sich ein DML-Befehl (Data Manipulation Language) nicht innerhalb einer Transaktion befindet, verwendet Aurora MySQL standardmäßig implizite Commits. Autocommit ist aktiviert. Dies bedeutet, dass für jede Zeileneinfügung ein Commit vorhanden ist. Performance Insights zeigt, dass die Verbindungen die meiste Zeit damit verbringen, auf das `io/aurora_redo_log_flush`-Warteereignis zu warten.

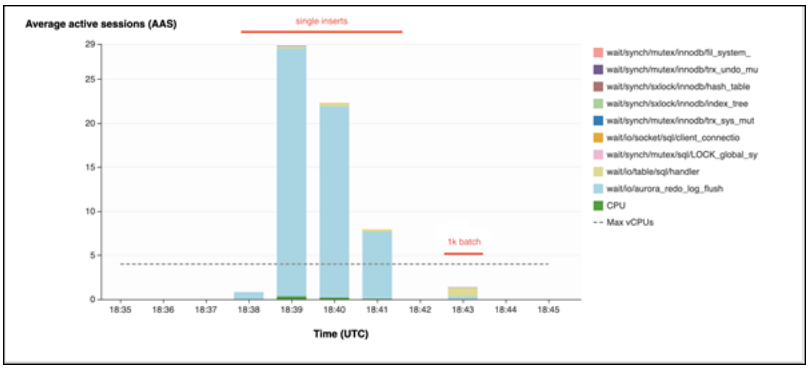


  Dies wird durch die verwendeten einfachen Einfügungsanweisungen verursacht.



Die 50 000 Datensätze benötigen 3,5 Minuten, um eingefügt zu werden.

- Im zweiten Beispiel werden Einfügungen in 1 000 Batches vorgenommen, d.h. jede Verbindung führt 10 Commits statt 10 000 aus. Performance Insights zeigt, dass die Verbindungen nicht die meiste Zeit mit dem `io/aurora_redo_log_flush`-Warteereignis verbringen.



Die 50 000 Datensätze benötigen 4 Sekunden, um eingefügt zu werden.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Identifizieren Sie die problematischen Sitzungen und Abfragen
- Gruppieren Sie Ihre Schreiboperationen
- Deaktivieren Sie Autocommit
- Transaktionen verwenden
- Verwenden von Batches

Identifizieren Sie die problematischen Sitzungen und Abfragen

Wenn Ihre DB-Instance einen Engpass hat, besteht Ihre erste Aufgabe darin, die Sitzungen und Abfragen zu finden, die ihn verursachen. Einen nützlichen AWS-Datenbank-Blogbeitrag finden Sie unter Analysieren von Amazon-Aurora-MySQL-Workloads mit Performance Insights.

So identifizieren Sie Sitzungen und Abfragen, die einen Engpass verursachen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie Ihre DB-Instance aus.
4. Wählen Sie unter Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Die Abfragen oben in der Liste verursachen die höchste Belastung der Datenbank.

Gruppieren Sie Ihre Schreiboperationen

Die folgenden Beispiele lösen das `io/aurora_redo_log_flush`-Warteereignis aus. (Autocommit ist aktiviert.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

Um die Wartezeit auf das `io/aurora_redo_log_flush`-Wartezeitereignis zu verkürzen, gruppieren Sie Ihre Schreibvorgänge logisch in einem einzigen Commit, um persistente Speicheraufrufe zu reduzieren.

Deaktivieren Sie Autocommit

Deaktivieren Sie Autocommit, bevor Sie große Änderungen vornehmen, die nicht in einer Transaktion enthalten sind, wie im folgenden Beispiel gezeigt.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

Transaktionen verwenden

Sie können Transaktionen verwenden, wie im folgenden Beispiel gezeigt.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

Verwenden von Batches

Sie können Änderungen in Batches vornehmen, wie im folgenden Beispiel gezeigt. Die Verwendung von zu großen Batches kann jedoch Leistungsprobleme verursachen, insbesondere bei Lesereplikaten oder bei zeitpunktbezogener Wiederherstellung (PITR).

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/aurora_respond_to_client

Das `io/aurora_respond_to_client`-Ereignis tritt auf, wenn ein Thread darauf wartet, eine Ergebnismenge an einen Client zurückzugeben.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 2

Kontext

Das Ereignis `io/aurora_respond_to_client` zeigt an, dass ein Thread darauf wartet, eine Ergebnismenge an einen Client zurückzugeben.

Die Abfrageverarbeitung ist abgeschlossen und die Ergebnisse werden an den Anwendungsclient zurückgegeben. Da jedoch nicht genügend Netzwerkbandbreite im DB-Cluster vorhanden ist, wartet ein Thread darauf, die Ergebnismenge zurückzugeben.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `io/aurora_respond_to_client`-Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

DB-Instance-Klasse reicht für die Workload nicht aus

Die vom DB-Cluster verwendete DB-Instance-Klasse verfügt nicht über die erforderliche Netzwerkbandbreite, um die Workload effizient zu verarbeiten.

Große Ergebnismengen

Es gab eine Zunahme der Größe der zurückgegebenen Ergebnismenge, da die Abfrage eine höhere Anzahl von Zeilen zurückgibt. Die größere Ergebnismenge verbraucht mehr Netzwerkbandbreite.

Erhöhte Belastung des Clients

Auf dem Client kann es zu CPU-, Speicher- oder Netzwerksättigung kommen. Eine Erhöhung der Belastung des Clients verzögert den Empfang von Daten aus dem Aurora-MySQL-DB-Cluster.

Erhöhte Netzwerklatenz

Es kann zu einer erhöhten Netzwerklatenz zwischen dem Aurora-MySQL-DB-Cluster und dem Client kommen. Eine höhere Netzwerklatenz erhöht die Zeit, die der Client benötigt, um die Daten zu empfangen.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen
- Skalieren Sie die DB-Instance-Klasse
- Überprüfen Sie die Workload auf unerwartete Ergebnisse
- Verteilen Sie die Workload mit Leser-Instances
- Verwenden Sie den Modifikator SQL_BUFFER_RESULT

Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen

Sie können Performance Insights verwenden, um Abfragen anzuzeigen, die durch das `io/aurora_respond_to_client`-Warteereignis gesperrt wurden Normalerweise weisen Datenbanken mit mäßiger bis erheblicher Last Warteereignisse auf. Die Warteereignisse sind möglicherweise akzeptabel, wenn die Leistung optimal ist. Wenn die Leistung nicht optimal ist, untersuchen Sie, wo die Datenbank die meiste Zeit verbringt. Schauen Sie sich die Warteereignisse an, die zur höchsten Belastung beitragen und finden Sie heraus, ob Sie die Datenbank und die Anwendung optimieren können, um diese Ereignisse zu reduzieren.

So suchen Sie SQL-Abfragen, die für hohe Last verantwortlich sind

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus. Das Performance-Insights-Dashboard wird für diese DB-Instance angezeigt.
4. Wählen Sie im Diagramm zur Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Das Diagramm listet die SQL-Abfragen auf, die für die Belastung verantwortlich sind. Diejenigen, die an der Spitze der Liste stehen, sind am meisten verantwortlich. Konzentrieren Sie sich auf diese Aussagen, um einen Engpass zu beheben.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im AWS-Database Blog-Beitrag Analysieren von Amazon-Aurora-MySQL-Workloads mit Performance Insights.

Skalieren Sie die DB-Instance-Klasse

Prüfen Sie, ob sich der Wert der Amazon-CloudWatch-Metriken in Bezug auf den Netzwerkdurchsatz erhöht, z. B. `NetworkReceiveThroughput` und `NetworkTransmitThroughput`. Wenn die Netzwerkbandbreite der DB-Instance-Klasse erreicht wird, können Sie die vom DB-Cluster verwendete DB-Instance-Klasse skalieren, indem Sie den DB-Cluster ändern. Eine DB-Instance-Klasse mit größerer Netzwerkbandbreite gibt Daten effizienter an Clients zurück.

Weitere Informationen zum Überwachen von Amazon-CloudWatch-Metriken finden Sie unter Anzeigen von Metriken in der Amazon-RDS-Konsole. Weitere Informationen zu DB-Instance-Klassen

finden Sie unter DB-Instance-Klassen von Amazon Aurora. Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Überprüfen Sie die Workload auf unerwartete Ergebnisse

Überprüfen Sie die Workload im DB-Cluster und stellen Sie sicher, dass keine unerwarteten Ergebnisse erzielt werden. Beispielsweise kann es Abfragen geben, die eine höhere Anzahl von Zeilen als erwartet zurückgeben. In diesem Fall können Sie Performance-Insights-Zählermetriken wie `Innodb_rows_read` verwenden. Weitere Informationen finden Sie unter Performance-Insights-Zählermetriken.

Verteilen Sie die Workload mit Leser-Instances

Sie können schreibgeschützte Workloads mit Aurora-Replikaten verteilen. Sie können horizontal skalieren, indem Sie weitere Aurora-Replikate hinzufügen. Dies kann zu einer Erhöhung der Drosselgrenzen für die Netzwerkbandbreite führen. Weitere Informationen finden Sie unter Amazon-Aurora-DB-Cluster.

Verwenden Sie den Modifikator SQL_BUFFER_RESULT

Sie können `SELECT`-Anweisungen den Modifikator `SQL_BUFFER_RESULT` hinzufügen, um das Ergebnis in eine temporäre Tabelle zu zwingen, bevor es an den Client zurückgegeben wird. Dieser Modifikator kann bei Leistungsproblemen helfen, wenn InnoDB-Sperren nicht freigegeben werden, weil sich Abfragen im Wartezustand `io/aurora_respond_to_client` befinden. Weitere Informationen finden Sie unter SELECT-Anweisung in der MySQL-Dokumentation.

io/redo_log_flush

Dieses `io/redo_log_flush`-Ereignis tritt ein, wenn eine Sitzung persistente Daten in den Amazon-Aurora-Speicher schreibt.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 3

Kontext

Das `io/redo_log_flush`-Ereignis ist für eine Schreiboperation mit Ein-/Ausgabe (I/O) in Aurora MySQL vorgesehen.

> **Note**
>
> In Aurora MySQL Version 2 heißt dieses Warteereignis io/aurora_redo_log_flush.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Aus Gründen der Datenpersistenz erfordern Commits ein dauerhaftes Schreiben in einen stabilen Speicher. Wenn die Datenbank zu viele Commits ausführt, gibt es ein Warteereignis für die I/O-Schreiboperation, das `io/redo_log_flush`-Warteereignis.

Beispiele für das Verhalten dieses Warteereignisses finden Sie unter io/aurora_redo_log_flush.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Identifizieren Sie die problematischen Sitzungen und Abfragen
- Gruppieren Sie Ihre Schreiboperationen
- Deaktivieren Sie Autocommit
- Transaktionen verwenden
- Verwenden von Batches

Identifizieren Sie die problematischen Sitzungen und Abfragen

Wenn Ihre DB-Instance einen Engpass hat, besteht Ihre erste Aufgabe darin, die Sitzungen und Abfragen zu finden, die ihn verursachen. Einen nützlichen AWS-Datenbank-Blogbeitrag finden Sie unter Analysieren von Amazon-Aurora-MySQL-Workloads mit Performance Insights.

So identifizieren Sie Sitzungen und Abfragen, die einen Engpass verursachen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie Ihre DB-Instance aus.
4. Wählen Sie unter Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Die Abfragen oben in der Liste verursachen die höchste Belastung der Datenbank.

Gruppieren Sie Ihre Schreiboperationen

Die folgenden Beispiele lösen das `io/redo_log_flush`-Warteereignis aus. (Autocommit ist aktiviert.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

Um die Wartezeit auf das `io/redo_log_flush`-Wartezeitereignis zu verkürzen, gruppieren Sie Ihre Schreibvorgänge logisch in einem einzigen Commit, um persistente Speicheraufrufe zu reduzieren.

Deaktivieren Sie Autocommit

Deaktivieren Sie Autocommit, bevor Sie große Änderungen vornehmen, die nicht in einer Transaktion enthalten sind, wie im folgenden Beispiel gezeigt.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

Transaktionen verwenden

Sie können Transaktionen verwenden, wie im folgenden Beispiel gezeigt.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

Verwenden von Batches

Sie können Änderungen in Batches vornehmen, wie im folgenden Beispiel gezeigt. Die Verwendung von zu großen Batches kann jedoch Leistungsprobleme verursachen, insbesondere bei Lesereplikaten oder bei zeitpunktbezogener Wiederherstellung (PITR).

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/socket/sql/client_connection

Dieses `io/socket/sql/client_connection`-Ereignis tritt auf, wenn ein Thread gerade eine neue Verbindung verarbeitet.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Versionen 2 und 3

Kontext

Das `io/socket/sql/client_connection`-Ereignis zeigt an, dass mysqld damit beschäftigt ist, Threads zu erstellen, um eingehende neue Clientverbindungen zu verarbeiten. In diesem Szenario verlangsamt sich die Verarbeitung neuer Clientverbindungsanfragen, während Verbindungen warten, bis der Thread zugewiesen wird. Weitere Informationen finden Sie unter MySQL-Server (mysqld).

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

- Es gibt einen plötzlichen Anstieg neuer Benutzerverbindungen von der Anwendung zu Ihrer Amazon-RDS-Instance.

- Ihre DB-Instance kann keine neuen Verbindungen verarbeiten, da das Netzwerk, die CPU oder der Speicher gedrosselt werden.

Aktionen

Wenn `io/socket/sql/client_connection` die Datenbankaktivität dominiert, weist dies nicht unbedingt auf ein Leistungsproblem hin. In einer Datenbank, die nicht im Leerlauf ist, ist ein Warteereignis immer im Vordergrund. Handeln Sie nur, wenn die Leistung nachlässt. Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir unterschiedliche Maßnahmen.

Themen

- Identifizieren Sie die problematischen Sitzungen und Abfragen
- Befolgen Sie die bewährten Methoden für die Verbindungsverwaltung
- Vergrößern Sie Ihre Instance, wenn Ressourcen gedrosselt werden
- Prüfen Sie die Top-Hosts und Top-Benutzer
- Fragen Sie die performance_schema-Tabellen ab
- Prüfen Sie die Thread-Zustand Ihrer Abfragen
- Prüfen Sie Ihre Anfragen und Abfragen
- Kombinieren Sie Ihre Datenbankverbindungen

Identifizieren Sie die problematischen Sitzungen und Abfragen

Wenn Ihre DB-Instance einen Engpass hat, besteht Ihre erste Aufgabe darin, die Sitzungen und Abfragen zu finden, die ihn verursachen. Einen nützlichen Blogbeitrag finden Sie unter Analysieren von Amazon-Aurora-MySQL-Workloads mit Performance Insights.

So identifizieren Sie Sitzungen und Abfragen, die einen Engpass verursachen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie Ihre DB-Instance aus.
4. Wählen Sie unter Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Die Abfragen oben in der Liste verursachen die höchste Belastung der Datenbank.

Befolgen Sie die bewährten Methoden für die Verbindungsverwaltung

Berücksichtigen Sie die folgenden Strategien, um Ihre Verbindungen zu verwalten:

- Verwenden Sie Verbindungspooling.

  Sie können die Anzahl der Verbindungen nach Bedarf schrittweise erhöhen. Weitere Informationen finden Sie im Whitepaper Amazon-Aurora-MySQL-Datenbankadministratorhandbuch.
- Verwenden Sie einen Reader-Knoten, um schreibgeschützten Datenverkehr umzuverteilen.

  Weitere Informationen erhalten Sie unter Aurora-Replikate und Amazon-Aurora-Endpunktverbindungen.

Vergrößern Sie Ihre Instance, wenn Ressourcen gedrosselt werden

Suchen Sie nach Beispielen für die Drosselung in den folgenden Ressourcen:

- CPU

  Überprüfen Sie Ihre Amazon-CloudWatch-Metriken auf hohe CPU-Auslastung.
- Netzwerk

  Prüfen Sie, ob der Wert der CloudWatch-Metriken `network receive throughput` und `network transmit throughput` ansteigt. Wenn Ihre Instance das Netzwerkbandbreitenlimit für Ihre Instance-Klasse erreicht hat, sollten Sie Ihre RDS-Instance auf einen höheren Instance-Klassentyp skalieren. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
- Freisetzbarer Speicher

  Suchen Sie nach einem Abfall der CloudWatch-Metrik `FreeableMemory`. Ziehen Sie auch in Erwägung, die erweiterte Überwachung zu aktivieren. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).

Prüfen Sie die Top-Hosts und Top-Benutzer

Verwenden Sie Performance Insights, um die Top-Hosts und Top-Benutzer zu überprüfen. Weitere Informationen finden Sie unter Analyse der Metriken mit dem Performance-Insights-Dashboard.

Fragen Sie die performance_schema-Tabellen ab

Um eine genaue Anzahl der aktuellen und gesamten Verbindungen zu erhalten, fragen Sie die `performance_schema`-Tabellen ab. Mit dieser Technik identifizieren Sie den Quellbenutzer oder Host, der für das Erstellen einer hohen Anzahl von Verbindungen verantwortlich ist. Fragen Sie die `performance_schema`-Tabellen beispielsweise wie folgt ab.

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

Prüfen Sie die Thread-Zustand Ihrer Abfragen

Wenn Ihr Leistungsproblem andauert, überprüfen Sie die Thread-Zustand Ihrer Abfragen. Geben Sie im `mysql`-Client den folgenden Befehl aus.

```
show processlist;
```

Prüfen Sie Ihre Anfragen und Abfragen

Um die Art der Anfragen und Abfragen von Benutzerkonten zu überprüfen, verwenden Sie AuroraAurora MySQL Advanced Auditing. Weitere Informationen zum Aktivieren der Prüfung finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.

Kombinieren Sie Ihre Datenbankverbindungen

Erwägen Sie, Amazon RDS Proxy für die Verbindungsverwaltung zu verwenden. Durch die Verwendung von RDS Proxy können Sie Ihren Anwendungen erlauben, Datenbankverbindungen zu bündeln und gemeinsam zu nutzen, um ihre Skalierbarkeit zu verbessern. RDS Proxy macht Anwendungen widerstandsfähiger gegenüber Datenbankfehlern, indem er automatisch eine Verbindung zu einer Standby-DB-Instance herstellt, während Anwendungsverbindungen erhalten bleiben. Weitere Informationen finden Sie unter Amazon-RDS-Proxy für Aurora.

io/table/sql/handler

Dieses `io/table/sql/handler`-Ereignis tritt ein, wenn Arbeit an eine Speicher-Engine delegiert wurde.

Themen

- Unterstützte Engine-Versionen

- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Versionen 2 und 3

Kontext

Das `io/table`-Ereignis zeigt ein Warten auf den Zugriff auf eine Tabelle an. Dieses Ereignis tritt unabhängig davon auf, ob die Daten im Pufferpool zwischengespeichert werden oder über die Festplatte darauf zugegriffen wird. Das `io/table/sql/handler`-Ereignis weist auf eine Zunahme der Workload-Aktivität hin.

Ein Handler ist eine Routine, die auf eine bestimmte Art von Daten spezialisiert ist oder sich auf bestimmte spezielle Aufgaben konzentriert. Beispielsweise empfängt und verdaut ein Ereignis-Handler Ereignisse und Signale vom Betriebssystem oder von einer Benutzeroberfläche. Ein Speicherhandler führt Aufgaben im Zusammenhang mit dem Speicher aus. Ein Dateieingabe-Handler ist eine Funktion, die Dateieingaben empfängt und je nach Kontext spezielle Aufgaben für die Daten ausführt.

Ansichten wie `performance_schema.events_waits_current` zeigen oft `io/table/sql/handler` an, wenn das eigentliche Warten ein verschachteltes Warteereignis wie eine Sperre ist. Wenn das tatsächliche Warten nicht `io/table/sql/handler` ist, meldet Performance Insights das verschachtelte Warteereignis. Wenn Performance Insights `io/table/sql/handler` meldet, steht dies für die InnoDB-Verarbeitung der I/O-Anforderung und nicht für ein verstecktes verschachteltes Warteereignis. Weitere Informationen finden Sie unter Leistungsschema-Atom- und Molekülereignisse im MySQL-Referenzhandbuch.

Das `io/table/sql/handler`-Ereignis erscheint oft in primären Warteereignissen mit I/O-Wartezeiten wie `io/aurora_redo_log_flush`.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

In Performance Insights weisen plötzliche Spitzen im `io/table/sql/handler`-Ereignis auf eine Zunahme der Workload-Aktivität hin. Erhöhte Aktivität bedeutet eine erhöhte I/O.

Performance Insights filtert die Verschachtelungsereignis-IDs und meldet keine `io/table/sql/handler`-Wartezeit, wenn das zugrunde liegende verschachtelte Ereignis eine Sperrwartezeit ist. Wenn beispielsweise das Grundursachenereignis synch/mutex/innodb/aurora_lock_thread_slot_futex ist, zeigt Performance Insights diese Wartezeit in den Top-Warteereignissen an und nicht `io/table/sql/handler`.

In Ansichten wie `performance_schema.events_waits_current` werden Wartezeiten für `io/table/sql/handler` oft angezeigt, wenn das eigentliche Warten ein verschachteltes Warteereignis wie eine Sperre ist. Wenn die tatsächliche Wartezeit von `io/table/sql/handler` abweicht, schlägt Performance Insights die verschachtelte Wartezeit nach und meldet die tatsächliche Wartezeit anstelle von `io/table/sql/handler`. Wenn Performance Insights `io/table/sql/handler` meldet, ist die tatsächliche Wartezeit `io/table/sql/handler` und kein verstecktes verschachteltes Warteereignis. Weitere Informationen finden Sie unter Leistungsschema-Atom- und Molekülereignisse im MySQL-Referenzhandbuch 5.7.

Aktionen

Wenn das Warteereignis die Datenbankaktivität dominiert, weist dies nicht unbedingt auf ein Leistungsproblem hin. Ein Warteereignis ist immer im Vordergrund, wenn die Datenbank aktiv ist. Sie müssen nur handeln, wenn sich die Leistung verschlechtert.

Abhängig von den anderen angezeigten Warteereignissen empfehlen wir unterschiedliche Aktionen.

Themen

- Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen
- Überprüfen Sie, ob eine Korrelation mit den Performance-Insights-Zählermetriken vorliegt
- Nach anderen korrelierten Warteereignissen suchen

Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen

Normalerweise weisen Datenbanken mit mäßiger bis erheblicher Last Warteereignisse auf. Die Warteereignisse sind möglicherweise akzeptabel, wenn die Leistung optimal ist. Wenn die Leistung nicht optimal ist, untersuchen Sie, wo die Datenbank die meiste Zeit verbringt. Schauen Sie sich die Warteereignisse an, die zur höchsten Belastung beitragen und finden Sie heraus, ob Sie die Datenbank und die Anwendung optimieren können, um diese Ereignisse zu reduzieren.

So suchen Sie SQL-Abfragen, die für hohe Last verantwortlich sind

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus. Das Performance-Insights-Dashboard wird für diese DB-Instance angezeigt.
4. Wählen Sie im Diagramm zur Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Das Diagramm listet die SQL-Abfragen auf, die für die Belastung verantwortlich sind. Diejenigen, die an der Spitze der Liste stehen, sind am meisten verantwortlich. Konzentrieren Sie sich auf diese Aussagen, um einen Engpass zu beheben.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights.

Überprüfen Sie, ob eine Korrelation mit den Performance-Insights-Zählermetriken vorliegt

Suchen Sie nach Performance-Insights-Zählermetriken wie `Innodb_rows_changed`. Wenn Zählermetriken mit `io/table/sql/handler` korreliert sind, gehen Sie folgendermaßen vor:

1. Suchen Sie in Performance Insights nach den SQL-Anweisungen, die das `io/table/sql/handler`-Top-Warteereignis berücksichtigen. Optimieren Sie diese Anweisung, wenn möglich, damit sie weniger Zeilen zurückgibt.
2. Rufen Sie die obersten Tabellen aus den `schema_table_statistics`- und x `$schema_table_statistics`-Ansichten ab. Diese Ansichten zeigen den Zeitaufwand pro Tabelle an. Weitere Informationen finden Sie unter Die Ansichten schema_table_statistics und x $schema_table_statistics im MySQL-Referenzhandbuch.

   Standardmäßig werden Zeilen nach absteigender Gesamtwartezeit sortiert. Tabellen mit dem größten Streit erscheinen zuerst. Die Ausgabe gibt an, ob Zeit für Lese-, Schreibvorgänge, Abrufe, Einfügungen, Aktualisierungen oder Löschungen aufgewendet wird.

```
mysql> select * from sys.schema_table_statistics limit 1\G

*************************** 1. row ***************************
     table_schema: read_only_db
```

```
       table_name: sbtest41
    total_latency: 54.11 m
     rows_fetched: 6001557
    fetch_latency: 39.14 m
    rows_inserted: 14833
   insert_latency: 5.78 m
     rows_updated: 30470
   update_latency: 5.39 m
     rows_deleted: 14833
   delete_latency: 3.81 m
 io_read_requests: NULL
          io_read: NULL
  io_read_latency: NULL
io_write_requests: NULL
         io_write: NULL
 io_write_latency: NULL
 io_misc_requests: NULL
  io_misc_latency: NULL
1 row in set (0.11 sec)
```

Nach anderen korrelierten Warteereignissen suchen

Wenn `synch/sxlock/innodb/btr_search_latch` und `io/table/sql/handler` zusammen am meisten zur DB-Ladeanomalie beitragen, prüfen Sie, ob die Variable `innodb_adaptive_hash_index` aktiviert ist. Wenn dies der Fall ist, sollten Sie den `innodb_adaptive_hash_index_parts`-Parameterwert erhöhen.

Wenn der Adaptive-Hash-Index ausgeschaltet, erwägen Sie, ihn einzuschalten. Weitere Informationen zum MySQL-Adaptive-Hash-Index finden Sie in den folgenden Ressourcen:

- Der Artikel Ist der Adaptive-Hash-Index in InnoDB für meinen Workload geeignet? auf der Percona-Website
- Adaptive-Hash-Index im MySQL-Referenzhandbuch
- Der Artikel Verbindung in MySQL InnoDB: Nützliche Informationen aus der Semaphores-Sektion auf der Percona-Website

> **Note**
>
> Der Adaptive-Hash-Index wird auf Aurora-Reader-DB-Instances nicht unterstützt.

In einigen Fällen kann die Leistung auf einer Reader-Instance schwach sein, wenn `synch/sxlock/innodb/btr_search_latch` und `io/table/sql/handler` dominant sind. Wenn ja, erwägen Sie, die Workload vorübergehend auf die Writer-DB-Instance umzuleiten und den Adaptive-Hash-Index zu aktivieren.

synch/cond/innodb/row_lock_wait

Das `synch/cond/innodb/row_lock_wait`-Ereignis tritt auf, wenn eine Sitzung eine Zeile für ein Update gesperrt hat und eine andere Sitzung versucht, dieselbe Zeile zu aktualisieren. Weitere Informationen finden Sie unter InnoDB Locking in der MySQL-Dokumentation.

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 3

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Mehrere DML-Anweisungen (Data Manipulation Language) greifen gleichzeitig auf dieselbe Zeile oder dieselben Zeilen zu.

Aktionen

Abhängig von den anderen angezeigten Warteereignissen empfehlen wir unterschiedliche Aktionen.

Themen

- Finden und antworten Sie auf die SQL-Anweisungen, die für dieses Warteereignis verantwortlich sind
- Suchen und antworten Sie auf die Blockiersitzung

Finden und antworten Sie auf die SQL-Anweisungen, die für dieses Warteereignis verantwortlich sind

Verwenden Sie Performance Insights, um die SQL-Anweisungen zu identifizieren, die für dieses Warteereignis verantwortlich sind. Berücksichtigen Sie dabei die folgenden Strategien:

- Wenn Zeilensperren ein dauerhaftes Problem darstellen, erwägen Sie, die Anwendung neu zu schreiben, um eine optimistische Sperre zu verwenden.

- Verwenden Sie mehrzeilige Anweisungen.
- Verteilen Sie die Workload auf verschiedene Datenbankobjekte. Sie können dazu die Partitionierung verwenden.
- Prüfen Sie den Wert des `innodb_lock_wait_timeout`-Parameters. Es steuert, wie lange Transaktionen warten, bevor ein Timeout-Fehler generiert wird.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights.

Suchen und antworten Sie auf die Blockiersitzung

Stellen Sie fest, ob die Blockiersitzung im Leerlauf oder aktiv ist. Finden Sie außerdem heraus, ob die Sitzung von einer Anwendung oder einem aktiven Benutzer stammt.

Um die Sitzung zu identifizieren, die die Sperre hält, können Sie `SHOW ENGINE INNODB STATUS` ausführen. Das folgende Beispiel zeigt eine Beispielausgabe.

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 1688153, ACTIVE 82 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 4244, OS thread handle 70369524330224, query id 4020834 172.31.14.179
 reinvent executing
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 24 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 4 n bits 72 index GEN_CLUST_INDEX of table test.t1 trx
 id 1688153 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

Oder Sie können die folgende Abfrage verwenden, um Details zu aktuellen Sperren zu extrahieren.

```
mysql> SELECT p1.id waiting_thread,
    p1.user waiting_user,
    p1.host waiting_host,
    it1.trx_query waiting_query,
    ilw.requesting_engine_transaction_id waiting_transaction,
    ilw.blocking_engine_lock_id blocking_lock,
    il.lock_mode blocking_mode,
    il.lock_type blocking_type,
```

```
    ilw.blocking_engine_transaction_id blocking_transaction,
    CASE it.trx_state
        WHEN 'LOCK WAIT'
        THEN it.trx_state
        ELSE p.state end blocker_state,
    concat(il.object_schema,'.', il.object_name) as locked_table,
    it.trx_mysql_thread_id blocker_thread,
    p.user blocker_user,
    p.host blocker_host
FROM performance_schema.data_lock_waits ilw
JOIN performance_schema.data_locks il
ON ilw.blocking_engine_lock_id = il.engine_lock_id
AND ilw.blocking_engine_transaction_id = il.engine_transaction_id
JOIN information_schema.innodb_trx it
ON ilw.blocking_engine_transaction_id = it.trx_id join information_schema.processlist p
ON it.trx_mysql_thread_id = p.id join information_schema.innodb_trx it1
ON ilw.requesting_engine_transaction_id = it1.trx_id join
 information_schema.processlist p1
ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
waiting_thread: 4244
waiting_user: reinvent
waiting_host: 123.456.789.012:18158
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 1688153
blocking_lock: 70369562074216:11:4:2:70369549808672
blocking_mode: X
blocking_type: RECORD
blocking_transaction: 1688142
blocker_state: User sleep
locked_table: test.t1
blocker_thread: 4243
blocker_user: reinvent
blocker_host: 123.456.789.012:18156
1 row in set (0.00 sec)
```

Wenn Sie die Sitzung identifizieren, umfassen Ihre Optionen die Folgenden:

- Wenden Sie sich an den Besitzer der Anwendung oder den Benutzer.
- Wenn die Sperrsitzung im Leerlauf ist, erwägen Sie, die Sperrsitzung zu beenden. Diese Aktion könnte einen langen Rollback auslösen. Informationen zum Beenden einer Sitzung finden Sie unter Beenden einer Sitzung oder Abfrage.

Weitere Informationen zum Identifizieren blockierender Transaktionen finden Sie unter Verwenden von InnoDB-Transaktions- und Sperrinformationen in der MySQL-Dokumentation.

synch/cond/innodb/row_lock_wait_cond

Das `synch/cond/innodb/row_lock_wait_cond`-Ereignis tritt auf, wenn eine Sitzung eine Zeile für ein Update gesperrt hat und eine andere Sitzung versucht, dieselbe Zeile zu aktualisieren. Weitere Informationen finden Sie unter InnoDB Locking in der MySQL-Dokumentation.

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 2

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Mehrere DML-Anweisungen (Data Manipulation Language) greifen gleichzeitig auf dieselbe Zeile oder dieselben Zeilen zu.

Aktionen

Abhängig von den anderen angezeigten Warteereignissen empfehlen wir unterschiedliche Aktionen.

Themen

- Finden und antworten Sie auf die SQL-Anweisungen, die für dieses Warteereignis verantwortlich sind
- Suchen und antworten Sie auf die Blockiersitzung

Finden und antworten Sie auf die SQL-Anweisungen, die für dieses Warteereignis verantwortlich sind

Verwenden Sie Performance Insights, um die SQL-Anweisungen zu identifizieren, die für dieses Warteereignis verantwortlich sind. Berücksichtigen Sie dabei die folgenden Strategien:

- Wenn Zeilensperren ein dauerhaftes Problem darstellen, erwägen Sie, die Anwendung neu zu schreiben, um eine optimistische Sperre zu verwenden.
- Verwenden Sie mehrzeilige Anweisungen.
- Verteilen Sie die Workload auf verschiedene Datenbankobjekte. Sie können dazu die Partitionierung verwenden.

- Prüfen Sie den Wert des `innodb_lock_wait_timeout`-Parameters. Es steuert, wie lange Transaktionen warten, bevor ein Timeout-Fehler generiert wird.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights.

Suchen und antworten Sie auf die Blockiersitzung

Stellen Sie fest, ob die Blockiersitzung im Leerlauf oder aktiv ist. Finden Sie außerdem heraus, ob die Sitzung von einer Anwendung oder einem aktiven Benutzer stammt.

Um die Sitzung zu identifizieren, die die Sperre hält, können Sie `SHOW ENGINE INNODB STATUS` ausführen. Das folgende Beispiel zeigt eine Beispielausgabe.

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 2771110, ACTIVE 112 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 24, OS thread handle 70369573642160, query id 13271336 172.31.14.179
 reinvent Sending data
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 43 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 3 n bits 0 index GEN_CLUST_INDEX of table test.t1 trx
 id 2771110 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

Oder Sie können die folgende Abfrage verwenden, um Details zu aktuellen Sperren zu extrahieren.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
```

```
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

Wenn Sie die Sitzung identifizieren, umfassen Ihre Optionen die Folgenden:

- Wenden Sie sich an den Besitzer der Anwendung oder den Benutzer.
- Wenn die Sperrsitzung im Leerlauf ist, erwägen Sie, die Sperrsitzung zu beenden. Diese Aktion könnte einen langen Rollback auslösen. Informationen zum Beenden einer Sitzung finden Sie unter Beenden einer Sitzung oder Abfrage.

Weitere Informationen zum Identifizieren blockierender Transaktionen finden Sie unter Verwenden von InnoDB-Transaktions- und Sperrinformationen in der MySQL-Dokumentation.

synch/cond/sql/MDL_context::COND_wait_status

Dieses `synch/cond/sql/MDL_context::COND_wait_status`-Ereignis tritt ein, wenn Threads auf eine Tabellenmetadatensperre warten.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Versionen 2 und 3

Kontext

Das `synch/cond/sql/MDL_context::COND_wait_status`-Ereignis zeigt an, dass Threads auf eine Tabellenmetadatensperre warten. In einigen Fällen hält eine Sitzung eine Metadatensperre für eine Tabelle und eine andere Sitzung versucht, dieselbe Sperre für dieselbe Tabelle zu erhalten. In einem solchen Fall wartet die zweite Sitzung auf das `synch/cond/sql/MDL_context::COND_wait_status`-Warteereignis.

MySQL verwendet die Sperre von Metadaten, um den gleichzeitigen Zugriff auf Datenbankobjekte zu verwalten und die Datenkonsistenz sicherzustellen. Das Sperren von Metadaten gilt für Tabellen, Schemata, geplante Ereignisse, Tablespaces und Benutzersperren, die mit der `get_lock`-Funktion erworben wurden, und gespeicherte Programme. Gespeicherte Programme umfassen Prozeduren, Funktionen und Auslöser. Weitere Informationen finden Sie unter Metadata Locking in der MySQL-Dokumentation.

Die MySQL-Prozessliste zeigt diese Sitzung im Zustand `waiting for metadata lock`. Wenn in Performance Insights `Performance_schema` aktiviert ist, wird das Ereignis `synch/cond/sql/MDL_context::COND_wait_status` angezeigt.

Das Standardtimeout für eine Abfrage, die auf eine Metadatensperre wartet, basiert auf dem Wert des `lock_wait_timeout`-Parameters, der standardmäßig 31.536.000 Sekunden (365 Tage) beträgt.

Weitere Informationen zu verschiedenen InnoDB-Sperren und den Arten von Sperren, die Konflikte verursachen können, finden Sie unter InnoDB-Sperren in der MySQL-Dokumentation.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `synch/cond/sql/MDL_context::COND_wait_status`-Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

Langlebige Transaktionen

Eine oder mehrere Transaktionen ändern eine große Datenmenge und halten Tabellen für eine sehr lange Zeit gesperrt.

Leerlauf-Transaktionen

Eine oder mehrere Transaktionen bleiben lange offen, ohne festgeschrieben oder rückgängig gemacht zu werden.

DDL-Anweisungen zu großen Tabellen

Für sehr große Tabellen wurden eine oder mehrere DDL-Anweisungen (Data Definition Statements) ausgeführt, z. B. `ALTER TABLE`-Befehle.

Explizite Tabellensperren

Es gibt explizite Sperren für Tabellen, die nicht zeitnah freigegeben werden. Beispielsweise kann eine Anwendung `LOCK TABLE`-Anweisungen nicht ordnungsgemäß ausführen.

Aktionen

Wir empfehlen verschiedene Aktionen, abhängig von den Ursachen Ihres Warteereignisses und von der Version des Aurora-MySQL-DB-Clusters.

Themen

- Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen
- Nach vergangenen Ereignissen suchen
- Führen Sie Anfragen für Aurora MySQL Version 2 aus

- Antworten Sie auf die Blockiersitzung

Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen

Sie können Performance Insights verwenden, um Abfragen anzuzeigen, die durch das `synch/cond/sql/MDL_context::COND_wait_status`-Warteereignis gesperrt wurden Um jedoch die blockierende Sitzung zu identifizieren, fragen Sie Metadatentabellen von `performance_schema` und `information_schema` im DB-Cluster ab.

Normalerweise weisen Datenbanken mit mäßiger bis erheblicher Last Warteereignisse auf. Die Warteereignisse sind möglicherweise akzeptabel, wenn die Leistung optimal ist. Wenn die Leistung nicht optimal ist, untersuchen Sie, wo die Datenbank die meiste Zeit verbringt. Schauen Sie sich die Warteereignisse an, die zur höchsten Belastung beitragen und finden Sie heraus, ob Sie die Datenbank und die Anwendung optimieren können, um diese Ereignisse zu reduzieren.

So suchen Sie SQL-Abfragen, die für hohe Last verantwortlich sind

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus. Das Performance-Insights-Dashboard wird für diese DB-Instance angezeigt.
4. Wählen Sie im Diagramm zur Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Das Diagramm listet die SQL-Abfragen auf, die für die Belastung verantwortlich sind. Diejenigen, die an der Spitze der Liste stehen, sind am meisten verantwortlich. Konzentrieren Sie sich auf diese Aussagen, um einen Engpass zu beheben.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im AWS-Database Blog-Beitrag Analysieren von Amazon-Aurora-MySQL-Workloads mit Performance Insights.

Nach vergangenen Ereignissen suchen

Sie können einen Einblick in dieses Warteereignis erhalten, um nach früheren Vorkommen zu suchen. Führen Sie dazu die folgenden Aktionen aus:

- Überprüfen Sie die Datenmanipulationssprache (DML) und den DDL-Durchsatz und die Latenz, um festzustellen, ob sich die Workload geändert hat.

  Sie können Performance Insights verwenden, um Abfragen zu finden, die zum Zeitpunkt des Problems auf dieses Ereignis warten. Sie können auch den Digest der Abfragen anzeigen, die nahe dem Zeitpunkt des Problems ausgeführt werden.
- Wenn Überwachungsprotokolle oder allgemeine Protokolle für den DB-Cluster aktiviert sind, können Sie nach allen Abfragen suchen, die für die Objekte (schema.table) ausgeführt werden, die an der Wartetransaktion beteiligt sind. Sie können auch nach den Abfragen suchen, die vor der Transaktion ausgeführt wurden.

Die Informationen, die zur Fehlerbehebung vergangener Ereignisse zur Verfügung stehen, sind begrenzt. Durch das Ausführen dieser Prüfungen wird nicht angezeigt, welches Objekt auf Informationen wartet. Sie können jedoch Tabellen mit hoher Last zum Zeitpunkt des Ereignisses und die Menge häufig operierter Zeilen identifizieren, die zum Zeitpunkt des Problems Konflikte verursachen. Sie können diese Informationen dann verwenden, um das Problem in einer Testumgebung zu reproduzieren und Einblicke in seine Ursache zu geben.

Führen Sie Anfragen für Aurora MySQL Version 2 aus

In Aurora MySQL Version 2 können Sie die blockierte Sitzung direkt identifizieren, indem Sie `performance_schema`-Tabellen oder `sys`-Schemaansichten abfragen. Ein Beispiel kann veranschaulichen, wie Tabellen abfragen, um blockierende Abfragen und Sitzungen zu identifizieren.

In der folgenden Prozesslistenausgabe wartet die Verbindungs-ID 89 auf eine Metadatensperre und führt einen `TRUNCATE TABLE`-Befehl aus. In einer Abfrage der `performance_schema`-Tabellen oder `sys`-Schemaansichten zeigt die Ausgabe, dass die blockierende Sitzung 76 ist.

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----------+------------------+
| @@version | @@aurora_version |
+-----------+------------------+
| 5.7.12    | 2.11.5           |
+-----------+------------------+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
```

```
| Id | User            | Host                | db        | Command | Time | State
                    | Info                          |
+----+-----------------+---------------------+-----------+---------+------
+--------------------------------+------------------------------+
|  2 | rdsadmin        | localhost           | NULL      | Sleep   |    0 | NULL
                    | NULL                          |
|  4 | rdsadmin        | localhost           | NULL      | Sleep   |    2 | NULL
                    | NULL                          |
|  5 | rdsadmin        | localhost           | NULL      | Sleep   |    1 | NULL
                    | NULL                          |
| 20 | rdsadmin        | localhost           | NULL      | Sleep   |    0 | NULL
                    | NULL                          |
| 21 | rdsadmin        | localhost           | NULL      | Sleep   |  261 | NULL
                    | NULL                          |
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL      | Query   |    0 | starting
                    | show processlist              |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL      | Query   |   22 | User sleep
                    | select sleep(10000)           |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL      | Query   |    5 | Waiting for
 table metadata lock | truncate table sbtest.sbtest1 |
+----+-----------------+---------------------+-----------+---------+------
+--------------------------------+------------------------------+
18 rows in set (0.00 sec)
```

Als Nächstes zeigt eine Abfrage der `performance_schema`-Tabellen oder `sys`-Schemaansichten, dass die blockierende Sitzung 76 ist.

```
MySQL [(none)]> select * from sys.schema_table_lock_waits;

+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+--------------------------+------------------------------+
| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
           | waiting_lock_type | waiting_lock_duration | waiting_query
   | waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
 blocking_thread_id | blocking_pid | blocking_account              | blocking_lock_type
 | blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+--------------------------+------------------------------+
| sbtest        | sbtest1     |               121 |          89 |
 auroramysql5712@192.0.2.0    | EXCLUSIVE         | TRANSACTION           | truncate
 table sbtest.sbtest1 |                 10 |                            0 |
                0 |                108 |           76 | auroramysql5712@192.0.2.0    |
 SHARED_READ        | TRANSACTION            | KILL QUERY 76           | KILL 76
               |
+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-------------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+--------------------------+------------------------------+
1 row in set (0.00 sec)
```

Antworten Sie auf die Blockiersitzung

Wenn Sie die Sitzung identifizieren, umfassen Ihre Optionen die Folgenden:

- Wenden Sie sich an den Besitzer der Anwendung oder den Benutzer.

- Wenn die Sperrsitzung im Leerlauf ist, erwägen Sie, die Sperrsitzung zu beenden. Diese Aktion könnte einen langen Rollback auslösen. Informationen zum Beenden einer Sitzung finden Sie unter Beenden einer Sitzung oder Abfrage.

Weitere Informationen zum Identifizieren blockierender Transaktionen finden Sie unter Verwenden von InnoDB-Transaktions- und Sperrinformationen in der MySQL-Dokumentation.

synch/mutex/innodb/aurora_lock_thread_slot_futex

Das `synch/mutex/innodb/aurora_lock_thread_slot_futex`-Ereignis tritt auf, wenn eine Sitzung eine Zeile für ein Update gesperrt hat und eine andere Sitzung versucht, dieselbe Zeile zu aktualisieren. Weitere Informationen finden Sie unter InnoDB-Sperren in der MySQL-Referenz.

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 2

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Mehrere DML-Anweisungen (Data Manipulation Language) greifen gleichzeitig auf dieselbe Zeile oder dieselben Zeilen zu.

Aktionen

Abhängig von den anderen angezeigten Warteereignissen empfehlen wir unterschiedliche Aktionen.

Themen

- Finden und antworten Sie auf die SQL-Anweisungen, die für dieses Warteereignis verantwortlich sind
- Suchen und antworten Sie auf die Blockiersitzung

Finden und antworten Sie auf die SQL-Anweisungen, die für dieses Warteereignis verantwortlich sind

Verwenden Sie Performance Insights, um die SQL-Anweisungen zu identifizieren, die für dieses Warteereignis verantwortlich sind. Berücksichtigen Sie dabei die folgenden Strategien:

- Wenn Zeilensperren ein dauerhaftes Problem darstellen, erwägen Sie, die Anwendung neu zu schreiben, um eine optimistische Sperre zu verwenden.
- Verwenden Sie mehrzeilige Anweisungen.
- Verteilen Sie die Workload auf verschiedene Datenbankobjekte. Sie können dazu die Partitionierung verwenden.
- Prüfen Sie den Wert des `innodb_lock_wait_timeout`-Parameters. Es steuert, wie lange Transaktionen warten, bevor ein Timeout-Fehler generiert wird.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights.

Suchen und antworten Sie auf die Blockiersitzung

Stellen Sie fest, ob die Blockiersitzung im Leerlauf oder aktiv ist. Finden Sie außerdem heraus, ob die Sitzung von einer Anwendung oder einem aktiven Benutzer stammt.

Um die Sitzung zu identifizieren, die die Sperre hält, können Sie `SHOW ENGINE INNODB STATUS` ausführen. Das folgende Beispiel zeigt eine Beispielausgabe.

```
mysql> SHOW ENGINE INNODB STATUS;

----------------------TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12
 reinvent updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
 `sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

Oder Sie können die folgende Abfrage verwenden, um Details zu aktuellen Sperren zu extrahieren.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
```

```
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
         FROM information_schema.innodb_lock_waits ilw
         JOIN information_schema.innodb_locks il
           ON ilw.blocking_lock_id = il.lock_id
          AND ilw.blocking_trx_id = il.lock_trx_id
         JOIN information_schema.innodb_trx it
           ON ilw.blocking_trx_id = it.trx_id
         JOIN information_schema.processlist p
           ON it.trx_mysql_thread_id = p.id
         JOIN information_schema.innodb_trx it1
           ON ilw.requesting_trx_id = it1.trx_id
         JOIN information_schema.processlist p1
           ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

Wenn Sie die Sitzung identifizieren, umfassen Ihre Optionen die Folgenden:

- Wenden Sie sich an den Besitzer der Anwendung oder den Benutzer.
- Wenn die Sperrsitzung im Leerlauf ist, erwägen Sie, die Sperrsitzung zu beenden. Diese Aktion könnte einen langen Rollback auslösen. Informationen zum Beenden einer Sitzung finden Sie unter Beenden einer Sitzung oder Abfrage.

Weitere Informationen zum Identifizieren blockierender Transaktionen finden Sie unter Verwenden von InnoDB-Transaktions- und Sperrinformationen im MySQL-Referenzhandbuch.

synch/mutex/innodb/buf_pool_mutex

Dieses `synch/mutex/innodb/buf_pool_mutex`-Ereignis tritt ein, wenn ein Thread eine Sperre des InnoDB-Puffer-Pools für den Zugriff auf eine Seite im Speicher erworben hat.

Themen

- Relevante Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Relevante Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 2

Kontext

Der `buf_pool`-Mutex ist ein einzelner Mutex, der die Kontrolldatenstrukturen des Pufferpools schützt.

Weitere Informationen finden Sie unter Überwachen von InnoDB-Mutex-Wartezeiten mithilfe des Leistungsschemas in der MySQL-Dokumentation.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Dies ist ein Workload-spezifisches Warteereignis. Zu den häufigsten Ursachen für das Auftreten von `synch/mutex/innodb/buf_pool_mutex` unter den häufigsten Warteereignissen gehören die folgenden:

- Die Pufferpoolgröße ist nicht groß genug, um den Arbeitsdatensatz zu speichern.
- Die Workload ist für bestimmte Seiten einer bestimmten Tabelle in der Datenbank spezifischer, was zu Konflikten im Pufferpool führt.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen
- Performance Insights verwenden
- Erstellen eines Aurora-Replikats
- Überprüfen Sie die Größe des Pufferpools
- Überwachen Sie den globalen Statusverlauf

Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen

Normalerweise weisen Datenbanken mit mäßiger bis erheblicher Last Warteereignisse auf. Die Warteereignisse sind möglicherweise akzeptabel, wenn die Leistung optimal ist. Wenn die Leistung nicht optimal ist, untersuchen Sie, wo die Datenbank die meiste Zeit verbringt. Schauen Sie sich die Warteereignisse an, die zur höchsten Belastung beitragen und finden Sie heraus, ob Sie die Datenbank und die Anwendung optimieren können, um diese Ereignisse zu reduzieren.

So zeigen Sie das Top-SQL-Diagramm in der AWS-Managementkonsole an

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus. Das Performance-Insights-Dashboard wird für diese DB-Instance angezeigt.
4. Wählen Sie im Diagramm zur Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unter dem Datenbankauslastungsdiagramm die Option Top-SQL aus.

   Das Diagramm listet die SQL-Abfragen auf, die für die Belastung verantwortlich sind. Diejenigen, die an der Spitze der Liste stehen, sind am meisten verantwortlich. Konzentrieren Sie sich auf diese Aussagen, um einen Engpass zu beheben.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights.

Performance Insights verwenden

Dieses Ereignis hängt mit der Workload zusammen. Mit Performance Insights können Sie Folgendes tun:

- Identifizieren Sie, wann Warteereignisse beginnen und ob sich die Workload zu dieser Zeit aus den Anwendungsprotokollen oder verwandten Quellen ändert.
- Identifizieren Sie die SQL-Anweisungen, die für dieses Warteereignis verantwortlich sind. Untersuchen Sie den Ausführungsplan der Abfragen, um sicherzustellen, dass diese Abfragen optimiert sind und geeignete Indizes verwenden.

  Wenn die obersten Abfragen, die für das Warteereignis verantwortlich sind, sich auf dasselbe Datenbankobjekt oder dieselbe Tabelle beziehen, sollten Sie erwägen, dieses Objekt oder die Tabelle zu partitionieren.

Erstellen eines Aurora-Replikats

Sie können Aurora-Replikationen erstellen, um schreibgeschützten Datenverkehr bereitzustellen. Sie können Aurora Auto Scaling auch verwenden, um Überspannungen im Leseverkehr zu behandeln. Stellen Sie sicher, dass geplante schreibgeschützte Aufgaben und logische Backups auf Aurora-Replikaten ausgeführt werden.

Weitere Informationen finden Sie unter Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten.

Überprüfen Sie die Größe des Pufferpools

Prüfen Sie anhand der Metrik `innodb_buffer_pool_wait_free`, ob die Pufferpoolgröße für die Workload ausreicht. Wenn der Wert dieser Metrik hoch ist und kontinuierlich zunimmt, deutet dies darauf hin, dass die Größe des Pufferpools nicht ausreicht, um die Workload zu bewältigen. Wenn `innodb_buffer_pool_size` richtig eingestellt wurde, sollte der Wert von `innodb_buffer_pool_wait_free` klein sein. Weitere Informationen finden Sie unter Innodb_buffer_pool_wait_free in der MySQL-Dokumentation.

Erhöhen Sie die Pufferpool-Größe, wenn die DB-Instance über genügend Speicher für Sitzungspuffer und Betriebssystemaufgaben verfügt. Wenn dies nicht der Fall ist, ändern Sie die DB-Instance in eine größere DB-Instance-Klasse, um zusätzlichen Speicher zu erhalten, der dem Pufferpool zugewiesen werden kann.

> **Note**
>
> Aurora MySQL passt den Wert von `innodb_buffer_pool_instances` automatisch basierend auf der konfigurierten `innodb_buffer_pool_size` an.

Überwachen Sie den globalen Statusverlauf

Durch die Überwachung der Änderungsraten von Statusvariablen können Sie Sperr- oder Speicherprobleme auf Ihrer DB-Instance erkennen. Aktivieren Sie den globalen Statusverlauf (GoSH), wenn er noch nicht aktiviert ist. Weitere Informationen zu GoSH finden Sie unter Verwalten des globalen Statusverlaufs.

Sie können auch benutzerdefinierte Amazon-CloudWatch-Metriken erstellen, um Statusvariablen zu überwachen. Weitere Informationen finden Sie unter Veröffentlichen benutzerdefinierter Metriken.

synch/mutex/innodb/fil_system_mutex

Dieses `synch/mutex/innodb/fil_system_mutex`-Ereignis tritt auf, wenn eine Sitzung darauf wartet, auf den Tablespace-Speichercache zuzugreifen.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Versionen 2 und 3

Kontext

InnoDB verwendet Tablespaces, um den Speicherbereich für Tabellen und Protokolldateien zu verwalten. Der Tablespace-Speichercache ist eine globale Speicherstruktur, die Informationen

über Tablespaces verwaltet. MySQL verwendet `synch/mutex/innodb/fil_system_mutex` Wartezeiten, um den gleichzeitigen Zugriff auf den Tablespace-Speichercache zu steuern.

Das Ereignis `synch/mutex/innodb/fil_system_mutex` zeigt an, dass derzeit mehr als eine Operation vorhanden ist, die Informationen im Tabellenbereichsspeichercache für denselben Tabellenbereich abrufen und bearbeiten muss.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `synch/mutex/innodb/fil_system_mutex`-Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, tritt dies normalerweise auf, wenn alle der folgenden Bedingungen vorliegen:

- Eine Zunahme der gleichzeitigen DML-Vorgänge (Data Manipulation Language), die Daten in derselben Tabelle aktualisieren oder löschen.
- Der Tablespace für diese Tabelle ist sehr groß und hat viele Datenseiten.
- Der Füllfaktor für diese Datenseiten ist niedrig.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen
- Reorganisieren Sie große Tabellen außerhalb der Hauptverkehrszeiten

Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen

Normalerweise weisen Datenbanken mit mäßiger bis erheblicher Last Warteereignisse auf. Die Warteereignisse sind möglicherweise akzeptabel, wenn die Leistung optimal ist. Wenn die Leistung nicht optimal ist, untersuchen Sie, wo die Datenbank die meiste Zeit verbringt. Schauen Sie sich die Warteereignisse an, die zur höchsten Belastung beitragen und finden Sie heraus, ob Sie die Datenbank und die Anwendung optimieren können, um diese Ereignisse zu reduzieren.

So suchen Sie SQL-Abfragen, die für hohe Last verantwortlich sind

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus. Das Performance-Insights-Dashboard wird für diese DB-Instance angezeigt.
4. Wählen Sie im Diagramm zur Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Das Diagramm listet die SQL-Abfragen auf, die für die Belastung verantwortlich sind. Diejenigen, die an der Spitze der Liste stehen, sind am meisten verantwortlich. Konzentrieren Sie sich auf diese Aussagen, um einen Engpass zu beheben.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights.

Eine andere Möglichkeit herauszufinden, welche Abfragen eine hohe Anzahl von `synch/mutex/innodb/fil_system_mutex`-Wartezeiten verursachen, besteht darin, `performance_schema` zu überprüfen, wie im folgenden Beispiel.

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/
synch/mutex/innodb/fil_system_mutex'\G
*************************** 1. row ***************************
            THREAD_ID: 19
             EVENT_ID: 195057
         END_EVENT_ID: 195057
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:6700
          TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
           TIMER_WAIT: 6405600
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 2. row ***************************
```

```
            THREAD_ID: 23
             EVENT_ID: 5480
         END_EVENT_ID: 5480
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:5906
          TIMER_START: 995269979908800
            TIMER_END: 995269980159200
           TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 3. row ***************************
            THREAD_ID: 55
             EVENT_ID: 23233794
         END_EVENT_ID: NULL
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:449
          TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
           TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: 23233786
   NESTING_EVENT_TYPE: WAIT
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
```

Reorganisieren Sie große Tabellen außerhalb der Hauptverkehrszeiten

Reorganisieren Sie große Tabellen, die Sie während eines Wartungsfensters außerhalb der Produktionszeiten als Quelle einer großen Anzahl von `synch/mutex/innodb/fil_system_mutex`-Warteereignissen identifizieren. Dadurch wird sichergestellt, dass die Bereinigung der internen Tablespaces-Zuordnung nicht erfolgt, wenn der schnelle Zugriff auf die Tabelle kritisch ist. Informationen zum Reorganisieren von Tabellen finden Sie unter OPTIMIZE TABLE-Anweisung in der MySQL-Referenz.

synch/mutex/innodb/trx_sys_mutex

Dieses `synch/mutex/innodb/trx_sys_mutex`-Ereignis tritt auf, wenn eine hohe Datenbankaktivität mit einer großen Anzahl von Transaktionen besteht.

Themen

- Relevante Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Relevante Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Versionen 2 und 3

Kontext

Intern verwendet das InnoDB-Datenbankmodul die wiederholbare Lese-Isolationsstufe mit Snapshots, um Lesekonsistenz zu gewährleisten. Dies gibt Ihnen eine zeitpunktbezogene Ansicht der Datenbank zum Zeitpunkt der Erstellung des Snapshots.

In InnoDB werden alle Änderungen auf die Datenbank angewendet, sobald sie eintreffen, unabhängig davon, ob sie festgeschrieben wurden. Dieser Ansatz bedeutet, dass ohne Multiversions-Parallelitätssteuerung (MVCC) alle mit der Datenbank verbundenen Benutzer alle Änderungen und die neuesten Zeilen sehen. Daher benötigt InnoDB eine Möglichkeit, die Änderungen zu verfolgen, um zu verstehen, was bei Bedarf zurückgesetzt werden soll.

Dazu verwendet InnoDB ein Transaktionssystem (`trx_sys`) zum Verfolgen von Snapshots. Das Transaktionssystem führt folgenden Aktionen:

- Verfolgt die Transaktions-ID für jede Zeile in den Rückgängig-Protokollen.
- Verwendet eine interne InnoDB-Struktur namens `ReadView`, die hilft zu identifizieren, welche Transaktions-IDs für einen Snapshot sichtbar sind.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Jede Datenbankoperation, die eine konsistente und kontrollierte Behandlung (Erstellen, Lesen, Aktualisieren und Löschen) von Transaktions-IDs erfordert, erzeugt einen Aufruf von `trx_sys` an den Mutex.

Diese Aufrufe erfolgen innerhalb von drei Funktionen:

- `trx_sys_mutex_enter` – Erzeugt den Mutex.
- `trx_sys_mutex_exit` – Gibt den Mutex frei.
- `trx_sys_mutex_own` – Testet, ob der Mutex im Besitz ist.

Die Instrumentierung des InnoDB-Leistungsschemas verfolgt alle `trx_sys`-Mutex-Aufrufe. Die Verfolgung umfasst unter anderem die Verwaltung von `trx_sys` beim Starten oder Herunterfahren der Datenbank, Rollback-Vorgänge, das Rückgängigmachen von Bereinigungen, den Zeilenlesezugriff und das Laden von Pufferpools. Eine hohe Datenbankaktivität mit einer großen Anzahl von Transaktionen führt dazu, dass `synch/mutex/innodb/trx_sys_mutex` unter den Top-Warteereignissen erscheint.

Weitere Informationen finden Sie unter Überwachen von InnoDB-Mutex-Wartezeiten mithilfe des Leistungsschemas in der MySQL-Dokumentation.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen
- Untersuchen Sie andere Warteereignisse

Identifizieren der Sitzungen und Abfragen, die die Ereignisse verursachen

Normalerweise weisen Datenbanken mit mäßiger bis erheblicher Last Warteereignisse auf. Die Warteereignisse sind möglicherweise akzeptabel, wenn die Leistung optimal ist. Wenn die Leistung nicht optimal ist, untersuchen Sie, wo die Datenbank die meiste Zeit verbringt. Schauen Sie sich die Warteereignisse an, die zur höchsten Belastung beitragen. Finden Sie heraus, ob Sie die Datenbank und die Anwendung optimieren können, um diese Ereignisse zu reduzieren.

So zeigen Sie das Top-SQL-Diagramm in AWS-Managementkonsole an

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus. Das Performance-Insights-Dashboard wird für diese DB-Instance angezeigt.
4. Wählen Sie im Diagramm zur Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unter dem Datenbankauslastungsdiagramm die Option Top-SQL aus.

   Das Diagramm listet die SQL-Abfragen auf, die für die Belastung verantwortlich sind. Diejenigen, die an der Spitze der Liste stehen, sind am meisten verantwortlich. Konzentrieren Sie sich auf diese Aussagen, um einen Engpass zu beheben.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im Blogbeitrag Analysieren Sie Amazon Aurora MySQL Workloads mit Performance Insights.

Untersuchen Sie andere Warteereignisse

Untersuchen Sie die anderen Warteereignisse, die dem `synch/mutex/innodb/trx_sys_mutex`-Warteereignis zugeordnet sind. Auf diese Weise finden Sie weitere Informationen über die Art der Workload. Eine große Anzahl von Transaktionen kann den Durchsatz verringern, die Workload kann dies jedoch auch erforderlich machen.

Weitere Informationen zum Optimieren von Transaktionen finden Sie unter Optimieren der InnoDB-Transaktionsverwaltung in der MySQL-Dokumentation.

synch/sxlock/innodb/hash_table_locks

Das `synch/sxlock/innodb/hash_table_locks`-Ereignis tritt ein, wenn Seiten, die nicht im Pufferpool gefunden wurden, aus dem Speicher gelesen werden müssen.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Versionen unterstützt:

- Aurora-MySQL-Versionen 2 und 3

Kontext

Das Ereignis `synch/sxlock/innodb/hash_table_locks` zeigt an, dass ein Workload häufig auf Daten zugreift, die nicht im Pufferpool gespeichert sind. Dieses Warteereignis ist mit neuen Seitenhinzufügungen und alten Datenräumungen aus dem Pufferpool verbunden. Die im Pufferpool gespeicherten Daten und neuen Daten müssen zwischengespeichert werden, sodass die gealterten Seiten geräumt werden, um das Zwischenspeichern der neuen Seiten zu ermöglichen. MySQL verwendet einen am wenigsten verwendeten (LRU) -Algorithmus, um Seiten aus dem Puffer-Pool zu entfernen. Die Workload versucht, auf Daten zuzugreifen, die nicht in den Pufferpool geladen wurden, oder auf Daten, die aus dem Pufferpool vertrieben wurden.

Dieses Warteereignis tritt auf, wenn der Workload auf die Daten in Dateien auf der Festplatte zugreifen muss oder wenn Blöcke von der LRU-Liste des Pufferpools befreit oder zur LRU-Liste des Pufferpools hinzugefügt werden. Diese Vorgänge warten darauf, eine gemeinsame ausgeschlossene Sperre (SX-Lock) zu erhalten. Diese SX-Sperre wird für die Synchronisation über die Hash-Tabelle verwendet, bei der es sich um eine Tabelle im Speicher handelt, die die Leistung des Pufferpoolzugriffs verbessern soll.

Weitere Informationen finden Sie unter Bufferpool in der MySQL-Dokumentation.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `synch/sxlock/innodb/hash_table_locks`-Warteereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

Ein unterdimensionierter Pufferpool

Die Größe des Pufferpools ist zu klein, um alle häufig aufgerufenen Seiten im Speicher zu behalten.

Starke Workload

Die Workload verursacht häufige Bereinigungen und das Neuladen von Datenseiten in den Puffercache.

Fehler beim Lesen der Seiten

Es gibt Fehler beim Lesen von Seiten im Pufferpool, die auf eine Beschädigung von Daten hinweisen können.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Erhöhen Sie die Größe des Pufferpools
- Verbesserung der Datenzugriffsmuster
- Reduzieren oder vermeiden Sie vollständige Tabellen-Scans
- Überprüfen Sie die Fehlerprotokolle auf Seitenbeschädigung

Erhöhen Sie die Größe des Pufferpools

Stellen Sie sicher, dass diese Pufferpools für die Workload entsprechend dimensioniert sind. Sie können dazu die Trefferrate des Pufferpools überprüfen. Wenn der Wert unter 95 Prozent sinkt, sollten Sie in der Regel die Größe des Pufferpools erhöhen. Ein größerer Pufferpool kann häufig aufgerufene Seiten länger im Speicher halten. Um die Größe des Pufferpools zu vergrößern, ändern Sie den Wert des `innodb_buffer_pool_size`-Parameters. Der Standardwert dieses Parameters basiert auf der DB-Instance-Klassengröße. Weitere Informationen finden Sie unter bewährte Methoden für die Amazon-Aurora-MySQL-Datenbankkonfiguration.

Verbesserung der Datenzugriffsmuster

Überprüfen Sie die von dieser Wartezeit betroffenen Abfragen und deren Ausführungspläne. Erwägen Sie, die Datenzugriffsmuster zu verbessern. Wenn Sie beispielsweise mysqli_result:: fetch_array verwenden können Sie versuchen, die Abrufgröße des Arrays zu erhöhen.

Sie können Performance Insights verwenden, um Abfragen und Sitzungen anzuzeigen, die möglicherweise ein `synch/sxlock/innodb/hash_table_locks`-Warteereignis verursachen.

So suchen Sie SQL-Abfragen, die für hohe Last verantwortlich sind

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine DB-Instance aus. Das Performance-Insights-Dashboard wird für diese DB-Instance angezeigt.
4. Wählen Sie im Diagramm zur Datenbanklast die Option Nach Wartezeit aufteilen.
5. Wählen Sie unten auf der Seite Top-SQL aus.

   Das Diagramm listet die SQL-Abfragen auf, die für die Belastung verantwortlich sind. Diejenigen, die an der Spitze der Liste stehen, sind am meisten verantwortlich. Konzentrieren Sie sich auf diese Aussagen, um einen Engpass zu beheben.

Eine nützliche Übersicht über die Fehlerbehebung mit Performance Insights finden Sie im AWS-Database Blog-Beitrag Analysieren von Amazon-Aurora-MySQL-Workloads mit Performance Insights.

Reduzieren oder vermeiden Sie vollständige Tabellen-Scans

Überwachen Sie Ihre Workload, um zu sehen, ob vollständige Tabellen-Scans ausgeführt werden, und wenn dies der Fall ist, reduzieren oder vermeiden Sie sie. Sie können beispielsweise Statusvariablen wie `Handler_read_rnd_next` überwachen. Weitere Informationen finden Sie unter Server-Status-Variablen in der MySQL-Dokumentation.

Überprüfen Sie die Fehlerprotokolle auf Seitenbeschädigung

Sie können mysql-error.log auf korruptionsbezogene Nachrichten überprüfen, die zum Zeitpunkt des Problems erkannt wurden. Nachrichten, mit denen Sie arbeiten können, um das Problem zu beheben, befinden sich im Fehlerprotokoll. Möglicherweise müssen Sie Objekte neu erstellen, die als beschädigt gemeldet wurden.

synch/mutex/innodb/temp_poolmanager_mutex

Das `synch/mutex/innodb/temp_pool_manager_mutex` Wait-Ereignis tritt ein, wenn eine Sitzung darauf wartet, einen Mutex für die Verwaltung des Pools temporärer Sitzungs-Tablespaces zu erhalten.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für die folgenden Engine-Versionen unterstützt:

- Aurora-MySQL-Version 3

Kontext

Aurora MySQL Version 3.x und höher steuert mehrere Sitzungen`temp_pool_manager_mutex`, die gleichzeitig auf den temporären Tablespace-Pool zugreifen. Aurora MySQL verwaltet Speicher über ein Aurora-Cluster-Volume für persistente Daten und lokalen Speicher für temporäre Dateien. Ein temporärer Tablespace wird benötigt, wenn eine Sitzung eine temporäre Tabelle auf dem Aurora-Cluster-Volume erstellt.

Wenn eine Sitzung zum ersten Mal einen temporären Tablespace anfordert, weist MySQL temporäre Sitzungs-Tablespaces aus dem Shared Pool zu. Eine Sitzung kann bis zu 2 temporäre Tablespaces gleichzeitig für die folgenden Tabellentypen enthalten:

- Vom Benutzer erstellte temporäre Tabellen
- Von Optimizer generierte interne temporäre Tabellen

Die `TempTable` Standard-Engine verwendet den folgenden Überlaufmechanismus, um temporäre Tabellen zu verarbeiten:

- Speichert Tabellen im RAM bis zum `temptable_max_ram`Limit.

- Wechselt zu Dateien im lokalen Speicher, die dem Arbeitsspeicher zugeordnet sind, wenn der Arbeitsspeicher voll ist.
- Verwendet das gemeinsam genutzte Cluster-Volume, wenn die Speicherzuordnungsdateien ihr Limit erreichen. `temptable_max_mmap`

Wenn temporäre Tabellen sowohl die RAM- als auch die lokalen Speichergrenzen überschreiten, verwaltet MySQL sie mithilfe von Tablespace auf der Festplatte.

Wenn für eine Sitzung eine temporäre Tabelle auf der Festplatte erforderlich ist, kann MySQL:

- Sucht nach verfügbaren `INACTIVE` Tablespaces im Pool, die wiederverwendet werden können.
- Erzeugt 10 neue Tablespaces, wenn keine Leerzeichen vorhanden sind. `INACTIVE`

Wenn eine Sitzung unterbrochen wird, geht MySQL wie folgt vor:

- Kürzt die temporären Tablespaces der Sitzung.
- Markiert sie im Pool zur Wiederverwendung als INAKTIV.
- Behält die aktuelle Poolgröße bis zum Neustart des Servers bei.
- Kehrt nach dem Neustart zur Standard-Poolgröße (10 Tablespaces) zurück.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Häufige Situationen, die dieses Warteereignis verursachen:

- Gleichzeitige Sitzungen, die interne temporäre Tabellen auf dem Cluster-Volume erstellen.
- Gleichzeitige Sitzungen, in denen temporäre Benutzertabellen auf dem Cluster-Volume erstellt werden.
- Plötzliches Beenden von Sitzungen, die aktive Tablespaces verwenden.
- Erweiterung des Tablespace-Pools bei hohen Schreiblasten.
- Gleichzeitige Abfragen, auf die zugegriffen wird `INFORMATION_SCHEMA.`

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Überwachen und optimieren Sie die Nutzung temporärer Tabellen
- Überprüfen Sie Abfragen mit INFORMATION_SCHEMA
- Erhöhen Sie den Parameter innodb_sync_array_size
- Verbindungspooling implementieren

Überwachen und optimieren Sie die Nutzung temporärer Tabellen

Verwenden Sie eine der folgenden Methoden, um die Nutzung temporärer Tabellen zu überwachen und zu optimieren:

- Überprüfen Sie den `Created_tmp_disk_tables` Zähler in Performance Insights, um die Erstellung temporärer Tabellen auf der Festplatte in Ihrem Aurora-Cluster nachzuverfolgen.
- Führen Sie diesen Befehl in Ihrer Datenbank aus, um die Erstellung temporärer Tabellen direkt zu überwachen:`mysql> show status like '%created_tmp_disk%'`.

> **Note**
>
> Das Verhalten temporärer Tabellen unterscheidet sich zwischen Aurora MySQL-Reader-Knoten und Writer-Knoten. Weitere Informationen finden Sie unter Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3.

Nachdem Sie Abfragen identifiziert haben, die temporäre Tabellen erstellen, führen Sie die folgenden Optimierungsschritte durch:

- Wird verwendet`EXPLAIN`, um Ausführungspläne für Abfragen zu untersuchen und zu ermitteln, wo und warum temporäre Tabellen erstellt werden.
- Ändern Sie Abfragen, um die Verwendung temporärer Tabellen nach Möglichkeit zu reduzieren.

Wenn Leistungsprobleme durch Abfrageoptimierung allein nicht behoben werden können, sollten Sie die folgenden Konfigurationsparameter anpassen:

- `temptable_max_ram`- Steuert die maximale RAM-Auslastung für temporäre Tabellen.
- `temptable_max_mmap`- Legt das Limit für die Speicherung von Dateien mit Speicherabbildung fest.

- `tmp_table_size`- Gilt, wenn `aurora_tmptable_enable_per_table_limit` aktiviert (standardmäßig deaktiviert).

> **Important**
>
> Beachten Sie, dass für bestimmte Abfragebedingungen immer temporäre Tabellen auf der Festplatte erforderlich sind, unabhängig von den Konfigurationseinstellungen. Weitere Informationen zur `TempTable` Speicher-Engine finden Sie unter Verwenden der TempTable Speicher-Engine auf Amazon RDS for MySQL und Amazon Aurora MySQL.

Überprüfen Sie Abfragen mit INFORMATION_SCHEMA

Wenn Sie `INFORMATION_SCHEMA` Tabellen abfragen, erstellt MySQL temporäre InnoDB-Tabellen auf dem Cluster-Volume. Jede Sitzung benötigt einen temporären Tablespace für diese Tabellen, was bei vielen gleichzeitigen Zugriffen zu Leistungseinbußen führen kann.

Um die Leistung zu verbessern:

- Verwenden Sie `PERFORMANCE_SCHEMA` statt `INFORMATION_SCHEMA` wo möglich.
- Wenn Sie diese Abfragen verwenden müssen`INFORMATION_SCHEMA`, reduzieren Sie die Häufigkeit, mit der Sie diese Abfragen ausführen.

Erhöhen Sie den Parameter innodb_sync_array_size

Der `innodb_sync_array_size` Parameter steuert die Größe des mutex/lock Warte-Arrays in MySQL. Der Standardwert von 1 funktioniert für allgemeine Workloads, aber wenn Sie ihn erhöhen, können Thread-Konflikte bei hoher Parallelität reduziert werden.

Wenn Ihre Arbeitslast eine zunehmende Anzahl wartender Threads aufweist:

- Überwachen Sie die Anzahl der wartenden Threads in Ihrem Workload.
- Legen Sie den `innodb_sync_array_size` Wert gleich oder höher als die vCPU-Anzahl Ihrer Instance fest, um die Thread-Koordinationsstruktur aufzuteilen und Konflikte zu reduzieren.

> **Note**
>
> Informationen zur Anzahl der auf Ihrer RDS-Instance CPUs verfügbaren v finden Sie in den vCPU-Spezifikationen unter Amazon RDS-Instance-Typen.

Verbindungspooling implementieren

MySQL weist jeder Sitzung, die eine temporäre Tabelle erstellt, einen eigenen Tablespace zu. Dieser Tablespace bleibt aktiv, bis die Datenbankverbindung beendet wird. Gehen Sie wie folgt vor, um Ihre Ressourcen effizienter zu verwalten:

- Implementieren Sie Verbindungspooling, um die Anzahl der aktiven temporären Tablespaces zu begrenzen.
- Verwenden Sie bestehende Verbindungen wieder, anstatt für jeden Vorgang neue zu erstellen.

Optimierung von Aurora MySQL mit Thread-Status

In der folgenden Tabelle werden die häufigsten allgemeinen Thread-Zustände für Aurora MySQL zusammengefasst.

Allgemeiner Thread-Zustand	Beschreibung
???	Dieser Thread-Zustand zeigt an, dass ein Thread eine `SELECT`-Anweisung verarbeitet, die die Verwendung einer internen temporären Tabelle zum Sortieren der Daten erfordert.
???	Dieser Thread-Zustand zeigt an, dass ein Thread Zeilen für eine Abfrage liest und filtert, um die richtige Ergebnismenge zu ermitteln.

Erstellen des Sortierindex

Dieser `creating sort index`-Thread-Zustand zeigt an, dass ein Thread eine `SELECT`-Anweisung verarbeitet, die die Verwendung einer internen temporären Tabelle zum Sortieren der Daten erfordert.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Thread-Statusinformationen werden für die folgenden Versionen unterstützt:

- Aurora MySQL Version 2 bis 2.09.2

Kontext

Der Zustand `creating sort index` wird angezeigt, wenn eine Abfrage mit einer `ORDER BY`- oder `GROUP BY`-Klausel keinen vorhandenen Index verwenden kann, um die Operation auszuführen.

In diesem Fall muss MySQL eine teurere `filesort`-Operation ausführen. Dieser Vorgang wird normalerweise im Speicher ausgeführt, wenn die Ergebnismenge nicht zu groß ist. Andernfalls geht es darum, eine Datei auf der Festplatte zu erstellen.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Das Auftreten von `creating sort index` deutet nicht allein auf ein Problem hin. Wenn die Leistung schlecht ist und Sie häufig `creating sort index`-Instances sehen, sind die wahrscheinlichste Ursache langsame Abfragen mit `ORDER BY`- oder `GROUP BY`-Operatoren.

Aktionen

Die allgemeine Richtlinie besteht darin, Abfragen mit `ORDER BY`- oder `GROUP BY`-Klauseln zu finden, die mit den Erhöhungen des Zustands `creating sort index` verbunden sind. Prüfen Sie dann, ob das Hinzufügen eines Index oder das Vergrößern der Sortierpuffergröße das Problem löst.

Themen

- Schalten Sie das Leistungsschema ein, wenn es nicht aktiviert ist
- Identifizieren Sie die Problemanfragen
- Untersuchen Sie die Erklärungspläne für die Dateisortierungs-Nutzung
- Erhöhen Sie die Sortierpuffergröße

Schalten Sie das Leistungsschema ein, wenn es nicht aktiviert ist

Performance Insights meldet Thread-Status nur, wenn Leistungsschema-Instrumente nicht aktiviert sind. Wenn Leistungsschema-Instrumente aktiviert sind, meldet Performance Insights stattdessen Warteereignisse. Leistungsschema-Instrumente bieten zusätzliche Erkenntnisse und bessere Tools, wenn Sie potenzielle Leistungsprobleme untersuchen. Daher wird empfohlen, dass Sie das Leistungsschema aktivieren. Weitere Informationen finden Sie unter Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL.

Identifizieren Sie die Problemanfragen

KennUm aktuelle Abfragen zu identifizieren, die eine Erhöhung des `creating sort index`-Zustands verursachen, führen Sie `show processlist` aus und prüfen Sie, ob eine der Abfragen `ORDER BY` oder `GROUP BY` hat. Führen Sie optional `explain for connection N` aus, wobei `N` die Prozesslisten-ID der Abfrage mit `filesort` ist.

Um frühere Abfragen zu identifizieren, die diese Zunahmen verursachen, aktivieren Sie das Protokoll für langsame Abfragen und suchen Sie die Abfragen mit `ORDER BY`. Führen Sie `EXPLAIN` für die langsamen Abfragen aus und suchen Sie nach „using filesort“. Weitere Informationen finden Sie unter Untersuchen Sie die Erklärungspläne für die Dateisortierungs-Nutzung.

Untersuchen Sie die Erklärungspläne für die Dateisortierungs-Nutzung

Identifizieren Sie die Anweisungen mit `ORDER BY`- oder `GROUP BY`-Klauseln, die den `creating sort index`-Zustand ergeben.

Das folgende Beispiel zeigt, wie `explain` für eine Abfrage ausgeführt wird. Die Spalte `Extra` zeigt, dass diese Abfrage `filesort` verwendet.

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2064548
     filtered: 100.00
        Extra: Using filesort
1 row in set, 1 warning (0.01 sec)
```

Das folgende Beispiel zeigt das Ergebnis der Ausführung von `EXPLAIN` für dieselbe Abfrage, nachdem ein Index für Spalte `c1` erstellt wurde.

```
mysql> alter table mytable add index (c1);
```

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: index
possible_keys: NULL
```

```
          key: c1
      key_len: 1023
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.01 sec)
```

Informationen zur Verwendung von Indizes zur Optimierung der Sortierreihenfolge finden Sie unter ORDER BY-Optimierung in der MySQL-Dokumentation.

Erhöhen Sie die Sortierpuffergröße

Um festzustellen, ob für eine bestimmte Abfrage ein `filesort`-Prozess erforderlich ist, der eine Datei auf dem Datenträger erstellt hat, überprüfen Sie den `sort_merge_passes`-Variablenwert, nachdem Sie die Abfrage ausgeführt haben. Es folgt ein Beispiel.

```
mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)
```

Wenn der Wert von `sort_merge_passes` hoch ist, sollten Sie die Größe des Sortierpuffers erhöhen. Wenden Sie den Anstieg auf Sitzungsebene an, da eine globale Erhöhung der RAM-MySQL-Nutzung erheblich erhöhen kann. Das folgende Beispiel zeigt, wie Sie die Größe des Sortierpuffers ändern, bevor Sie eine Abfrage ausführen.

```
mysql> set session sort_buffer_size=10*1024*1024;
Query OK, 0 rows affected (0.00 sec)
-- run query
```

Senden von Daten

Dieser `sending data`-Thread-Zustand zeigt an, dass ein Thread Zeilen für eine Abfrage liest und filtert, um die richtige Ergebnismenge zu ermitteln. Der Name ist irreführend, da er andeutet, dass der Zustand Daten überträgt, nicht sammelt und vorbereitet, um später gesendet zu werden.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Thread-Statusinformationen werden für die folgenden Versionen unterstützt:

- Aurora MySQL Version 2 bis 2.09.2

Kontext

Viele Thread-Zustände sind kurzlebig. Operationen, die während `sending data` auftreten, führen in der Regel eine große Anzahl von Platten- oder Cache-Lesevorgängen durch. Daher ist `sending data` häufig der am längsten ausgeführte Zustand während der Lebensdauer einer bestimmten Abfrage. Dieser Status wird angezeigt, wenn Aurora MySQL Folgendes tut:

- Lesen und Verarbeiten von Zeilen für eine `SELECT`-Anweisung
- Durchführen einer großen Anzahl von Lesevorgängen von der Festplatte oder vom Speicher
- Vollständiges Lesen aller Daten aus einer bestimmten Abfrage durchführen
- Lesen von Daten aus einer Tabelle, einem Index oder der Arbeit einer gespeicherten Prozedur
- Daten sortieren, gruppieren oder bestellen

Nachdem der `sending data`-Zustand die Vorbereitung der Daten abgeschlossen hat, zeigt der Thread-Zustand `writing to net` die Rückgabe der Daten an den Client an. Normalerweise wird `writing to net` nur erfasst, wenn die Ergebnismenge sehr groß ist oder eine starke Netzwerklatenz die Übertragung verlangsamt.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Das Auftreten von `sending data` deutet nicht allein auf ein Problem hin. Wenn die Leistung schlecht ist und Sie häufige Instances von `sending data` sehen, sind die wahrscheinlichsten Ursachen wie folgt.

Themen

- Ineffiziente Abfrage
- Suboptimale Serverkonfiguration

Ineffiziente Abfrage

In den meisten Fällen ist für diesen Status eine Abfrage verantwortlich, die keinen geeigneten Index verwendet, um die Ergebnismenge einer bestimmten Abfrage zu finden. Betrachten Sie beispielsweise eine Abfrage, die eine 10-Millionen-Datensatztabelle für alle Bestellungen in Kalifornien liest, wobei die Bundesstaatsspalte nicht indiziert oder schlecht indiziert ist. Im letzteren Fall könnte der Index existieren, aber der Optimierer ignoriert ihn aufgrund der geringen Kardinalität.

Suboptimale Serverkonfiguration

Wenn im Zustand `sending data` mehrere Abfragen erscheinen, ist der Datenbankserver möglicherweise schlecht konfiguriert. Insbesondere könnte der Server die folgenden Probleme haben:

- Der Datenbankserver verfügt nicht über genügend Rechenkapazität: Festplatten-I/O, Festplattentyp und Geschwindigkeit, CPU oder Anzahl der CPUs.
- Der Server sortiert nach allokierten Ressourcen, wie dem InnoDB-Pufferpool für InnoDB-Tabellen oder dem Schlüsselpuffer für MyIsam-Tabellen.
- Arbeitsspeichereinstellungen pro Thread wie `sort_buffer`, `read_buffer` und `join_buffer` verbrauchen mehr RAM als erforderlich, wodurch der physische Server an Speicherressourcen mangelt.

Aktionen

Die allgemeine Richtlinie besteht darin, Abfragen zu finden, die eine große Anzahl von Zeilen zurückgeben, indem Sie das Leistungsschema überprüfen. Wenn die Protokollierung von Abfragen aktiviert ist, die keine Indizes verwenden, können Sie auch die Ergebnisse der langsamen Protokolle untersuchen.

Themen

- Schalten Sie das Leistungsschema ein, wenn es nicht aktiviert ist
- Prüfen Sie die Speichereinstellungen
- Untersuchen Sie die Erklärungspläne zur Indexverwendung
- Überprüfen Sie das Volume der zurückgegebenen Daten
- Auf Parallelitätsprobleme prüfen
- Überprüfen der Struktur Ihrer Abfrage

Schalten Sie das Leistungsschema ein, wenn es nicht aktiviert ist

Performance Insights meldet Thread-Status nur, wenn Leistungsschema-Instrumente nicht aktiviert sind. Wenn Leistungsschema-Instrumente aktiviert sind, meldet Performance Insights stattdessen Warteereignisse. Leistungsschema-Instrumente bieten zusätzliche Erkenntnisse und bessere Tools, wenn Sie potenzielle Leistungsprobleme untersuchen. Daher wird empfohlen, dass Sie das Leistungsschema aktivieren. Weitere Informationen finden Sie unter Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL.

Prüfen Sie die Speichereinstellungen

Untersuchen Sie die Speichereinstellungen für die primären Pufferpools. Stellen Sie sicher, dass diese Pools für die Workload entsprechend dimensioniert sind. Wenn Ihre Datenbank mehrere Pufferpool-Instances verwendet, stellen Sie sicher, dass sie nicht in viele kleine Pufferpools unterteilt sind. Threads können jeweils nur einen Pufferpool verwenden.

Stellen Sie sicher, dass die folgenden Speichereinstellungen, die für jeden Thread verwendet werden, richtig dimensioniert sind:

- read_buffer
- read_rnd_buffer

- sort_buffer
- join_buffer
- binlog_cache

Wenn Sie keinen bestimmten Grund haben, die Einstellungen zu ändern, verwenden Sie die Standardwerte.

Untersuchen Sie die Erklärungspläne zur Indexverwendung

Überprüfen Sie bei Abfragen im `sending data`-Thread-Zustand den Plan, um festzustellen, ob geeignete Indizes verwendet werden. Wenn eine Abfrage keinen nützlichen Index verwendet, sollten Sie Hinweise wie `USE INDEX` oder `FORCE INDEX` hinzufügen. Hinweise können die Zeit zum Ausführen einer Abfrage erheblich verlängern oder verkürzen, seien Sie also vorsichtig, bevor Sie sie hinzufügen.

Überprüfen Sie das Volume der zurückgegebenen Daten

Überprüfen Sie die abgefragten Tabellen und die Menge der Daten, die sie enthalten. Kann irgendeine dieser Daten archiviert werden? In vielen Fällen ist die Ursache für schlechte Ausführungszeiten der Abfragen nicht das Ergebnis des Abfrageplans, sondern das Datenvolumen, das verarbeitet werden soll. Viele Entwickler fügen einer Datenbank sehr effizient Daten hinzu, berücksichtigen jedoch selten den Lebenszyklus von Datensätzen in den Entwurfs- und Entwicklungsphasen.

Suchen Sie nach Abfragen, die in Datenbanken mit geringem Volume gut funktionieren, aber in Ihrem aktuellen System schlecht funktionieren. Manchmal erkennen Entwickler, die bestimmte Abfragen entwerfen, möglicherweise nicht, dass diese Abfragen 350.000 Zeilen zurückgeben. Die Entwickler haben die Abfragen möglicherweise in einer Umgebung mit geringerem Volumen und kleineren Datensätzen entwickelt als in Produktionsumgebungen.

Auf Parallelitätsprobleme prüfen

Überprüfen Sie, ob mehrere Abfragen desselben Typs gleichzeitig ausgeführt werden. Einige Formen von Abfragen werden effizient ausgeführt, wenn sie alleine laufen. Wenn jedoch ähnliche Abfrageformen zusammen oder in hohem Volumen ausgeführt werden, können sie Parallelitätsprobleme verursachen. Oft werden diese Probleme verursacht, wenn die Datenbank temporäre Tabellen verwendet, um Ergebnisse zu rendern. Eine restriktive Transaktionsisolationsstufe kann auch Parallelitätsprobleme verursachen.

Wenn Tabellen gleichzeitig gelesen und geschrieben werden, verwendet die Datenbank möglicherweise Sperren. Um Perioden schlechter Leistung zu identifizieren, untersuchen Sie die Verwendung von Datenbanken durch umfangreiche Batch-Prozesse. Um die letzten Sperren und Rollbacks anzuzeigen, überprüfen Sie die Ausgabe des `SHOW ENGINE INNODB STATUS`-Befehls.

Überprüfen der Struktur Ihrer Abfrage

Prüfen Sie, ob erfasste Abfragen aus diesen Status Unterabfragen verwenden. Diese Art von Abfrage führt häufig zu einer schlechten Leistung, da die Datenbank die Ergebnisse intern kompiliert und sie dann wieder in die Abfrage einfügt, um Daten zu rendern. Dieser Prozess ist ein zusätzlicher Schritt für die Datenbank. In vielen Fällen kann dieser Schritt zu einer schlechten Leistung in einem stark gleichzeitigen Ladezustand führen.

Überprüfen Sie auch, ob Ihre Abfragen eine große Anzahl von `ORDER BY`- und `GROUP BY`-Klauseln verwenden. Bei solchen Operationen muss die Datenbank oft zuerst den gesamten Datensatz im Speicher bilden. Dann muss es auf eine bestimmte Weise bestellen oder gruppieren, bevor es an den Kunden zurückgegeben wird.

Optimierung von Aurora MySQL mit proaktiven Einblicken von Amazon DevOps Guru

Die proaktiven Einblicke von DevOps Guru erkennen bekannte problematische Bedingungen auf Ihren Aurora-MySQL-DB-Clustern, bevor sie auftreten. DevOps Guru kann Folgendes tun:

- Vermeiden vieler häufig auftretender Datenbankprobleme durch Abgleich der Datenbankkonfiguration mit den allgemein empfohlenen Einstellungen.
- Warnen vor kritischen Problemen in der Flotte, die, wenn sie nicht überprüft werden, später zu größeren Problemen führen können.
- Benachrichtigung bei neu erkannten Problemen.

Jeder proaktive Einblick beinhaltet eine Analyse der Problemursache und Empfehlungen für Korrekturmaßnahmen.

Themen

- Die Länge der InnoDB-Verlaufsliste wurde deutlich erhöht
- Die Datenbank erstellt temporäre Tabellen auf der Festplatte

Die Länge der InnoDB-Verlaufsliste wurde deutlich erhöht

Ab dem *Datum* wurde Ihre Verlaufsliste für Zeilenänderungen erheblich erweitert, bis hin zur *Länge* auf der *DB-Instance*. Dieser Anstieg wirkt sich auf die Leistung beim Herunterfahren von Abfragen und Datenbanken aus.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Mögliche Ursachen für dieses Problem
- Aktionen
- Relevante Metriken

Unterstützte Engine-Versionen

Diese Insight-Informationen werden für alle Versionen von unterstützt.

Kontext

Das InnoDB-Transaktionssystem behält die Multiversion Concurrency Control (MVCC) bei. Wenn eine Zeile geändert wird, wird die Version der geänderten Daten vor der Änderung als Undo-Datensatz in einem Undo-Protokoll gespeichert. Jeder Undo-Datensatz hat einen Verweis auf seinen vorherigen Redo-Datensatz, wodurch eine verknüpfte Liste entsteht.

Die InnoDB-Verlaufsliste ist eine globale Liste der Undo-Protokolle für übernommene Transaktionen. MySQL verwendet die Verlaufsliste, um Datensätze und Protokollseiten zu löschen, wenn Transaktionen den Verlauf nicht mehr benötigen. Die Länge der Verlaufsliste ist die Gesamtzahl der Undo-Protokolle, die Änderungen in der Verlaufsliste enthalten. Jedes Protokoll umfasst eine oder mehrere Änderungen. Wenn die Länge der InnoDB-Verlaufsliste zu groß wird, was auf eine große Anzahl alter Zeilenversionen hinweist, werden Abfragen und Datenbankabschaltungen langsamer.

Mögliche Ursachen für dieses Problem

Zu den typischen Ursachen einer langen Verlaufsliste gehören die folgenden:

- Transaktionen mit langer Laufzeit, entweder beim Lesen oder Schreiben
- Eine hohe Schreiblast

Aktionen

Abhängig von den Ursachen Ihres Einblicks empfehlen wir verschiedene Aktionen.

Themen

- Beginnen Sie keine Operation, die ein Herunterfahren der Datenbank beinhaltet, bis die InnoDB-Verlaufsliste kürzer wird
- Identifizieren und beenden Sie lang andauernde Transaktionen
- Verwenden Sie Performance Insights, um die Top-Hosts und Top-Benutzer zu identifizieren.

Beginnen Sie keine Operation, die ein Herunterfahren der Datenbank beinhaltet, bis die InnoDB-Verlaufsliste kürzer wird

Da eine lange InnoDB-Verlaufsliste das Herunterfahren von Datenbanken verlangsamt, sollten Sie die Listengröße reduzieren, bevor Sie Operationen einleiten, die ein Herunterfahren der Datenbank beinhalten. Zu diesen Vorgängen gehören Datenbank-Upgrades der Hauptversionen.

Identifizieren und beenden Sie lang andauernde Transaktionen

Sie können Transaktionen mit langer Laufzeit finden, indem Sie `information_schema.innodb_trx` abfragen.

> **Note**
>
> Achten Sie auch darauf, auf Lesereplikaten nach Transaktionen mit langer Laufzeit zu suchen.

So identifizieren und beenden Sie lang andauernde Transaktionen

1. Führen Sie das folgende Skript in Ihrem SQL-Client aus:

```
SELECT a.trx_id,
      a.trx_state,
      a.trx_started,
      TIMESTAMPDIFF(SECOND,a.trx_started, now()) as "Seconds Transaction Has Been
 Open",
      a.trx_rows_modified,
      b.USER,
```

```
      b.host,
      b.db,
      b.command,
      b.time,
      b.state
FROM  information_schema.innodb_trx a,
      information_schema.processlist b
WHERE a.trx_mysql_thread_id=b.id
  AND TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 10
ORDER BY trx_started
```

2. Beenden Sie jede lang laufende Transaktion mit dem gespeicherten Verfahren mysql.rds_kill.

Verwenden Sie Performance Insights, um die Top-Hosts und Top-Benutzer zu identifizieren.

Optimieren Sie Transaktionen, so dass eine große Anzahl geänderter Zeilen sofort bestätigt wird.

Relevante Metriken

Die folgenden Metriken beziehen sich auf diesen Einblick:

- `trx_rseg_history_len` – Diese Zählermetrik kann sowohl in Performance Insights als auch in der Tabelle `INFORMATION_SCHEMA.INNODB_METRICS` angezeigt werden. Weitere Informationen finden Sie in der Metriktabelle „InnoDB INFORMATION_SCHEMA in der MySQL-Dokumentation.
- `RollbackSegmentHistoryListLength` – Diese Amazon CloudWatch-Metrik misst die Undo-Protokolle, die festgeschriebene Transaktionen mit zum Löschen markierten Datensätzen aufzeichnen. Diese Datensätze sind für die Verarbeitung durch den InnoDB-Löschvorgang geplant. Die Metrik `trx_rseg_history_len` hat denselben Wert wie `RollbackSegmentHistoryListLength`.
- `PurgeBoundary` – Die Transaktionsnummer, bis zu der eine InnoDB-Bereinigung zulässig ist. Wenn sich diese CloudWatch-Metrik über einen längeren Zeitraum nicht erhöht, ist dies ein gutes Anzeichen dafür, dass die InnoDB-Bereinigung durch Transaktionen mit langer Laufzeit blockiert wird. Zur Untersuchung überprüfen Sie die aktiven Transaktionen auf Ihrem Aurora-MySQL-DB-Cluster. Dieser Parameter wird für Aurora-MySQL-Version 2.11 und höher sowie für Version 2.04.5 und höher unterstützt.
- `PurgeFinishedPoint` – Die Transaktionsnummer, bis zu der eine InnoDB-Bereinigung ausgeführt wird. Mithilfe dieser CloudWatch-Metrik können Sie untersuchen, wie schnell die InnoDB-Bereinigung voranschreitet. Dieser Parameter wird für Aurora-MySQL-Version 2.11 und höher sowie für Version 2.04.5 und höher unterstützt.

- `TransactionAgeMaximum` – Das Alter der ältesten aktiven ausgeführten Transaktion. Diese CloudWatch-Metrik ist nur für Aurora-MySQL-Version 3.08 verfügbar.
- `TruncateFinishedPoint` – Die Transaktions-ID, bis zu der die Kürzung rückgängig gemacht wird. Diese CloudWatch-Metrik wird für Aurora-MySQL-Version 2.11 und höher sowie für Version 3.08 und höher unterstützt.

Weitere Informationen zu CloudWatch-Metriken finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

Die Datenbank erstellt temporäre Tabellen auf der Festplatte

Ihre aktuelle Nutzung temporärer Tabellen auf der Festplatte ist erheblich gestiegen, und zwar bis zu *`Prozentsatz`*. Die Datenbank erstellt ungefähr die *`Anzahl`* temporärer Tabellen pro Sekunde. Dies kann die Leistung beeinträchtigen und die Festplattenoperationen auf der *`DB-Instance`* erhöhen.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Mögliche Ursachen für dieses Problem
- Aktionen
- Relevante Metriken

Unterstützte Engine-Versionen

Diese Insight-Informationen werden für alle Versionen von unterstützt.

Kontext

Manchmal ist es notwendig, dass der MySQL-Server während der Verarbeitung einer Abfrage eine interne temporäre Tabelle erstellt. Aurora MySQL kann eine interne temporäre Tabelle im Speicher speichern, wo sie von der TempTable- oder MEMORY-Speicherengine verarbeitet oder von InnoDB auf der Festplatte gespeichert werden kann. Weitere Informationen finden Sie unter Interne Verwendung temporärer Tabellen in MySQL im MySQL-Referenzhandbuch.

Mögliche Ursachen für dieses Problem

Eine Zunahme temporärer Tabellen auf der Festplatte weist auf die Verwendung komplexer Abfragen hin. Wenn der konfigurierte Speicher nicht ausreicht, um temporäre Tabellen im Speicher

zu speichern, erstellt Aurora MySQL die Tabellen auf der Festplatte. Dies kann die Leistung beeinträchtigen und den Festplattenbetrieb erhöhen.

Aktionen

Abhängig von den Ursachen Ihres Einblicks empfehlen wir verschiedene Aktionen.

- Für Aurora-MySQL-Version 3 empfehlen wir die Verwendung der TempTable-Speicher-Engine.
- Optimieren Sie Ihre Abfragen, um weniger Daten zurückzugeben, indem Sie nur die erforderlichen Spalten auswählen.

  Wenn Sie das Performance-Schema aktivieren und alle `statement`-Instrumente aktiviert und zeitgesteuert sind, können Sie `SYS.statements_with_temp_tables` abfragen, um die Liste der Abfragen abzurufen, die temporäre Tabellen verwenden. Weitere Informationen finden Sie unter Voraussetzungen für die Verwendung des sys–Schemas in der MySQL-Dokumentation.
- Erwägen Sie die Indizierung von Spalten, die an Sortier- und Gruppierungsoperationen beteiligt sind.
- Schreiben Sie Ihre Abfragen neu, um `BLOB`- und `TEXT`-Spalten zu vermeiden. Diese Spalten verwenden immer die Festplatte.
- Optimieren Sie die folgenden Datenbankparameter: `tmp_table_size` und `max_heap_table_size`.

  Der Standardwert für diese Parameter ist 16 MiB. Wenn Sie die MEMORY-Speicher-Engine für temporäre In-Memory-Tabellen verwenden, wird deren maximale Größe durch den `tmp_table_size`- oder `max_heap_table_size`-Wert definiert, je nachdem, welcher Wert kleiner ist. Wenn diese maximale Größe erreicht ist, konvertiert MySQL die interne temporäre In-Memory-Tabelle automatisch in eine interne temporäre InnoDB-Tabelle auf der Festplatte. Weitere Informationen finden Sie unter Verwenden der TempTable-Speicher-Engine Amazon RDS für MySQL und Amazon Aurora MySQL.

  > **Note**
  >
  > Beim expliziten Erstellen von MEMORY-Tabellen mit CREATE TABLE bestimmt nur die `max_heap_table_size`-Variable, wie groß eine Tabelle werden kann. Es erfolgt auch keine Konvertierung in ein Festplattenformat.

Relevante Metriken

Die folgenden Performance Insights-Metriken beziehen sich auf diesen Einblick:

- Created_tmp_disk_tables
- Created_tmp_tables

Weitere Informationen finden Sie unter Created_tmp_disk_tables in der MySQL-Dokumentation.

Parallel Query für Amazon Aurora MySQL

Dieses Thema beschreibt, wie Sie Amazon-Aurora-MySQL-kompatible Edition Parallel Query optimal nutzen. Diese Funktion verwendet für bestimmte datenintensive Abfragen einen speziellen Verarbeitungspfad. Dabei kommt zum Tragen, dass die Architektur von Aurora verschiedene Speicherquellen nutzt. Parallel Query ist am wirkungsvollsten in Verbindung mit Aurora MySQL DB-Clustern, deren Tabellen Millionen Zeilen und analytische Abfragen unterstützen, deren Abarbeitung Minuten oder Stunden dauert.

Themen

- Übersicht über Parallel Query für Aurora MySQL
- Erstellen eines Parallel-Query-DB-Clusters in Aurora MySQL
- Aktivieren und Deaktivieren von Parallel Query in Aurora MySQL
- Optimieren von Parallel Query in MySQL
- Überprüfen, welche Anweisungen Parallel Query für Aurora MySQL verwenden
- Parallel Query für Aurora MySQL überwachen
- SQL-Konstrukte für Parallel Query in Aurora MySQL

Übersicht über Parallel Query für Aurora MySQL

Aurora MySQL Parallel Query ist eine Optimierung, die bei der Verarbeitung datenintensiver Abfragen einen Teil der I/O-Operationen und Berechnungen parallelisiert. Die parallelisierten Vorgänge beinhalten das Abrufen von Zeilen aus dem Speicher, das Extrahieren von Spaltenwerten und das Bestimmen der Zeilen, die den Bedingungen der `WHERE`-Klausel und der JOIN-Klauseln entsprechen. Diese datenintensive Arbeit wird auf der Ebene des verteilten Speichers von Aurora an mehrere Knoten in der verteilten Speicherschicht delegiert (aus Datenbanksicht herabgestuft). Ohne eine Parallelabfrage leitet jede Abfrage die gescannten Daten zu einem einzigen Knoten innerhalb des Aurora MySQL-Clusters (Hauptknoten) und die Verarbeitung jeglicher Abfragen erfolgt dort.

> **Tip**
>
> Die PostgreSQL-Datenbank-Engine hat auch eine Funktion, die auch „parallel query bzw. Paralleabfragen“ genannt wird. Diese Funktion steht in keinem Zusammenhang mit der parallelen Abfrage von Aurora.

Wenn die Funktion für parallele Abfragen aktiviert ist, bestimmt die Aurora-MySQL-Engine automatisch, wann Abfragen von Nutzen sein können, ohne dass SQL-Änderungen wie Hinweise oder Tabellenattribute erforderlich sind. In den folgenden Abschnitten lesen Sie, in welchen Fällen Abfragen parallelisiert werden. Dabei erfahren Sie auch, wie Parallelabfragen nur dann eingesetzt werden, wenn sie den größten Nutzen bringen.

> **Note**
>
> Die Funktion für die Optimierung der parallelen Abfrageausführung ist besonders bei Abfragen sinnvoll, die mehrere Minuten oder Stunden in Anspruch nehmen. Aurora MySQL führt in der Regel keine parallele Abfrageoptimierung für kostengünstige Abfragen durch. Es führt auch im Allgemeinen keine parallele Abfrageoptimierung durch, wenn ein anderes Optimierungsverfahren geeigneter wäre (z. B. Abfrage-Caching, Caching im Bufferpool oder Index-Lookups). Wenn Sie der Meinung sind, dass die parallele Abfrageausführung anders als erwartet angewendet wird, lesen Sie den Abschnitt Überprüfen, welche Anweisungen Parallel Query für Aurora MySQL verwenden.

Themen

- Vorteile
- Architektur
- Voraussetzungen
- Einschränkungen
- I/O-Kosten bei Parallelabfragen

Vorteile

Mit parallelen Abfragen können Sie datenintensive analytische Abfragen für Aurora MySQL-Tabellen ausführen. In vielen Fällen verbessert sich die Leistung im Vergleich mit herkömmlichen Verfahren, in denen Abfragen arbeitsteilig verarbeitet werden, um mehrere Größenordnungen.

Vorteile der parallelen Abfrageausführung:

- Höhere I/O-Leistung, weil physische Leseanforderungen auf mehrere Speicherknoten parallelisiert werden.
- Reduzierter Netzwerkverkehr. Aurora überträgt nicht komplette Datenseiten von den Speicherknoten zum Hauptknoten, um dort überflüssige Zeilen und Spalten herauszufiltern. Aurora

überträgt stattdessen kompakte Tupel, die nur die Spaltenwerte enthalten, die für den Ergebnissatz erforderlich sind.

- Niedrigere CPU-Last im Hauptknoten, weil die Funktionsverarbeitung, Zeilenfilterung und Spaltenprojektion für die `WHERE`-Klausel herabgestuft werden.
- Weniger Speicherdruck im Bufferpool. Die von der parallelen Abfrage verarbeiteten Seiten werden dem Bufferpool nicht hinzugefügt. Dieser Ansatz reduziert die Wahrscheinlichkeit, dass ein datenintensiver Scan häufig verwendete Daten aus dem Bufferpool entfernt.
- Möglicherweise weniger Datenduplikate in der ETL-Pipeline (Extract, Transform, Load), weil langwierige analytische Abfragen auf Bestandsdaten ausgeführt werden können.

Architektur

Parallele Abfragen machen sich die maßgeblichen Architekturprinzipien von Aurora MySQL zunutze: Entkopplung zwischen Datenbank-Engine und Speichersubsystem und weniger Netzwerkdatenverkehr durch Optimierung von Kommunikationsprotokollen. Aurora MySQL verwendet diese Techniken, um schreibintensive Operationen wie Redo-Protokoll-Verarbeitung zu beschleunigen. Parallel Query wendet die gleichen Prinzipien auf Leseoperationen an.

> **Note**
>
> Die Architektur von Aurora MySQL Parallel Query ist anders als die ähnlich benannter Funktionen in anderen Datenbanksystemen. Aurora MySQL Parallel Query ist kein symmetrisches Multiprocessing (SMP) und ist daher nicht von der CPU-Kapazität des Datenbankservers abhängig. Die Parallelverarbeitung erfolgt auf der Speicherschicht, völlig losgelöst vom Aurora MySQL-Server, der als Abfragekoordinator fungiert.

Ohne eine Parallelabfrage beinhaltet die Verarbeitung einer Aurora-Abfrage standardmäßig die Übertragung von Rohdaten an einen Einzelknoten innerhalb des Aurora-Clusters (dem Hauptknoten). Aurora führt dann die gesamte weitere Verarbeitung für diese Abfrage in einem einzigen Thread auf diesem einzelnen Knoten durch. Parallel Query delegiert einen Großteil dieser I/O- und CPU-intensiven Arbeit an Knoten aus der Speicherschicht. Nur die kompakten Zeilen aus dem Ergebnissatz werden an den Hauptknoten zurück übertragen. Die Zeilen sind dann bereits gefiltert und die Spaltenwerte bereits extrahiert und umgewandelt. Der Leistungsvorteil ergibt sich aus dem reduzierten Netzwerkdatenverkehr, einer niedrigeren CPU-Last im Hauptknoten und der Parallelisierung der I/O-Vorgänge aller Speicherknoten. Wie viele E/A-Vorgänge, Filterungen und

Projektionen parallel ablaufen, ist unabhängig von der Anzahl der DB-Instances im Aurora-Cluster, der die Abfrage ausführt.

Voraussetzungen

Um alle Funktionen von Parallel Query verwenden zu können, wird ein DB-Cluster von Aurora MySQL benötigt, auf dem Version 2.09 und höher ausgeführt wird. Wenn Sie bereits einen Cluster haben, den Sie mit der parallelen Abfrage verwenden möchten, können Sie ihn auf eine kompatible Version aktualisieren und anschließend die parallele Abfrage einschalten. Stellen Sie in diesem Fall sicher, dass Sie das Upgrade-Verfahren unter Wichtige Punkte bei einem Upgrade für parallele Abfragen befolgen, da die Namen der Konfigurationseinstellungen und Standardwerte in diesen neueren Versionen unterschiedlich sind.

Die DB-Instances in Ihrem Cluster müssen die `db.r*`-Instance-Klassen verwenden.

Stellen Sie sicher, dass die Hash-Join-Optimierung für Ihren Cluster aktiviert ist. Um zu erfahren wie dies geht, vgl. Hash-Join für parallele Abfrage-Cluster aktivieren.

Zum Anpassen von Parametern wie `aurora_parallel_query` und `aurora_disable_hash_join` benötigen Sie eine benutzerdefinierte Parametergruppe, die Sie mit dem Cluster verwenden. Sie können diese Parameter für jede DB-Instance einzeln angeben, indem Sie eine DB-Parametergruppe verwenden. Es wird jedoch empfohlen, sie in einer DB-Cluster-Parametergruppe anzugeben. Auf diese Weise übernehmen alle DB-Instances in Ihrem Cluster die gleichen Einstellungen für diese Parameter.

Einschränkungen

Parallele Abfragen unterliegen folgenden Einschränkungen:

- Parallelabfragen werden mit der DB-Cluster-Speicherkonfiguration Aurora I/O-Optimized nicht unterstützt.
- Sie können eine parallele Abfrage nicht mit den Instance-Klassen db.t2 oder db.t3 verwenden. Diese Einschränkung gilt auch dann, wenn Sie eine Parallelabfrage mit dem SQL-Hinweis `aurora_pq_force` anfordern.
- Die parallele Abfrage gilt nicht für Tabellen, die das Zeilenformat `COMPRESSED` oder `REDUNDANT` verwenden. Verwenden Sie die Zeilenformate `COMPACT` oder `DYNAMIC` für Tabellen, die Sie mit der parallelen Abfrage verwenden möchten.
- Aurora verwendet einen kostenbasierten Algorithmus, um zu bestimmen, ob der parallele Abfragemechanismus für jede SQL-Anweisung verwendet werden soll. Die Verwendung

bestimmter SQL-Konstrukte in einer Anweisung kann eine parallele Abfrage verhindern oder eine parallele Abfrage für diese Anweisung unwahrscheinlich machen. Hinweise zur Kompatibilität von SQL-Konstrukten mit parallelen Abfragen finden Sie unter SQL-Konstrukte für Parallel Query in Aurora MySQL.

- Jede Aurora-DB-Instance kann nur eine bestimmte Anzahl Parallelabfrage-Sitzungen gleichzeitig leisten. In Abfragen, die mehrere Komponenten mit paralleler Abfrageausführung enthalten (z. B. Unterabfragen, Join-Abfragen oder `UNION`-Operatoren), werden diese Phasen nacheinander ausgeführt. Die Anweisung zählt stets nur als einzelne Parallelabfrage-Sitzung. Wenn Sie überwachen möchten, wie viele Sitzungen derzeit geöffnet sind, verwenden Sie die Parallel-Query-Statusvariablen. Um herauszufinden, wie viele gleichzeitige Sitzungen eine DB-Instance maximal zulässt, fragen Sie die Statusvariable a `Aurora_pq_max_concurrent_requests`.
- Parallel Query ist in allen AWS-Regionen verfügbar, die Aurora unterstützt. Für die meisten AWS-Regionen ist für die Verwendung von Parallel Query mindestens die Aurora-MySQL-Version 2.09 erforderlich.
- Parallelabfragen wurden entwickelt, um die Leistung datenintensiver Abfragen zu verbessern. Diese Funktion ist nicht für einfache Abfragen konzipiert.
- Wir empfehlen, Leserknoten für SELECT-Anweisungen zu verwenden, insbesondere für datenintensive Anweisungen.

I/O-Kosten bei Parallelabfragen

Wenn Ihr Aurora MySQL Cluster eine parallele Abfrage verwendet, erfolgt möglicherweise eine Zunahme der `VolumeReadIOPS`-Werte. Parallel Query verwendet den Bufferpool nicht. Obwohl die Abfragen schnell sind, kann diese optimierte Verarbeitung zu einem Anstieg der Lesevorgänge und der damit verbundenen Gebühren führen.

Die I/O-Kosten für parallele Abfragen werden für Ihre Abfrage auf der Speicherebene gemessen und sind gleich oder höher, wenn die parallele Abfrage aktiviert ist. Ihr Vorteil ist die Verbesserung der Abfrageleistung. Es gibt zwei Gründe für potenziell höhere I/O-Kosten bei parallelen Abfragen:

- Selbst wenn sich einige Daten in einer Tabelle im Pufferpool befinden, müssen bei der parallelen Abfrage alle Daten auf der Speicherebene gescannt werden, was I/O-Kosten verursacht.
- Das Ausführen einer parallelen Abfrage führt nicht zu einem Aufwärmvorgang des Pufferpools. Folglich fallen bei aufeinanderfolgenden Läufen derselben parallelen Abfrage die vollen I/O-Kosten an.

Erstellen eines Parallel-Query-DB-Clusters in Aurora MySQL

Um einen für Parallel Query geeigneten Aurora MySQL-Cluster anzulegen, ihm neue Instances hinzuzufügen oder andere administrative Aufgaben zu erledigen, verwenden Sie die gleichen AWS-Managementkonsole- und AWS CLI-Techniken wie für andere Aurora-MySQL-Cluster. Sie können einen neuen Cluster erstellen, auf dem Sie parallele Abfragen ausführen können. Sie können auch einen für parallele Abfragen geeigneten DB-Cluster erstellen, indem Sie diesen aus einem Snapshot eines MySQL-kompatiblen Aurora-DB-Clusters wiederherstellen. Wenn Sie sich nicht sicher sind, wie ein neuer Aurora MySQL-Cluster erstellt wird, finden Sie unter diesem Link die erforderlichen Hintergrundinformationen und Anforderungen: Erstellen eines Amazon Aurora-DB Clusters.

Wenn Sie sich für eine Version einer Aurora MySQL-Engine entscheiden, empfehlen wir Ihnen, die neueste verfügbare Engine auszuwählen. Derzeit wird die Parallel-Query-Funktion von allen Versionen von Aurora MySQL unterstützt. Mit den neuesten Versionen haben Sie mehr Flexibilität beim Aktivieren/Deaktivieren der Parallel-Query-Funktion oder bei deren Verwendung für vorhandene Cluster.

Unabhängig davon, ob Sie einen neuen Cluster erstellen oder aus einem Snapshot wiederherstellen, gilt: Sie fügen neue DB-Instances genauso hinzu wie bei anderen Aurora MySQL-Clustern.

Sie können einen Parallel-Query-Cluster mit der Amazon-RDS-Konsole oder der AWS CLI erstellen.

Inhalt

- Erstellen eines Parallel Query-Clusters über die Konsole
- Erstellen eines Parallel Query-Clusters mit der CLI

Erstellen eines Parallel Query-Clusters über die Konsole

Sie können wie folgt über die Konsole einen neuen Parallelabfragecluster erstellen.

So erstellen Sie über die -Konsole einen Parallelabfragecluste AWS-Managementkonsole

1. Gehen Sie vor, wie im allgemeinen AWS-Managementkonsole-Verfahren auf der Seite Erstellen eines Amazon Aurora-DB Clusters beschrieben.
2. Wählen Sie für Engine-Typ die Option „Aurora MySQL“ aus.
3. Wählen Sie unter Zusätzliche Konfiguration eine Parametergruppe aus, die Sie für die DB-Cluster-Parametergruppe erstellt haben. Die Verwendung einer solchen benutzerdefinierten Parametergruppe ist für Aurora MySQL 2.09 und höher erforderlich. Geben Sie in Ihrer DB-

Cluster-Parametergruppe die Parametereinstellungen `aurora_parallel_query=ON` und `aurora_disable_hash_join=OFF` an. Dadurch wird die parallele Abfrage für den Cluster und die Hash-Join-Optimierung aktiviert, die in Kombination mit der parallelen Abfrage funktioniert.

So kontrollieren Sie, ob ein neuer Cluster parallelabfragefähig ist:

1. Erstellen Sie einen Cluster, wie vorhergehend beschrieben.
2. (Für Aurora MySQL Version 2 oder 3): Prüfen Sie, ob auf die Konfigurationseinstellung `aurora_parallel_query` die Bedingung „true“ zutrifft.

```
mysql> select @@aurora_parallel_query;
+-------------------------+
| @@aurora_parallel_query |
+-------------------------+
|                       1 |
+-------------------------+
```

3. (Für Aurora-MySQL-Version 2) Überprüfen Sie, ob die `aurora_disable_hash_join`-Einstellung „false“ ist.

```
mysql> select @@aurora_disable_hash_join;
+----------------------------+
| @@aurora_disable_hash_join |
+----------------------------+
|                          0 |
+----------------------------+
```

4. Überprüfen Sie bei einigen großen Tabellen und datenintensiven Abfragen die Abfragepläne, um zu bestätigen, dass einige Ihrer Abfragen die Optimierung für die parallele Abfrage verwenden. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Überprüfen, welche Anweisungen Parallel Query für Aurora MySQL verwenden.

Erstellen eines Parallel Query-Clusters mit der CLI

Sie können wie folgt mit der CLI einen neuen Parallelabfragecluster erstellen.

So erstellen Sie über die -Konsole einen Parallelabfragecluste AWS CLI

1. (Optional) Überprüfen Sie, welche Aurora MySQL-Versionen mit parallelen Abfrage-Clustern kompatibel sind. Verwenden Sie dazu den Befehl `describe-db-engine-versions` und

überprüfen Sie den Wert des Feldes `SupportsParallelQuery`. Ein Beispiel finden Sie unter Überprüfen der Versionskompatibilität von Aurora MySQL für Parallel Query.

2. (Optional) Erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe mit den Einstellungen `aurora_parallel_query=ON` und `aurora_disable_hash_join=OFF`. Verwenden Sie Befehle wie die folgenden.

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-
mysql8.0 --db-cluster-parameter-group-name pq-enabled-80-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-80-compatible \
  --parameters
 ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-
reboot
```

   Wenn Sie diesen Schritt ausführen, geben Sie die Option `--db-cluster-parameter-group-name` *`my_cluster_parameter_group`* in der nachfolgenden `create-db-cluster`-Anweisung an. Geben Sie den Namen Ihrer eigenen Parametergruppe an. Wenn Sie diesen Schritt auslassen, erstellen Sie die Parametergruppe und ordnen sie später dem Cluster zu, wie unter beschriebe Aktivieren und Deaktivieren von Parallel Query in Aurora MySQL.

3. Gehen Sie vor, wie im allgemeinen AWS CLI-Verfahren auf der Seite Erstellen eines Amazon Aurora-DB Clusters beschrieben.

4. Geben Sie die folgenden Optionen an:

   - Verwenden Sie für die Option `--engine` den Wert `aurora-mysql`. Diese Werte erzeugen parallele Abfrage-Cluster, die mit MySQL 5.7 oder 8.0 kompatibel sind.
   - Geben Sie für die Option `--db-cluster-parameter-group-name` den Namen einer DB-Cluster-Parametergruppe an, die Sie erstellt und für die Sie den Parameterwert `aurora_parallel_query=ON` zugewiesen haben. Wenn Sie diese Option auslassen, können Sie den Cluster mit einer Standardparametergruppe erstellen und ihn später so ändern, dass er eine solche benutzerdefinierte Parametergruppe verwendet.
   - Verwenden Sie für die Option `--engine-version` eine Aurora MySQL-Version, die mit der parallelen Abfrage kompatibel ist. Verwenden Sie das Verfahren aus Optimieren von Parallel Query in MySQL, um bei Bedarf eine Liste der Versionen abzurufen.

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode. Ersetzen Sie Ihren eigenen Wert für jede der Umgebungsvariablen wie *$CLUSTER_ID*. In diesem Beispiel wird auch die Option `--manage-master-user-password` zum Generieren des Hauptbenutzerpassworts und zum Verwalten dieses Passworts in Secrets Manager angegeben. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager. Alternativ können Sie die Option `--master-password` verwenden, um das Passwort selbst festzulegen und zu verwalten.

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID \
  --engine aurora-mysql --engine-version 8.0.mysql_aurora.3.04.1 \
  --master-username $MASTER_USER_ID --manage-master-user-password \
  --db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
  --engine same_value_as_in_create_cluster_command \
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. Überprüfen Sie, ob ein von Ihnen erstellter oder wiederhergestellter Cluster parallelabfragefähig ist.

   Prüfen Sie, ob auf die Konfigurationseinstellung `aurora_parallel_query` vorhanden ist. Wenn diese Einstellung den Wert „1“ hat, können Sie die parallele Abfrage verwenden. Wenn diese Einstellung den Wert „0“ hat, müssen Sie ihn auf „1“ festlegen, bevor Sie die parallele Abfrage verwenden können. In jedem Fall ist der Cluster in der Lage, die parallele Abfrage durchzuführen.

   ```
   mysql> select @@aurora_parallel_query;
   +------------------------+
   | @@aurora_parallel_query|
   +------------------------+
   |                      1 |
   +------------------------+
   ```

Stellen Sie einen Snapshot in einem parallelen Abfragecluster mit dem AWS CLI wie folgt wieder her:

1. Überprüfen Sie, welche Aurora MySQL-Versionen mit parallelen Abfrage-Clustern kompatibel sind. Verwenden Sie dazu den Befehl `describe-db-engine-versions` und überprüfen Sie den Wert des Feldes `SupportsParallelQuery`. Ein Beispiel finden Sie unter Überprüfen der

Versionskompatibilität von Aurora MySQL für Parallel Query. Entscheiden Sie, welche Version für den wiederhergestellten Cluster verwendet werden soll.

2. Suchen Sie einen Aurora MySQL-kompatiblen Cluster-Snapshot.
3. Gehen Sie vor, wie im allgemeinen AWS CLI-Verfahren auf der Seite Wiederherstellen aus einem DB-Cluster-Snapshot beschrieben.

```
aws rds restore-db-cluster-from-snapshot \
  --db-cluster-identifier mynewdbcluster \
  --snapshot-identifier mydbclustersnapshot \
  --engine aurora-mysql
```

4. Überprüfen Sie, ob ein von Ihnen erstellter oder wiederhergestellter Cluster parallelabfragefähig ist. Verwenden Sie das gleiche Verifizierungsverfahren wie in Erstellen eines Parallel Query-Clusters mit der CLI.

Aktivieren und Deaktivieren von Parallel Query in Aurora MySQL

Wenn die parallele Abfrage aktiviert ist, bestimmt Aurora MySQL, ob sie zur Laufzeit für jede Abfrage verwendet wird. Werden Join-Abfragen, Union-Abfragen, Unterabfragen usw. durchgeführt, entscheidet Aurora MySQL bei jedem Abfrageblock, ob zur Laufzeit parallel abgefragt werden soll. Details dazu finden Sie unter Überprüfen, welche Anweisungen Parallel Query für Aurora MySQL verwenden und SQL-Konstrukte für Parallel Query in Aurora MySQL.

Sie können parallele Abfragen sowohl auf globaler als auch auf Sitzungsebene für eine DB-Instance dynamisch ein- und ausschalten, indem Sie die Option aurora_parallel_query verwenden. Sie können die Einstellung `aurora_parallel_query` in Ihrer DB-Clustergruppe so ändern, dass die parallele Abfrage standardmäßig aktiviert oder deaktiviert wird.

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

Um de Parameter `aurora_parallel_query` auf Sitzungsebene zu aktivieren und zu deaktivieren, verwenden Sie die Standardmethoden zum Anpassen von Client-Konfigurationseinstellungen. Sie können dies beispielsweise über die `mysql`-Befehlszeile oder in einer JDBC- oder

ODBC-Anwendung tun. Der Befehl für den Standard-MySQL-Client lautet `set session aurora_parallel_query = {'ON'/'OFF'}`. Sie können den Parameter auf Sitzungsebene auch der JDBC-Konfiguration oder innerhalb Ihres Anwendungscodes hinzufügen, um parallele Abfragen dynamisch ein- oder auszuschalten.

Sie können die Einstellung für den Parameter `aurora_parallel_query` dauerhaft ändern, entweder für eine bestimmte DB-Instance oder für den gesamten Cluster. Wenn Sie den Parameterwert in einer DB-Parametergruppe angeben, gilt dieser Wert nur für bestimmte DB-Instances in Ihrem Cluster. Wenn Sie den Parameterwert in einer DB-Cluster-Parametergruppe angeben, erben alle DB-Instances im Cluster dieselbe Einstellung. Um den Parameter `aurora_parallel_query` umzuschalten, wenden Sie die Methoden an, die Sie auch für Parametergruppen verwenden, siehe Parametergruppen für Amazon Aurora. Dazu gehen Sie wie folgt vor:

1. Erstellen Sie eine benutzerdefinierte Cluster-Parametergruppe (empfohlen) oder eine benutzerdefinierte DB-Parametergruppe.
2. Aktualisieren Sie in dieser Parametergruppe `parallel_query` auf den gewünschten Wert.
3. Abhängig davon, ob Sie eine DB-Cluster-Parametergruppe oder eine DB-Parametergruppe erstellt haben, hängen Sie die Parametergruppe Ihrem Aurora-Cluster oder den spezifischen DB-Instances an, in denen Sie die Parallelabfrage-Funktion verwenden möchten.

> **Tip**
>
> Da es sich bei `aurora_parallel_query` um einen dynamischen Parameter handelt, ist ein Neustart des Clusters nach dem Ändern dieser Einstellung nicht erforderlich. Alle Verbindungen, die vor dem Umschalten der Option eine parallele Abfrage verwendeten, setzen diesen Vorgang fort, bis die Verbindung geschlossen oder die Instance neu gestartet wird.

Sie können den Parallelabfrageparameter nachträglich anpassen. Verwenden Sie dazu die API-Operationen ModifyDBClusterParameterGroup oder ModifyDBParameterGroup oder die AWS-Managementkonsole.

Sie können einen Hash-Join für Parallel-Query-Cluster aktivieren, Parallel Query mit der Amazon-RDS-Konsole oder der AWS CLI aktivieren oder deaktivieren und den Optimierer für Parallel Query überschreiben.

Inhalt

- Hash-Join für parallele Abfrage-Cluster aktivieren
- Ein- und Ausschalten der parallelen Abfrage mit der Konsole
- Ein- und Ausschalten der parallelen Abfrage mit der CLI
- Überschreiben des Parallelabfrageoptimierers

Hash-Join für parallele Abfrage-Cluster aktivieren

Parallele Abfragen werden typischerweise für ressourcenintensive Abfragen verwendet, die von der Hash-Join-Optimierung profitieren. Daher ist es hilfreich sicherzustellen, dass Hash-Joins für Cluster aktiviert sind, in denen Sie parallele Abfragen verwenden möchten. Informationen zur effektiven Verwendung von Hash-Joins finden Sie unter Optimierung von großen Aurora-MySQL-Join-Abfragen mit Hash-Joins.

Ein- und Ausschalten der parallelen Abfrage mit der Konsole

Sie können die parallele Abfrage auf DB-Instance-Ebene oder DB-Cluster-Ebene aktivieren oder deaktivieren, indem Sie mit Parametergruppen arbeiten.

So aktivieren oder deaktivieren Sie die parallele Abfrage für einen DB-Cluster mit der AWS-Managementkonsole

1. Erstellen Sie eine benutzerdefinierte Parametergruppe, wie in Parametergruppen für Amazon Aurora beschrieben.
2. Aktualisieren Sie aurora_parallel_query zu 1 (aktiviert) oder 0 (deaktiviert). Für Cluster, in denen die parallele Abfragefunktion verfügbar ist, ist aurora_parallel_query standardmäßig deaktiviert.
3. Wenn Sie eine benutzerdefinierte Cluster-Parametergruppe verwenden, fügen Sie sie dem Aurora-DB-Cluster zu, in dem Sie die parallele Abfrage verwenden möchten. Wenn Sie eine benutzerdefinierte DB-Parametergruppe verwenden, fügen Sie sie einer oder mehreren DB-Instances im Cluster zu. Wir empfehlen die Verwendung einer Cluster-Parametergruppe. Dadurch wird sichergestellt, dass alle DB-Instances im Cluster die gleichen Einstellungen für die parallele Abfrage und zugehörige Funktionen wie Hash-Join haben.

Ein- und Ausschalten der parallelen Abfrage mit der CLI

Sie können den Parallelabfrageparameter nachträglich anpassen. Verwenden Sie dazu den Befehl `modify-db-cluster-parameter-group` oder `modify-db-parameter-group`. Wählen Sie den

entsprechenden Befehl, je nachdem, ob Sie den Wert von `aurora_parallel_query` über eine DB-Cluster-Parametergruppe oder eine DB-Parametergruppe angeben.

So aktivieren oder deaktivieren Sie die parallele Abfrage für einen DB-Cluster mit der CLI

- Passen Sie den Parallelabfrageparameter nachträglich an. Verwenden Sie dazu den Befehl `modify-db-cluster-parameter-group`. Verwenden Sie einen Befehl wie den folgenden. Ersetzen Sie den entsprechenden Namen mit Ihrer eigenen benutzerdefinierten Parametergruppe. Setzen Sie entweder `ON` oder `OFF` ein in den `ParameterValue`-Teil der Option `--parameters`.

```
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "cluster_param_group_name"
}

aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot
```

Sie können parallele Abfragen auch auf Sitzungsebene aktivieren oder deaktivieren, beispielsweise über die Befehlszeile `mysql` oder in einer JDBC- oder ODBC-Anwendung. Verwenden Sie dafür die Standardmethoden zum Anpassen von Client-Konfigurationseinstellungen. Beispielsweise lautet der Befehl für den Standard-MySQL-Client `set session aurora_parallel_query = {'ON'/'OFF'}` für Aurora MySQL.

Sie können den Parameter auf Sitzungsebene auch der JDBC-Konfiguration oder innerhalb Ihres Anwendungscodes hinzufügen, um parallele Abfragen dynamisch ein- oder auszuschalten.

Überschreiben des Parallelabfrageoptimierers

Sie können die Sitzungsvariable `aurora_pq_force` verwenden, um den Parallelabfrageoptimierer zu überschreiben und für jede Abfrage eine Parallelabfrage anzufordern. Wir empfehlen diese Vorgehensweise nur zu Testzwecken. Das folgende Beispiel zeigt, wie Sie `aurora_pq_force` in einer Sitzung verwenden.

```
set SESSION aurora_parallel_query = ON;
set SESSION aurora_pq_force = ON;
```

Gehen Sie wie folgt vor, um die Überschreibung zu deaktivieren:

```
set SESSION aurora_pq_force = OFF;
```

Optimieren von Parallel Query in MySQL

Um Ihren DB-Cluster für Parallel Query zu optimieren, sollten Sie überlegen, welche DB-Cluster von dieser Funktion profitieren würden und ob Sie ein Upgrade auf diese Funktion durchführen sollten. Optimieren Sie dann Ihren Workload und erstellen Sie Schemaobjekte für Parallel Query.

Inhalt

- Planen eines Parallel Query-Clusters
  - Überprüfen der Versionskompatibilität von Aurora MySQL für Parallel Query
- Wichtige Punkte bei einem Upgrade für parallele Abfragen
  - Upgrade paralleler Abfrage-Cluster auf Aurora-MySQL-Version 3
  - Upgrade auf Aurora MySQL 2.09 und höher
- Optimierung der Abfrageleistung fürParallel Query
- Erstellen von Schema-Objekten, mit denen die Vorteile von Parallel Query genutzt werden können

Planen eines Parallel Query-Clusters

Bei der Planung eines DB-Clusters mit aktivierter paralleler Abfrage müssen einige Entscheidungen getroffen werden. Dazu gehören das Ausführen von Einrichtungsschritten (das Erstellen oder Wiederherstellen eines vollständigen Aurora MySQL-Clusters) und die Entscheidung, wie weit in Ihrem DB-Cluster die parallele Abfrage eingeschaltet werden sollen.

Berücksichtigen Sie bei der Planung Folgendes:

- Wenn Sie Aurora MySQL verwenden, das mit MySQL 5.7 kompatibel ist, müssen Sie einen bereitgestellten Cluster erstellen. Anschließend aktivieren Sie die parallele Abfrage mit dem Parameter `aurora_parallel_query`.

  Ist ein Aurora-MySQL-Cluster vorhanden, müssen Sie für die Verwendung von Parallel Query keinen neuen Cluster erstellen. Sie können Ihren Cluster oder bestimmte DB-Instances im Cluster

einer Parametergruppe zuordnen, bei der der Parameter `aurora_parallel_query` aktiviert ist. Dadurch können Sie die Zeit und den Aufwand für die Einrichtung der relevanten Daten für die parallele Abfrage reduzieren.

- Planen Sie alle großen Tabellen, die neu organisiert werden müssen, damit Sie die parallele Abfrage verwenden können, wenn Sie auf diese zugreifen. Möglicherweise müssen Sie neue Versionen einiger großer Tabellen erstellen, bei denen die parallele Abfrage nützlich ist. Beispielsweise müssen Sie möglicherweise Volltextsuchindizes entfernen. Details hierzu finden Sie unter Erstellen von Schema-Objekten, mit denen die Vorteile von Parallel Query genutzt werden können.

Überprüfen der Versionskompatibilität von Aurora MySQL für Parallel Query

Um zu überprüfen, welche Aurora-MySQL-Versionen mit parallelen Abfrage-Clustern kompatibel sind, verwenden Sie den AWS CLI-Befehl `describe-db-engine-versions` und überprüfen Sie den Wert des Feldes `SupportsParallelQuery`. Das folgende Codebeispiel zeigt, wie Sie überprüfen können, welche Kombinationen für Cluster für parallele Abfragen in einer bestimmten AWS-Region verfügbar sind. Stellen Sie sicher, dass Sie die vollständige `--query`-Parameterzeichenfolge in einer einzigen Zeile angeben.

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql \
--query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

Die vorhergehenden Befehle erzeugen eine Ausgabe, die der folgenden ähnelt: Die Ausgabe hängt davon ab, welche Aurora-MySQL-Versionen in der angegebenen AWS-Region verfügbar sind.

```
5.7.mysql_aurora.2.11.1
5.7.mysql_aurora.2.11.2
5.7.mysql_aurora.2.11.3
5.7.mysql_aurora.2.11.4
5.7.mysql_aurora.2.11.5
5.7.mysql_aurora.2.11.6
5.7.mysql_aurora.2.12.0
5.7.mysql_aurora.2.12.1
5.7.mysql_aurora.2.12.2
5.7.mysql_aurora.2.12.3
5.7.mysql_aurora.2.12.4
8.0.mysql_aurora.3.04.0
8.0.mysql_aurora.3.04.1
8.0.mysql_aurora.3.04.2
```

```
8.0.mysql_aurora.3.04.3
8.0.mysql_aurora.3.05.2
8.0.mysql_aurora.3.06.0
8.0.mysql_aurora.3.06.1
8.0.mysql_aurora.3.07.0
8.0.mysql_aurora.3.07.1
```

Nachdem Sie angefangen haben, die parallele Abfrage mit einem Cluster zu verwenden, können Sie die Leistung überwachen und Hindernisse bei der Nutzung der parallelen Abfrage beseitigen. Diese Anweisungen finden Sie unter Optimierung der Abfrageleistung fürParallel Query.

Wichtige Punkte bei einem Upgrade für parallele Abfragen

Abhängig von der Original- und Zielversionen beim Upgrade eines parallelen Abfrage-Clusters finden Sie möglicherweise Verbesserungen bei den Abfragetypen, die parallele Abfragen optimieren können. Möglicherweise müssen Sie auch feststellen, dass Sie keinen speziellen Engine-Modusparameter für parallele Abfragen angeben müssen. In den folgenden Abschnitten werden die Überlegungen beim Upgrade eines Clusters erläutert, bei dem die parallele Abfrage aktiviert ist.

Upgrade paralleler Abfrage-Cluster auf Aurora-MySQL-Version 3

Mehrere SQL-Anweisungen, Klauseln und Datentypen haben ab Aurora-MySQL-Version 3 neuen oder verbesserten Support für parallele Abfragen. Prüfen Sie beim Upgrade von einer Version, die vor Version 3 liegt, ob zusätzliche Abfragen von parallelen Abfrageoptimierungen profitieren können. Informationen zu diesen Erweiterungen für parallele Abfragen finden Sie unter Spaltendatentypen, Partitionierte Tabellen und Aggregationsfunktionen, GROUP BY-Klauseln und HAVING-Klauseln.

Wenn Sie einen Parallelabfragecluster von Aurora MySQL 2.08 oder niedriger aktualisieren, informieren Sie sich auch über Änderungen beim Einschalten der parallelen Abfrage. Lesen Sie dazu Upgrade auf Aurora MySQL 2.09 und höher.

In Aurora-MySQL-Version 3 ist die Hash-Join-Optimierung standardmäßig aktiviert. Die Konfigurationsoption `aurora_disable_hash_join` aus früheren Versionen wird nicht verwendet.

Upgrade auf Aurora MySQL 2.09 und höher

In Aurora MySQL 2.09 und höher funktioniert die parallele Abfrage für bereitgestellte Cluster und erfordert keinen `parallelquery`-Engine-Modus-Parameter. Daher müssen Sie keinen neuen Cluster erstellen oder aus einem vorhandenen Snapshot wiederherstellen, um die parallele Abfrage mit diesen Versionen verwenden zu können. Sie können die unter Upgrade der Nebenversion

oder der Patch-Ebene eines Aurora MySQL-DB-Clusters beschriebenen Upgrade-Verfahren verwenden, um den Cluster auf eine solche Version zu aktualisieren. Sie können einen älteren Cluster aktualisieren, unabhängig davon, ob es sich um einen Parallelabfragecluster oder um einen bereitgestellten Cluster handelte. Um die Anzahl der Optionen im Menü Engine-Version zu reduzieren, können Sie Versionen anzeigen, die die parallele Abfrage unterstützen auswählen, um die Einträge in diesem Menü zu filtern. Wählen Sie dann Aurora MySQL 2.09 oder höher aus.

Nachdem Sie einen früheren Parallelabfragecluster auf Aurora MySQL 2.09 oder höher aktualisiert haben, aktivieren Sie die parallele Abfrage im aktualisierten Cluster. Die parallele Abfrage ist in diesen Versionen standardmäßig deaktiviert, und die Vorgehensweise zum Aktivieren ist anders. Die Hash-Join-Optimierung ist ebenfalls standardmäßig deaktiviert und muss separat aktiviert werden. Stellen Sie daher sicher, dass Sie diese Einstellungen nach dem Upgrade wieder aktivieren. Anweisungen dazu finden Sie unter Aktivieren und Deaktivieren von Parallel Query in Aurora MySQL und Hash-Join für parallele Abfrage-Cluster aktivieren.

Insbesondere schalten Sie die parallele Abfrage mithilfe der Konfigurationsparameter `aurora_parallel_query=ON` und `aurora_disable_hash_join=OFF` anstelle von `aurora_pq_supported` und `aurora_pq` ein. Die Parameter `aurora_pq_supported` und `aurora_pq` sind in den neueren Aurora MySQL-Versionen veraltet.

Im aktualisierten Cluster weist das Attribut `EngineMode` den Wert `provisioned` anstelle von `parallelquery` auf. Um zu prüfen, ob die parallele Abfrage für eine angegebene Engine-Version verfügbar ist, überprüfen Sie nun den Wert des Feldes `SupportsParallelQuery` in der Ausgabe des `describe-db-engine-versions`-Befehls AWS CLI. In früheren Aurora MySQL-Versionen haben Sie überprüft, ob `parallelquery` in der Liste `SupportedEngineModes` vorhanden ist.

Nach dem Upgrade auf Aurora MySQL 2.09 oder höher können Sie die folgenden Funktionen nutzen. Diese Funktionen sind nicht für Parallelabfragecluster verfügbar, auf denen ältere Aurora MySQL-Versionen ausgeführt werden.

- Performance-Insights. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .
- Rückverfolgung. Weitere Informationen finden Sie unter Rückverfolgen eines Aurora-DB-Clusters.
- Stoppen und Starten des Clusters. Weitere Informationen finden Sie unter Stoppen und Starten eines Amazon Aurora-DB-Clusters.

Optimierung der Abfrageleistung fürParallel Query

Um mit Parallel Query die Abarbeitung einer Workload beeinflussen zu können, müssen Sie überlegen, wie Sie Parallel Query für die Abfragen einsetzen, bei denen diese Optimierung den größten Effekt hat.

Gehen Sie dazu wie folgt vor:

- Stellen Sie sicher, dass Ihre größten Tabellen mit der parallelen Abfrage kompatibel sind. Sie können Tabelleneigenschaften ändern oder einige Tabellen neu erstellen, damit Abfragen für diese Tabellen die Optimierung für die parallele Abfrage nutzen können. Um zu erfahren wie, siehe Erstellen von Schema-Objekten, mit denen die Vorteile von Parallel Query genutzt werden können.
- Überwachen Sie, welche Abfragen mit Parallel Query ausgeführt werden. Um zu erfahren wie, siehe Parallel Query für Aurora MySQL überwachen.
- Stellen Sie sicher, dass die parallele Abfrage für die datenintensivsten und am längsten laufenden Abfragen und mit der richtigen Parallelität für Ihren Workload verwendet wird. Um zu erfahren wie, siehe Überprüfen, welche Anweisungen Parallel Query für Aurora MySQL verwenden.
- Optimieren Sie Ihren SQL-Code, um die parallele Abfrage zu aktivieren, um sie auf die erwarteten Abfragen anzuwenden. Um zu erfahren wie, siehe SQL-Konstrukte für Parallel Query in Aurora MySQL.

Erstellen von Schema-Objekten, mit denen die Vorteile von Parallel Query genutzt werden können

Bevor Sie Tabellen erstellen oder ändern, die Sie für die parallele Abfrage verwenden möchten, sollten Sie sich mit den unter Voraussetzungen und Einschränkungen beschriebenen Anforderungen vertraut machen.

Wenn Sie Parallelabfragen starten, müssen Tabellen die Einstellungen `ROW_FORMAT=Compact` oder `ROW_FORMAT=Dynamic` verwenden. Öffnen Sie deshalb die Konfigurationseinstellungen von Aurora und prüfen Sie, ob an der Konfigurationsoption `INNODB_FILE_FORMAT` Änderungen vorgenommen wurden. Geben Sie die Anweisung `SHOW TABLE STATUS` aus, um das Zeilenformat aller Tabellen einer Datenbank zu bestätigen.

Bevor Sie Ihr Schema ändern, um die parallele Abfrage zu aktivieren, um mit mehr Tabellen zu arbeiten, stellen Sie sicher, dass Sie es testen. Ihre Tests sollten bestätigen, ob die parallele Abfrage zu einer besseren Nettoleistung für diese Tabellen führt. Stellen Sie außerdem sicher, dass die Schema-Anforderungen an Parallel Query mit Ihren Zielen vereinbar sind.

Zum Beispiel: Prüfen Sie vor der Umstellung von `ROW_FORMAT=Compressed` auf `ROW_FORMAT=Compact` oder `ROW_FORMAT=Dynamic` die Leistung von Workloads anhand der ursprünglichen und der neuen Tabellen. Berücksichtigen Sie außerdem eventuelle sonstige Effekte (z. B. erhöhtes Datenvolumen).

Überprüfen, welche Anweisungen Parallel Query für Aurora MySQL verwenden

Im normalen Betrieb müssen Sie nichts Besonderes unternehmen, um Parallel Query zu nutzen. Erfüllt eine Abfrage die grundlegenden Anforderungen einer Parallelabfrage, entscheidet der Abfrageoptimierer bei jeder Abfrage automatisch, ob Parallel Query verwendet werden soll.

Wenn parallele Abfragen in Versuchen in einer Entwicklungs- oder Testumgebung nicht verwendet werden, kann dies daran liegen, dass Ihre Tabellen zu klein sind (zu wenige Zeilen oder zu wenig Datenvolumen). Die Daten für die Tabelle können sich auch vollständig im Pufferpool befinden, insbesondere bei Tabellen, die Sie kürzlich erstellt haben, um Experimente durchzuführen.

Während Sie die Cluster-Leistung überwachen oder optimieren, müssen Sie entscheiden, ob parallele Abfragen in den richtigen Kontexten verwendet werden. Sie haben die Möglichkeit, das Datenbankschema, die Einstellungen, die SQL-Abfragen oder sogar die Cluster-Topologie und die Anwendungsverbindungseinstellungen nachzubessern, um die Vorteile der Funktion nutzen zu können.

Um zu prüfen, ob eine Abfrage Parallel Query verwendet, prüfen Sie den Abfrageausführungsplan (auch als „Erläuterungsplan“ bezeichnet), indem Sie die Anweisung EXPLAIN ausführen. Anhand unserer Beispiele können Sie nachvollziehen, wie sich SQL-Anweisungen, SQL-Klauseln und SQL-Ausdrücke auf die `EXPLAIN`-Ausgabe von Parallel Query auswirken: SQL-Konstrukte für Parallel Query in Aurora MySQL

Das nachfolgende Beispiel verdeutlicht den Unterschied zwischen einem normalen Abfrageplan und einem parallelen Abfrageplan (Parallel Query). Dieser Erläuterungsplan stammt aus Abfrage 3 aus dem TPC-H-Benchmark. Viele der Beispielabfragen aus diesem Abschnitt verwenden die Tabellen aus dem TPC-H-Dataset. Sie können die Tabellendefinitionen, Abfragen und das `dbgen`-Programm abrufen, das Beispieldaten von der TPC-h-Website generiert.

```
EXPLAIN SELECT l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) AS revenue,
  o_orderdate,
  o_shippriority
```

```
FROM customer,
  orders,
  lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
  o_orderdate,
  o_shippriority
ORDER BY revenue DESC,
  o_orderdate LIMIT 10;
```

Standardmäßig hat die Abfrage möglicherweise einen Plan wie den folgenden. Wenn im Abfrageplan kein Hash-Join verwendet wird, stellen Sie sicher, dass zuerst die Optimierung aktiviert ist.

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra                                              |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
|  1 | SIMPLE      | customer | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |  1480234 |    10.00 | Using where; Using temporary; Using filesort       |
|  1 | SIMPLE      | orders   | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 14875240 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | lineitem | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 59270573 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
```

Sie können bei Aurora MySQL 3 den Hash-Join auf Sitzungsebene aktivieren, indem Sie die folgende Anweisung ausgeben.

```
SET optimizer_switch='block_nested_loop=on';
```

Für Aurora MySQL Version 2.09 und höher setzen Sie den`aurora_disable_hash_join`-DB-Parameter oder DB-Cluster-Parameter auf `0` (aus). Durch Deaktivierung von `aurora_disable_hash_join` wird der Wert von `optimizer_switch` auf `hash_join=on` gesetzt.

Nachdem Sie den Hash-Join aktiviert haben, versuchen Sie erneut, die `EXPLAIN`-Anweisung auszuführen. Informationen zur effektiven Verwendung von Hash-Joins finden Sie unter Optimierung von großen Aurora-MySQL-Join-Abfragen mit Hash-Joins.

Bei aktiviertem Hash-Join, aber deaktivierter paralleler Abfrage kann die Abfrage einen Plan wie den folgenden haben, der Hash-Join, aber keine parallele Abfrage verwendet.

```
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
| id | select_type | table    |...| rows      | Extra
                          |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | customer |...|   5798330 | Using where; Using index; Using
 temporary; Using filesort       |
|  1 | SIMPLE      | orders   |...| 154545408 | Using where; Using join buffer (Hash
 Join Outer table orders)   |
|  1 | SIMPLE      | lineitem |...| 606119300 | Using where; Using join buffer (Hash
 Join Outer table lineitem) |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
```

Nachdem die parallele Abfrage aktiviert wurde, können zwei Schritte in diesem Abfrageplan die parallele Abfrageoptimierung verwenden, wie unter der Spalte `Extra` in der Ausgabe `EXPLAIN` angezeigt. Die I/O-intensive und CPU-intensive Verarbeitung aus diesen Schritten wird in die Speicherschicht hinabgestuft.

```
+----+...
+-------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                                    |
+----+...
+-------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using index; Using temporary; Using filesort
                                                    |
|  1 |...| Using where; Using join buffer (Hash Join Outer table orders); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra)   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table lineitem); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
```

```
+----+...
+-------------------------------------------------------------------------------------------------------
+
```

Hinweise zur Interpretierung der `EXPLAIN`-Ausgabe einer Parallelabfrage und der Teile von SQL-Anweisungen, für die Parallel Query in Frage kommt, lesen Sie im Beitrag SQL-Konstrukte für Parallel Query in Aurora MySQL.

Aus der nachfolgenden Beispielausgabe geht hervor, welche Ergebnisse zustande kommen, wenn die vorherige Abfrage auf einer db.r4.2xlarge-Instance mit kaltem Bufferpool ausgeführt wird. Die Abfrage wird mit Parallel Query erheblich schneller ausgeführt.

> **Note**
>
> Weil Durchlaufzeiten von vielen Umgebungsfaktoren abhängen, können Ihre Ergebnisse anders lauten. Führen Sie deshalb stets eigene Leistungstests durch. Damit prüfen Sie die Ergebnisse gegen Ihre eigene Umgebung, Workloads und andere Faktoren.

```
-- Without parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (1 min 49.91 sec)
```

In vielen Beispielabfragen aus diesem Abschnitt werden Tabellen aus diesem TPC-H-Dataset verwendet, besonders häufig die PART-Tabelle mit 20 Millionen Zeilen und folgender Definition.

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| p_partkey     | int(11)       | NO   | PRI | NULL    |       |
| p_name        | varchar(55)   | NO   |     | NULL    |       |
| p_mfgr        | char(25)      | NO   |     | NULL    |       |
| p_brand       | char(10)      | NO   |     | NULL    |       |
| p_type        | varchar(25)   | NO   |     | NULL    |       |
| p_size        | int(11)       | NO   |     | NULL    |       |
| p_container   | char(10)      | NO   |     | NULL    |       |
| p_retailprice | decimal(15,2) | NO   |     | NULL    |       |
| p_comment     | varchar(23)   | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

Durch Versuchsläufe mit Ihrer Workload gewinnen Sie ein Gefühl dafür, ob einzelne SQL-Anweisungen von Parallel Query profitieren. Anschließend können Sie mit den folgenden Überwachungstechniken nachprüfen, wie oft Parallel Query im Laufe der Zeit in echten Workloads verwendet wird. Für tatsächliche Workloads gelten zusätzliche Faktoren (z. B. Obergrenzen für gleichzeitige Abfragen).

Parallel Query für Aurora MySQL überwachen

Wenn Ihr Aurora MySQL Cluster Parallel Query verwendet, erfolgt möglicherweise eine Zunahme der `VolumeReadIOPS`-Werte. Parallel Query verwendet den Bufferpool nicht. Obwohl die Abfragen schnell sind, kann diese optimierte Verarbeitung zu einem Anstieg der Lesevorgänge und damit verbundenen Gebühren führen.

Neben den Amazon CloudWatch-Metriken (siehe Anzeigen von Metriken in der Amazon-RDS-Konsole) enthält Aurora auch verschiedene Statusvariable. Sie können diese globalen Statusvariablen verwenden, um die parallele Abfrageausführung zu überwachen. Sie können Ihnen Einblicke in die Gründe geben, warum der Optimierer die parallele Abfrage in einer bestimmten Situation verwendet oder nicht verwendet. Sie können diese Variablen mit dem Befehl `SHOW GLOBAL STATUS` abrufen. Eine Liste der Variablen finden Sie auch nachfolgend auf dieser Seite.

Eine parallele Abfragesitzung ist nicht notwendigerweise eine 1:1-Zuweisung mit den von der Datenbank durchgeführten Abfragen. Angenommen, Ihr Abfrageplan enthält zwei Schritte, die

Parallel Query verwenden. In diesem Fall besteht die Abfrage aus zwei parallelen Sitzungen. Die Zähler für Aufrufversuche und erfolgreiche Aufrufe werden inkrementell jeweils um zwei erhöht.

Wenn Sie mit Parallel Query experimentieren und dabei `EXPLAIN`-Anweisungen herausgeben, kann sein, dass Zähleranstiege als "nicht ausgewählt" bezeichnet sind, obwohl die Abfragen nicht ausgeführt werden. Wenn Sie Parallel Query in einer Produktionsumgebung einsetzen, können Sie nachprüfen, ob die Meldung "nicht ausgewählt" schneller als erwartet zunimmt. An diesem Punkt können Sie sie so anpassen, dass die parallele Abfrage für die erwarteten Abfragen ausgeführt wird. Dazu können Sie Ihre Clustereinstellungen, den Abfragemix, DB-Instances, bei denen die parallele Abfrage aktiviert ist, usw. ändern.

Die Aktionen werden auf DB-Instance-Ebene nachverfolgt. Wenn Sie eine Verbindung zu einem anderen Endpunkt herstellen, könnten sich andere Metriken ergeben, weil jede DB-Instance ihre eigenen Parallelabfragen ausführt. Unterschiedliche Metriken können sich auch dann ergeben, wenn sich der Leserendpunkt in jeder Sitzung mit einer anderen DB-Instance verbindet.

Name	Beschreibung
`Aurora_pq_bytes_returned`	Die Anzahl der Bytes, die in Zusammenh ang mit Tupel-Datenstrukturen während der Parallelabfragen an den Hauptknoten übertrage n wurden. Für den Abgleich mit durch 16 348 teile `Aurora_pq_pages_pushed_down` .
`Aurora_pq_max_concurrent_re quests`	Die Höchstanzahl der Parallel Query-Sit zungen, die auf dieser Aurora-DB-Instance gleichzeitig ausgeführt werden können. Diese Zahl ist festgelegt und hängt von der AWS-DB-Instance-Klasse ab.
`Aurora_pq_pages_pushed_down`	Die Anzahl der Datenseiten (jeweils fix 16 KiB groß), auf denen Parallel Query eine netzwerkg ebundene Übertragung an den Hauptknoten verhindert hat.
`Aurora_pq_request_attempted`	Die Anzahl der angeforderten Parallel Query-Sitzungen. Hinter diesem Wert können pro Abfrage mehrere Sitzungen stehen. Ausschlag

Name	Beschreibung
	gebend sind SQL-Konstrukte wie Unterabfragen und Join-Abfragen.
`Aurora_pq_request_executed`	Die Anzahl der erfolgreich ausgeführten Parallel Query-Sitzungen.
`Aurora_pq_request_failed`	Die Anzahl der Parallel Query-Sitzungen, die dem Client einen Fehler zurückgaben. Es kann vorkommen, dass eine angeforderte Parallelabfrage fehlschlägt – z. B. wegen eines Problems auf der Speicherschicht. In diesen Fällen wird der fehlgeschlagene Abfrageteil wiederholt. Dafür kommt der nicht-parallele Abfragemechanismus zum Einsatz. Wenn auch die wiederholte Abfrage fehlschlägt, wird dem Client ein Fehler zurückgegeben. Die Zähleranzeige erhöht sich.
`Aurora_pq_request_in_progress`	Die Anzahl der derzeit in Ausführung befindlichen Parallel Query-Sitzungen. Die Zahl bezieht sich auf die angeschlossene Aurora-DB-Instance, mit der Sie verbunden sind, nicht auf das gesamte Aurora-DB-Cluster. Sie können prüfen, ob eine DB-Instance die Obergrenze für gleichzeitige Abfragen annähernd erreicht hat. Gleichen Sie dazu diesen Wert mit a `Aurora_pq_max_concurrent_requests` .

Name	Beschreibung
`Aurora_pq_request_not_chosen`	Die Anzahl der Situationen, in denen Parallel Query nicht für die Abarbeitung einer Abfrage genutzt wurde. Dieser Wert setzt sich aus den Beiträgen mehrerer Zähler mit höherer Granularität zusammen. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chosen_below_min_rows`	Die Anzahl der Situationen, in denen die Anzahl der Tabellenzeilen der Grund für den Nichteinsatz von Parallel Query war. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chosen_column_bit`	Die Anzahl der Parallelabfrageanforderungen, die aufgrund eines nicht unterstützten Datentyps in der Liste der projizierten Spalten den nicht-parallelen Abfrageverarbeitungspfad nutzen.
`Aurora_pq_request_not_chosen_column_geometry`	Die Anzahl der Parallelabfrageanforderungen, die den nicht-parallelen Abfrageverarbeitungspfad nutzen, da die Tabelle Spalten mit dem Datentyp `GEOMETRY` enthält. Informationen zu Aurora-MySQL-Versionen, die diese Einschränkung aufheben, finden Sie unter Upgrade paralleler Abfrage-Cluster auf Aurora-MySQL-Version 3.

Name	Beschreibung
`Aurora_pq_request_not_chosen_column_lob`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle Spalten mit einem `LOB`-Datentyp enthält oder `VARCHAR`-Spalten, die aufgrund der deklarierten Länge extern gespeichert werden. Informationen zu Aurora-MySQL-Versionen, die diese Einschränkung aufheben, finden Sie unter Upgrade paralleler Abfrage-Cluster auf Aurora-MySQL-Version 3.
`Aurora_pq_request_not_chosen_column_virtual`	Die Anzahl der Parallelabfrageanforderungen, die den nicht-parallelen Abfrageverarbeitungspfad nutzen, da die Tabelle eine virtuelle Spalte enthält.
`Aurora_pq_request_not_chosen_custom_charset`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle Spalten mit einem benutzerdefinierten Zeichensatz enthält.
`Aurora_pq_request_not_chosen_fast_ddl`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle zurzeit durch eine schnelle `ALTER`-DDL-Anweisung geändert wird.
`Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool`	Die Anzahl der Situationen, in denen Parallel Query nicht genutzt wurde, obwohl weniger als 95 Prozent der Tabellendaten im Bufferpool waren. Grund: Es gab zu wenig nicht gepufferte Tabellendaten. Eine Parallelabfrage lohnte sich deshalb nicht.

Name	Beschreibung
`Aurora_pq_request_not_chose n_full_text_index`	Die Anzahl der parallelen Abfrageanforderung en, die den nicht parallelen Abfrageverarbeitun gspfad verwenden, da die Tabelle Volltexti ndizes enthält.
`Aurora_pq_request_not_chosen_ high_buffer_pool_pct`	Die Anzahl der Situationen, in denen Parallel Query nicht genutzt wurde, weil ein hoher Anteil der Tabellendaten (derzeit >95 Prozent) bereits im Bufferpool war. In diesen Fällen entscheidet der Optimierer, dass das Lesen der Daten aus dem Bufferpool effizienter ist. Die Zähleranz eige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chose n_index_hint`	Die Anzahl der parallelen Abfrageanforderung en, die den nicht parallelen Abfrageverarbeitun gspfad verwenden, da die Abfrage einen Indexhinweis enthält.
`Aurora_pq_request_not_chose n_innodb_table_format`	Die Anzahl der parallelen Abfrageanforderung en, die den nicht parallelen Abfrageverarbeitun gspfad verwenden, da die Tabelle ein nicht unterstütztes InnoDB-Zeilenformat verwendet . Die parallele Aurora-Abfrage gilt nur für die `COMPACT`-, `REDUNDANT` - und `DYNAMIC`-Zeilenformate.
`Aurora_pq_request_not_chose n_long_trx`	Die Anzahl der Parallelabfrageanforderunge n, die den nicht-parallelen Abfrageverarbeitun gspfad nutzten, weil die Abfrage in einer lang laufenden Transaktion gestartet wurde. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.

Name	Beschreibung
`Aurora_pq_request_not_chosen_no_where_clause`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage keine `WHERE`-Klausel enthält.
`Aurora_pq_request_not_chosen_range_scan`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage einen Bereichsscan für einen Index verwendet.
`Aurora_pq_request_not_chosen_row_length_too_long`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Gesamtlänge aller Spalten zu groß ist.
`Aurora_pq_request_not_chosen_small_table`	Die Anzahl der Situationen, in denen die Gesamtgröße der Tabelle (Zeilenanzahl und durchschnittliche Zeilenlänge) der Grund für den Nichteinsatz von Parallel Query war. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chosen_temporary_table`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage auf temporäre Tabellen verweist, die die nicht unterstützten Tabellentypen `MyISAM` oder `memory` verwenden.

Name	Beschreibung
`Aurora_pq_request_not_chosen_tx_isolation`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage eine nicht unterstützte Transaktionsisolationsstufe verwendet. Bei DB-Leser-Instances wird die parallele Abfrage nur auf die Isolationsstufen `REPEATABLE READ` und `READ COMMITTED` angewendet.
`Aurora_pq_request_not_chosen_update_delete_stmts`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage Teil einer `UPDATE`- oder `DELETE`-Anweisung ist.
`Aurora_pq_request_not_chosen_unsupported_access`	Die Anzahl der Parallelabfrageanforderungen, die den nicht-parallelen Abfrageverarbeitungspfad nutzen, weil die `WHERE`-Klausel nicht den Kriterien einer Parallelabfrage gerecht wird. Dieses Ergebnis kann eintreten, wenn für die Abfrage kein datenintensiver Scan erforderlich ist oder wenn es sich bei der Abfrage um eine `DELETE`- oder `UPDATE`-Anweisung handelt.
`Aurora_pq_request_not_chosen_unsupported_storage_type`	Die Anzahl der Parallelabfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da der DB-Cluster von Aurora MySQL keine unterstützte Aurora-Cluster-Speicherkonfiguration verwendet. Dieser Parameter ist in Aurora-MySQL-Version 3.04 und höher verfügbar. Weitere Informationen finden Sie unter Einschränkungen.

Name	Beschreibung
`Aurora_pq_request_throttled`	Die Anzahl der Situationen, in denen Parallel Query nicht genutzt wurde, weil die Höchstmenge gleichzeitiger Parallelabfragen bereits auf einer bestimmten Aurora-DB-Instance ausgeführt wird.

SQL-Konstrukte für Parallel Query in Aurora MySQL

Im folgenden Abschnitt finden Sie weitere Details dazu, warum bestimmte SQL-Anweisungen parallele Abfragen verwenden oder nicht verwenden. In diesem Abschnitt wird auch beschrieben, wie Aurora MySQL-Funktionen mit Parallel Query interagieren. Mit Hilfe dieser Informationen können Sie Leistungsprobleme in Clustern diagnostizieren, die Parallel Query verwenden. Außerdem können Sie nachvollziehen, wie Parallel Query auf Ihren spezifischen Workload angewendet wird.

Die Entscheidung für Parallel Query hängt von vielen Faktoren ab, die zu dem Zeitpunkt wirken, an dem die Anweisung ausgeführt wird. Es kann also sein, dass Parallel Query für bestimmte Abfragen immer, nie oder nur unter bestimmten Bedingungen zum Einsatz kommt.

> **Tip**
>
> Wenn Sie diese Beispiele in HTML anzeigen, können Sie das Widget Copy (Kopieren) in der oben rechts in jeder Codeverzeichnis verwenden, um den SQL-Code zu kopieren und es selbst zu versuchen. Durch die Verwendung des Widgets Copy (Kopieren) wird ein Kopieren der zusätzlichen Zeichen um die Aufforderung `mysql>` und die Fortsetzungszeilen `->` vermieden.

Themen

- EXPLAIN-Anweisung
- WHERE-Klausel
- Data Definition Language (DDL)
- Spaltendatentypen
- Partitionierte Tabellen
- Aggregationsfunktionen, GROUP BY-Klauseln und HAVING-Klauseln

- Funktionsaufrufe in WHERE-Klausel
- LIMIT-Klausel
- Vergleichsoperatoren
- Joins
- Unterabfragen
- UNION
- Ansichten
- DML-Anweisungen (Data Manipulation Language)
- Transaktionen und Sperren
- B-Baum-Indizes
- Volltextsuche (FTS)-Indizes
- Virtuelle Spalten
- Integrierte Caching-Mechanismen
- Optimierungshinweise
- Temporäre MyISAM-Tabellen

EXPLAIN-Anweisung

Wie aus den Beispielen aus diesem Abschnitt hervorgeht, gibt die `EXPLAIN`-Anweisung an, ob die jeweilige Stufe einer Abfrage derzeit parallelabfragetauglich ist. Darüber hinaus gibt sie auch an, welche Aspekte einer Abfrage auf die nächste Speicherschicht herabgestuft werden können. Die wichtigsten Bestandteile im Abfrageplan sind folgende:

- Wenn in der `NULL`-Spalte ein anderer Wert als `key` steht, liegt nahe, dass die Abfrage mit Index-Lookups effizient möglich ist. Der Einsatz von Parallel Query ist unwahrscheinlich.
- Wenn in der Spalte `rows` ein kleiner Wert steht (d. h. nicht im Millionenbereich), kann dies bedeuten, dass für die Abfrage zu wenig Daten verfügbar sind, als dass sich eine Parallelabfrage lohnen würde. Das bedeutet, dass eine parallele Abfrage unwahrscheinlich ist.
- Die Spalte `Extra` zeigt an, ob mit einer Parallelabfrage zu rechnen ist. Die Ausgabe kann in etwa wie im folgenden Beispiel aussehen.

  ```
  Using parallel query (A columns, B filters, C exprs; D extra)
  ```

  Die Zahl vor `columns` gibt an, wie viele Spalten im Abfrageblock hinterlegt sind.

Die Zahl vor `filters` gibt an, wie viele `WHERE`-Prädikate einen einfachen Vergleich zwischen Spaltenwert und einer Konstante darstellen. Es können Größen wie „Gleichheit“ und „Ungleichheit“, aber auch Bereiche verglichen werden. Aurora parallelisiert diese Prädikattypen am effektivsten.

Die Zahl vor `exprs` gibt die Anzahl der Ausdrücke (z. B. Funktionsaufrufe, Operatoren oder sonstige Ausdrücke) an, die auch parallelisiert werden können – wenn auch nicht so effektiv wie eine Filterbedingung.

Die Zahl vor `extra` gibt an, wie viele Ausdrücke nicht herabgestuft werden können und deshalb vom Hauptknoten ausgeführt werden.

Betrachten Sie dazu die folgende `EXPLAIN`-Ausgabe.

```
mysql> explain select p_name, p_mfgr from part
    -> where p_brand is not null
    -> and upper(p_type) is not null
    -> and round(p_retailprice) is not null;
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table |...| rows     | Extra
                                |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5
 columns, 1 filters, 2 exprs; 0 extra) |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
```

Aus der `Extra`-Spalte geht hervor, dass aus jeder Zeile fünf Spalten extrahiert werden, mit denen die Abfragebedingungen bewertet und der Ergebnissatz konstruiert werden. Ein `WHERE`-Prädikat beinhaltet einen Filter – in diesem Fall eine Spalte, die direkt in der `WHERE`-Klausel getestet wird. Kommen zwei `WHERE`-Klauseln vor, müssen kompliziertere Ausdrücke bewertet werden (die in diesem Fall Funktionsaufrufe beinhalten). Das Feld `0 extra` bestätigt, dass alle Operationen in der `WHERE`-Klausel im Zuge der Parallelabfrageverarbeitung in die Speicherschicht herabgestuft werden.

Wird Parallel Query nicht ausgewählt, lässt sich der Grund dafür meist aus den anderen Spalten der `EXPLAIN`-Ausgabe ableiten. So kann möglicherweise sein, dass der `rows`-Wert zu klein ist oder dass die Spalte `possible_keys` angibt, dass die Abfrage statt auf einen datenintensiven Scan auf einen Index-Lookup zurückgreift. Das folgende Beispiel zeigt eine Abfrage, bei der der Optimierer schätzen

kann, dass die Abfrage nur eine kleine Anzahl von Zeilen durchsucht. Dies erfolgt auf der Basis der Eigenschaften des Primärschlüssels. Eine Parallelabfrage ist deshalb nicht erforderlich.

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows |
 Extra                    |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
|  1 | SIMPLE      | part  | range | PRIMARY       | PRIMARY | 4       | NULL |   99 |
 Using where; Using index |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
```

Die Ausgabe mit Auskunft zur Verwendung von Parallel Query berücksichtigt alle Faktoren, die zu dem Zeitpunkt verfügbar waren, als die EXPLAIN-Anweisung ausgeführt wurde. Möglicherweise trifft der Optimierer zum Zeitpunkt der tatsächlichen Abfrageausführung eine andere Auswahl, wenn bis dahin eine andere Ausgangslage herrscht. EXPLAIN könnte beispielsweise melden, dass eine Anweisung Parallel Query verwendet. Wird die Abfrage später dann tatsächlich ausgeführt, könnte sein, dass Parallel Query aufgrund der dann vorliegenden Bedingungen nicht nötig ist. Zu solchen Bedingungen können mehrere weitere parallele Abfragen gehören, die gleichzeitig ausgeführt werden. Es kann sich auch um Zeilen handeln, die aus der Tabelle gelöscht werden, um einen neuen Index, der erstellt wird, um zu viel Zeit, die in einer offenen Transaktion verstreicht, usw.

WHERE-Klausel

Damit die parallele Abfrageausführung genutzt werden kann, muss die Abfrage eine WHERE-Klausel enthalten.

Die parallele Abfrageausführung beschleunigt viele unterschiedliche Ausdrücke, die in der WHERE-Klausel vorkommen:

- Einfache Abgleiche zwischen einem Spaltenwert und einer Konstante (Filter). Bei diesen Abgleichen lohnt sich die Herabstufung auf die Speicherschicht am meisten. Wie viele Filterausdrücke in einer Abfrage vorkommen, ist in der EXPLAIN-Ausgabe zusammengefasst.
- Wo dies möglich ist, werden auch andere Ausdrücke aus der WHERE-Klausel in die Speicherschicht hinabgestuft. Wie viele solche Ausdrücke in einer Abfrage vorkommen, ist in der EXPLAIN-Ausgabe zusammengefasst. Bei diesen Ausdrücken kann es sich um Funktionsaufrufe, LIKE-Operatoren, CASE-Ausdrücke und Ähnliches handeln.

- Bestimmte Funktionen und Operatoren werden von Parallel Query derzeit noch nicht herabgestuft. Wie viele solche Ausdrücke in einer Abfrage vorkommen, wird mit dem `extra`-Zähler in der `EXPLAIN`-Ausgabe angegeben. Der Rest der Abfrage verwendet Parallel Query.
- Ausdrücke aus der Auswahlliste werden nicht herabgestuft. Für Abfragen mit solchen Funktionen kann es sich aber positiv auswirken, dass die Zwischenergebnisse von Parallelabfragen weniger Netzwerkdatenverkehr verursachen. So können beispielsweise Abfragen, die Aggregationsfunktionen aus der Auswahlliste aufrufen, von Parallelabfragen profitieren, selbst wenn die Aggregationsfunktionen selbst nicht herabgestuft werden.

Die nachfolgend abgebildete Abfrage scannt beispielsweise die gesamte Tabelle und verarbeitet alle Werte aus der `P_BRAND`-Spalte. Es handelt sich dabei allerdings nicht um eine Parallelabfrage, weil sie keine `WHERE`-Klausel enthält.

```
mysql> explain select count(*), p_brand from part group by p_brand;
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows     |
 Extra                           |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
|  1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
 Using temporary; Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
```

Die nächste Abfrage enthält hingegen `WHERE`-Prädikate, die die Ergebnisse filtern. Deshalb ist eine Parallelabfrage möglich:

```
mysql> explain select count(*), p_brand from part where p_name is not null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
    -> group by p_brand;
+----+...+----------
+-------------------------------------------------------------------------------------------
+
| id |...| rows     | Extra
                                          |
+----+...+----------
+-------------------------------------------------------------------------------------------
+
```

```
|  1 |...| 20427936 | Using where; Using temporary; Using filesort; Using parallel
 query (5 columns, 1 filters, 2 exprs; 0 extra) |
+----+...+----------
+-----------------------------------------------------------------------------------------
+
```

Wenn der Optimierer davon ausgeht, dass zu einem Abfrageblock nur wenige Zeilen zurückgegeben werden, wird Parallel Query für diesen Abfrageblock nicht verwendet. Im nachfolgenden Beispiel wird ein Größer-als-Operator in der Primärschlüssel-Spalte auf mehrere Millionen Zeilen angewendet. Das sorgt dafür, dass Parallel Query verwendet wird. Dagegen wird davon ausgegangen, dass die Kleiner-als-Gegenprobe nur auf wenige Zeilen anwendbar ist. Parallel Query kommt nicht zum Einsatz.

```
mysql> explain select count(*) from part where p_partkey > 10;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+

mysql> explain select count(*) from part where p_partkey < 10;
+----+...+------+--------------------------+
| id |...| rows | Extra                    |
+----+...+------+--------------------------+
|  1 |...|    9 | Using where; Using index |
+----+...+------+--------------------------+
```

Data Definition Language (DDL)

In Aurora-MySQL-Version 2 ist die parallele Abfrage nur für Tabellen verfügbar, für die keine schnellen DDL-Vorgänge (Data Definition Language) anstehen. In Aurora-MySQL-Version 3 können Sie eine parallele Abfrage für eine Tabelle gleichzeitig mit einem sofortigen DDL-Vorgang verwenden.

Instant DDL in Aurora-MySQL-Version 3 ersetzt die schnelle DDL-Funktion in Aurora-MySQL-Version 2. Informationen zu Instant-DDL finden Sie unter Sofortige DDL (Aurora MySQL Version 3).

Spaltendatentypen

In Aurora-MySQL-Version 3 kann die parallele Abfrage mit Tabellen arbeiten, die Spalten mit den Datentypen `TEXT`, `BLOB`, `JSON` und `GEOMETRY` enthalten. Es kann auch mit `VARCHAR`- und `CHAR`-Spalten mit einer maximalen deklarierten Länge von mehr als 768 Byte arbeiten. Wenn sich Ihre Abfrage auf Spalten bezieht, die so große Objekttypen enthalten, erhöht die zusätzliche Arbeit zum Abrufen der Abfrageverarbeitung einen gewissen Aufwand. Überprüfen Sie in diesem Fall, ob die Abfrage die Verweise auf diese Spalten weglassen kann. Wenn dies nicht der Fall ist, führen Sie Benchmarks aus, um zu bestätigen, ob solche Abfragen schneller sind, wenn die parallele Abfrage aktiviert oder deaktiviert ist.

In Aurora-MySQL-Version 2 hat die parallele Abfrage folgende Einschränkungen für große Objekttypen:

- Die Datentypen `TEXT`, `BLOB`, `JSON` und `GEOMETRY` werden von parallelen Abfragen nicht unterstützt. Eine Abfrage, die sich auf Spalten dieses Typs bezieht, kann keine parallelen Abfragen nutzen.
- Spalten unterschiedlicher Länge (Datentypen `VARCHAR` und `CHAR`) sind bis zur deklarierten Länge (max. 768 Byte) mit Parallel Query kompatibel. Eine Abfrage, die sich auf Spalten der Typen bezieht, die mit mehr Höchstlänge deklariert sind, kann Parallel Query nicht nutzen. In Spalten, in denen Zeichensätze mit mehreren Byte Länge vorkommen, ist im Byte-Höchstwert die maximale Byte-Anzahl des Zeichensatzes berücksichtigt. Für eine `utf8mb4`-Spalte mit dem Zeichensatz `VARCHAR(192)` (maximal 4 Byte Zeichenlänge) kann beispielsweise eine Parallelabfrage gestartet werden, nicht jedoch für eine `VARCHAR(193)`-Spalte.

Partitionierte Tabellen

In Aurora-MySQL-Version 3 können Sie partitionierte Tabellen mit Parallel Query verwenden. Da partitionierte Tabellen intern als mehrere kleinere Tabellen dargestellt werden, verwendet eine Abfrage, die eine parallele Abfrage für eine nicht partitionierte Tabelle verwendet, möglicherweise keine parallele Abfrage für eine identische partitionierte Tabelle. Aurora MySQL prüft, ob jede Partition groß genug ist, um sich für die parallele Abfrageoptimierung zu qualifizieren, anstatt die Größe der gesamten Tabelle auszuwerten. Prüfen Sie, ob die Statusvariable `Aurora_pq_request_not_chosen_small_table` inkrementiert wird, wenn eine Abfrage in einer partitionierten Tabelle die parallele Abfrage nicht wie erwartet verwendet.

Betrachten Sie beispielsweise eine Tabelle, die mit `PARTITION BY HASH (`*`column`*`) PARTITIONS 2` partitioniert ist, und eine andere Tabelle, die mit `PARTITION BY HASH (`*`column`*`) PARTITIONS`

10 partitioniert ist. In der Tabelle mit zwei Partititionen sind die Partitionen fünfmal so groß wie die Tabelle mit zehn Partitionen. Daher wird eine parallele Abfrage eher für Abfragen gegen die Tabelle mit weniger Partitionen verwendet. Im folgenden Beispiel hat die Tabelle PART_BIG_PARTITIONS zwei Partitionen und PART_SMALL_PARTITIONS hat zehn Partitionen. Bei identischen Daten wird eine parallele Abfrage eher für die Tabelle mit weniger großen Partitionen verwendet.

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+--------------------+------------
+-------------------------------------------------------------------------------------------
+
| id | select_type | table              | partitions | Extra
                                                                                          |
+----+-------------+--------------------+------------
+-------------------------------------------------------------------------------------------
+
|  1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary;
 Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-bys, 1 aggrs) |
+----+-------------+--------------------+------------
+-------------------------------------------------------------------------------------------
+

mysql> explain select count(*), p_brand from part_small_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+----------------------+------------------------------
+------------------------------+
| id | select_type | table                | partitions                   | Extra
                  |
+----+-------------+----------------------+------------------------------
+------------------------------+
|  1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using
 where; Using temporary |
+----+-------------+----------------------+------------------------------
+------------------------------+
```

Aggregationsfunktionen, GROUP BY-Klauseln und HAVING-Klauseln

In Abfragen mit Aggregationsfunktionen werden große Tabellen mit sehr vielen Zeilen gescannt. Sie eignen sich deshalb oft besonders gut für Parallelabfragen.

In Aurora MySQL 3 kann eine parallele Abfrage Aggregatfunktionsaufrufe in der Auswahlliste und der Klausel `HAVING` optimieren.

Vor Aurora MySQL 3 werden Aggregatfunktionsaufrufe in der Auswahlliste oder der Klausel `HAVING` nicht auf die Speicherschicht übertragen. Eine parallele Abfrage kann dennoch die Leistung solcher Abfragen mit Aggregationsfunktionen verbessern – indem sie zuerst auf der Speicherschicht Spaltenwerte aus den Rohdatenseiten parallel extrahiert. Anschließend überträgt Parallel Query diese Werte in einem kompakten Tupelformat an den Hauptknoten zurück und nicht als vollständige Datenseiten. Wie immer muss die Abfrage mindestens 1 `WHERE`-Prädikat enthalten, das für Parallel Query aktiviert ist.

Die nachfolgenden einfach gehaltenen Beispiele veranschaulichen, welche aggregierte Abfragen von Parallel Query profitieren. Dies gilt zum einen durch Rückgabe unmittelbarer, kompakt gehaltener Ergebnisse an den Hauptknoten und eventuell zusätzlich durch Herausfilterung nicht passender Zeilen aus den Zwischenergebnissen.

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
 'Manufacturer#5';
+----+...+--------------------------------------------------------------------------+
| id |...| Extra                                                                    |
+----+...+--------------------------------------------------------------------------+
|  1 |...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+----+...+--------------------------------------------------------------------------+

mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
 p_mfgr having count(*) > 100;
+----+...
+-----------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-----------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (3
 columns, 0 filters, 1 exprs; 0 extra) |
```

```
+----+...
+-------------------------------------------------------------------------------------------
+
```

Funktionsaufrufe in WHERE-Klausel

Aurora kann die Optimierung der parallelen Abfrageausführung auf Aufrufe der meisten integrierten Funktionen der `WHERE`-Klausel anwenden. Durch die Parallelisierung dieser Funktionsaufrufe wird der Hauptknoten entlastet, weil etwas CPU-Arbeitslast entfällt. Die parallel ablaufende Bewertung der Prädikatfunktionen in der ersten Abfragephase ermöglicht es Aurora, die Datenmenge zu minimieren, die im späteren Verlauf übertragen und verarbeitet werden muss.

Aktuell werden nicht alle Funktionsaufrufe aus der Auswahlliste parallelisiert. Diese Funktionen werden vom Hauptknoten auch dann bewertet, wenn in der `WHERE`-Klausel dieselben Funktionsaufrufe enthalten sind. Die ursprünglichen Werte aus betroffenen Spalten werden in die Tupel aufgenommen, die vom Speicherknoten zum Hauptknoten zurückübertragen werden. Der Hauptknoten führt alle Transformationen aus, z. B. `UPPER`, `CONCATENATE` usw., und generiert die endgültigen Werte für den Abfragesatz.

Im folgenden Beispiel parallelisiert Parallel Query den Aufruf von `LOWER`, da diese Funktion in der `WHERE`-Klausel enthalten ist. Parallel Query wirkt sich nicht auf die Aufrufe von `SUBSTR` und `UPPER` aus, da sie in der Auswahlliste stehen.

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
    -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

Die gleichen Überlegungen gelten für andere Ausdrücke wie CASE-Ausdrücke oder LIKE-Operatoren. Im der folgenden Beispiel ist beispielsweise zu sehen, dass die parallele Abfrage in der CASE-Klausel den LIKE-Ausdruck und die WHERE-Operatoren evaluiert.

```
mysql> explain select p_mfgr, p_retailprice from part
    -> where p_retailprice > case p_mfgr
    ->   when 'Manufacturer#1' then 1000
    ->   when 'Manufacturer#2' then 1200
    ->   else 950
    -> end
    -> and p_name like '%vanilla%'
    -> group by p_retailprice;
+----+...
+-------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (4
 columns, 0 filters, 2 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------
+
```

LIMIT-Klausel

Derzeit können Abfrageblöcke mit LIMIT-Klausel nicht parallel abgefragt werden. Parallel Query könnte jedoch für frühere Abfragephasen mit GROUP BY, ORDER BY oder Join-Abfragen geeignet sein.

Vergleichsoperatoren

Der Optimierer schätzt ab, wie viele Zeilen gescannt werden müssen, um die Vergleichsoperatoren bewerten zu können, und entscheidet ausgehend von dieser Schätzung, ob Parallel Query verwendet wird.

Das erste Beispiel belegt, dass ein Gleichheitsvergleich gegen die Hauptschlüsselspalte effizient ohne Parallel Query möglich ist. Das zweite Beispiel belegt, dass für einen ähnlichen Vergleich gegen eine nicht indizierte Spalte mehrere Millionen Zeilen gescannt werden müssen. Parallel Query lohnt sich deshalb.

```
mysql> explain select * from part where p_partkey = 10;
+----+...+------+-------+
| id |...| rows | Extra |
+----+...+------+-------+
|  1 |...|    1 | NULL  |
+----+...+------+-------+

mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

Die gleichen Kriterien gelten für "ist ungleich"-Prüfungen und Bereichsvergleiche (z. B. kleiner als, größer als, ist gleich oder `BETWEEN`. Der Optimierer schätzt ab, wie viele Zeilen zu scannen sind, und entscheidet dann, ob sich eine parallele Abfrage angesichts des E/A-Gesamt-Volumens lohnt.

Joins

Join-Abfragen mit großen Tabellen sind in der Regel datenintensive Operationen, die von der Optimierung der parallelen Abfrageausführung profitieren. Derzeit werden Vergleiche von Spaltenwerten aus mehreren Tabellen (also die Join-Prädikate) nicht parallelisiert. Parallel Query kann allerdings einen Teil der internen Verarbeitung aus anderen Join-Phasen herunterstufen – z. B. die Erstellung des Bloom-Filters während eines Hash-Join. Parallel Query kann auch ohne `WHERE`-Klausel auf Join-Abfragen angewendet werden. Join-Abfragen sind in dieser Hinsicht Ausnahmen von der Regel, dass Parallelabfragen ohne `WHERE`-Klausel nicht möglich sind.

Jede Phase der Join-Verarbeitung wird ausgewertet, um festzustellen, ob sie für eine Parallelabfrage in Frage kommt. Falls mehrere Phasen parallelabfragetauglich sind, werden sie nacheinander ausgeführt. Was die Obergrenze für gleichzeitige Abfragen angeht, bedeutet dies, dass jede Join-Abfrage als separate Parallelabfragesitzung zählt.

Wenn eine Join-Abfrage beispielsweise mit `WHERE`-Prädikaten die Zeilen einer verknüpften Tabelle filtert, kann diese Filteroption Parallel Query verwenden. Ein weiteres Beispiel ist die Verknüpfung

einer großen mit einer kleinen Tabelle mittels Hash-Join in einer Join-Abfrage. In diesem Fall kann der Tabellenscan für die Generierung der Datenstruktur des Bloom-Filters Parallel Query möglicherweise verwenden.

> **Note**
>
> Parallele Abfragen werden typischerweise für ressourcenintensive Abfragen verwendet, die von der Hash-Join-Optimierung profitieren. Die Methode zum Einschalten der Hash-Join-Optimierung hängt von der Aurora-MySQL-Version ab. Einzelheiten zu den einzelnen Versionen finden Sie unter Hash-Join für parallele Abfrage-Cluster aktivieren. Informationen zur effektiven Verwendung von Hash-Joins finden Sie unter Optimierung von großen Aurora-MySQL-Join-Abfragen mit Hash-Joins.

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
| id |...| table    | type  | possible_keys | key         |...| rows      | Extra

              |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
|  1 |...| customer | index | PRIMARY       | c_nationkey |...|  15051972 | Using index

              |
|  1 |...| orders   | ALL   | o_custkey     | NULL        |...| 154545408 | Using join
 buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
```

In Join-Abfragen, die mit geschachtelten Schleifen arbeiten, könnte der äußerste geschachtelte Schleifenblock parallele Abfragen verwenden. Die Anwendung von parallelen Abfragen hängt von den gleichen Faktoren wie immer ab, z. B. dem Vorhandensein zusätzlicher Filterbedingungen in der `WHERE`-Klausel.

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
```

```
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and
 p_name is not null and ps_availqty > 0;
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table    |...| rows     | Extra
                                 |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part     |...| 20427936 | Using where; Using parallel query (2
 columns, 1 filters, 0 exprs; 0 extra) |
|  1 | SIMPLE      | partsupp |...| 78164450 | Using where; Using join buffer (Block
 Nested Loop)                       |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
```

Unterabfragen

Der äußere Abfrageblock und der innere Unterabfrageblock können jeweils parallele Abfragen verwenden oder nicht. Ob sie dies tun, ist für jeden Block von den gewöhnlichen Eigenschaften der Tabelle, der `WHERE`-Klausel usw. abhängig. In der folgenden Abfrage kommt Parallel Query am Unterabfrageblock zum Einsatz, nicht jedoch am Außenblock.

```
mysql> explain select count(*) from part where
   --> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type |...| rows     | Extra
                       |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY     |...|     NULL | Impossible WHERE noticed after reading const tables
                      |
|  2 | SUBQUERY    |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
```

Für korrelierte Unterabfragen steht die Optimierung der parallelen Abfrageausführung nicht zur Verfügung.

UNION

Jeder Abfrageblock einer UNION-Abfrage kann parallele Abfragen Query nutzen (oder nicht). Ausschlaggebend sind die Tabelleneigenschaften und die WHERE-Klausel usw. des jeweiligen UNION-Teils.

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
    -> union select p_partkey from part where p_name like '%vanil_a%';
+----+---------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type    |...| rows     | Extra
                          |
+----+---------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY        |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
|  2 | UNION          |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
| NULL | UNION RESULT | <union1,2> |...|     NULL | Using temporary
                       |
+----+--------------+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> Die UNION-Klauseln in der Abfrage werden nacheinander ausgeführt. Selbst wenn die Abfrage mehrstufig ist und in jeder Phase Parallel Query zum Einsatz kommt, führt sie Parallel Query jedes Mal separat aus. Deshalb gilt mit Hinblick auf die Höchstzahl gleichzeitig zulässiger Parallelabfragen auch eine komplexe mehrstufige Abfrage nur als 1 Abfrage.

Ansichten

Der Optimierer verwendet beim Umschreiben jeder Abfrage eine Ansicht als längere Abfrage. Dazu werden die zugrunde liegenden Tabellen verwendet. Das bedeutet, dass die Funktionsweise von Parallel Query unabhängig von der Tabellenreferenz (Ansicht oder tatsächliche Tabelle) gleich ist. Für die fertige umgeschriebene Abfrage gelten die gleichen Kriterien hinsichtlich der Verwendung von Parallel Query für eine Abfrage und hinsichtlich der Herabstufung einzelner Teile.

Beispielsweise zeigt der folgende Abfrageplan eine Ansichtsdefinition, die Parallel Query in der Regel nicht verwendet. Erst wenn die Ansicht mit zusätzlichen `WHERE`-Klauseln abgefragt wird, verwendet Aurora MySQL Parallel Query.

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs;
 1 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

DML-Anweisungen (Data Manipulation Language)

Die `INSERT`-Anweisung kann Parallel Query für die `SELECT`-Verarbeitungsphase verwenden, wenn der `SELECT`-Teil die sonstigen Bedingungen für eine Parallelabfrage erfüllt.

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr =
 'Manufacturer#1';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> In der Regel liegen die Daten zu den neu eingefügten Zeilen nach der `INSERT`-Anweisung im Bufferpool. Deshalb kann sein, dass eine Tabelle unmittelbar nach dem Einfügen vieler Zeilen nicht für Parallel Query in Frage kommt. Erst wenn die Daten im normalen Betrieb aus dem

Bufferpool entfernt wurden, können in Abfragen gegen die Tabelle wieder Parallelabfragen zum Einsatz kommen.

Die `CREATE TABLE AS SELECT`-Anweisung greift nicht auf Parallel Query zurück – auch dann nicht, wenn der `SELECT`-Teil der Anweisung ansonsten parallelabfragefähig ist. Diese Anweisung ist wegen ihres DDL-Anteils nicht für die Parallelabfrageverarbeitung geeignet. Dagegen kann der `INSERT ... SELECT`-Teil der `SELECT`-Anweisung Parallel Query verwenden.

Parallel Query wird nie verwendet, wenn `DELETE`- oder `UPDATE`-Anweisungen vorliegen. Dies gilt unabhängig von der Größe der Tabelle und der Prädikate aus der `WHERE`-Klausel.

```
mysql> explain delete from part where p_name is not null;
+----+-------------+...+----------+-------------+
| id | select_type |...| rows     | Extra       |
+----+-------------+...+----------+-------------+
|  1 | SIMPLE      |...| 20427936 | Using where |
+----+-------------+...+----------+-------------+
```

Transaktionen und Sperren

Sie können alle Isolationsebenen auf der primären Aurora-Instance verwenden.

Auf Aurora-Reader-DB-Instances gilt die parallele Abfrage für Anweisungen, die unter der `REPEATABLE READ`-Isolierungsstufe ausgeführt werden. Aurora MySQL-Versionen 2.09 oder höher können die `READ COMMITTED`-Isolierungsstufe auch auf Reader-DB-Instances verwenden. `REPEATABLE READ` ist die Standardisolierungsstufe für Aurora-Reader-DB-Instances. Um die Isolierungsstufe `READ COMMITTED` auf DB-Leser-Instances anzuwenden, muss die Konfigurationsoption `aurora_read_replica_read_committed` auf Sitzungsebene festgelegt werden. Die Isolierungsstufe `READ COMMITTED` für Reader-Instances entspricht dem SQL-Standardverhalten. Die Isolierung ist jedoch bei Reader-Instances weniger streng als bei Abfragen, welche die `READ COMMITTED`-Isolierungsstufe bei der Writer-Instance verwenden.

Weitere Informationen zu Aurora-Isolierungsstufen, insbesondere zu den Unterschieden in `READ COMMITTED` zwischen Writer- und Reader-Instances, finden Sie unter Aurora MySQL-Isolierungsstufen.

Nach Abschluss einer großen Transaktion kann es sein, dass die Tabellenstatistik veraltet ist. Für solche Statistiken ist möglicherweise eine `ANALYZE TABLE`-Anweisung erforderlich, damit Aurora die Zeilenzahl zuverlässig schätzen kann. Auch mit Hilfe einer umfangreichen DML-Anweisung könnte

ein beträchtlicher Teil der Tabellendaten in den Bufferpool gelangen. Sind diese Daten im Bufferpool kann sein, dass diese Tabelle weniger häufig von Parallel Query abgefragt wird. Dies ändert sich, wenn die Daten aus dem Pool entfernt sind.

Wenn Ihre Sitzung Teil einer langwierigen Transaktion (standardmäßig 10 Minuten) ist, verwenden weitere Abfragen aus dieser Sitzung keine Parallelabfragen. Eine einzelne lang laufende Abfrage kann auch wegen Zeitüberschreitung abgebrochen werden. Dieser Überschreitungsabbruch kann eintreten, wenn die Abfrage länger läuft als zulässig (Höchstdauer derzeit: 10 Minuten), bevor die Parallelverarbeitung der Abfragen beginnt.

Das Risiko, dass lang laufende Transaktionen ungewollt gestartet werden, kann reduziert werden. Legen Sie dazu in `autocommit=1`-Sitzungen, in denen Sie Ad-hoc-Abfragen durchführen, die Einstellung `mysql` fest. Selbst eine `SELECT`-Anweisung gegen eine Tabelle startet eine Transaktion, indem sie eine Leseansicht erstellt. Eine Leseansicht ist ein einheitlicher Datensatz für nachfolgende Abfragen. Dieser bleibt bestehen, bis die Transaktion übernommen wurde. Denken Sie an diese Einschränkung, wenn Aurora für JDBC- oder ODBC-Anwendungen verwendet wird. Solche Anwendungen könnten nämlich auch dann zur Ausführung kommen, wenn die Einstellung `autocommit` deaktiviert ist.

Im nachfolgenden Beispiel ist zu sehen, wie eine Abfrageausführung gegen eine Tabelle (Einstellung `autocommit` deaktiviert) eine Leseansicht erzeugt, die implizit eine Transaktion in Gang setzt. Abfragen, die kurz danach gestartet werden, können Parallel Query noch nutzen. Nach mehreren Minuten kommen Abfragen jedoch nicht mehr für Parallel Query in Frage. Wenn Sie ans Ende der Transaktion `COMMIT` oder `ROLLBACK` stellen, kann Parallel Query wieder ausgeführt werden.

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+

mysql> select sleep(720); explain select sql_no_cache count(*) from part where
 p_retailprice > 10.0;
```

```
+------------+
| sleep(720) |
+------------+
|          0 |
+------------+
1 row in set (12 min 0.00 sec)

+----+...+---------+-------------+
| id |...| rows    | Extra       |
+----+...+---------+-------------+
|  1 |...| 2976129 | Using where |
+----+...+---------+-------------+

mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+
```

Um festzustellen, wie oft Abfragen nicht für Parallel Query in Frage kamen, weil sie Teil lang laufender Transaktionen waren, untersuchen Sie die Statusvariable `Aurora_pq_request_not_chosen_long_trx`.

```
mysql> show global status like '%pq%trx%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Aurora_pq_request_not_chosen_long_trx | 4     |
+---------------------------------+-------+
```

`SELECT`-Anweisungen, die Sperren annehmen (z. B. Syntax `SELECT FOR UPDATE` oder `SELECT LOCK IN SHARE MODE`), können Parallel Query nicht verwenden.

Parallel Query kann an Tabellen funktionieren, die mit einer `LOCK TABLES`-Anweisung gesperrt sind.

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055';
+----+...+-----------
+-------------------------------------------------------------------------------+
| id |...| rows      | Extra
          |
+----+...+-----------
+-------------------------------------------------------------------------------+
|  1 |...| 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0
 exprs; 0 extra) |
+----+...+-----------
+-------------------------------------------------------------------------------+

mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055' for update;
+----+...+-----------+-------------+
| id |...| rows      | Extra       |
+----+...+-----------+-------------+
|  1 |...| 154545408 | Using where |
+----+...+-----------+-------------+
```

B-Baum-Indizes

Die von der `ANALYZE TABLE`-Anweisung erfassten Statistiken unterstützen den Optimierer bei der Entscheidung für Parallelabfragen oder Index-Lookups. Ausschlaggebend sind die Eigenschaften der Daten in den Spalten. Nach DML-Operationen können die Daten einer Tabelle nachhaltig verändert sein. Um die Statistiken auf dem aktuellen Stand zu halten, führen Sie nach der Operation `ANALYZE TABLE` aus.

Wenn Index-Lookups ausreichen, um eine Abfrage effizient ohne datenintensiven Scan auszuführen, verwendet Aurora möglicherweise Index-Lookups. Dadurch erübrigt sich der Zusatzaufwand der Parallelabfrageverarbeitung. Außerdem regeln Obergrenzen, wie viele Abfragen gleichzeitig in einem Aurora-DB-Cluster zulässig sind. Die Einhaltung bewährter Methoden bei der Indizierung von Tabellen trägt dazu bei, dass in den häufigsten und besonders oft gleichzeitig ausgeführten Abfragen Index-Lookups zur Anwendung kommen.

Volltextsuche (FTS)-Indizes

Zurzeit wird Parallel Query nicht für Tabellen mit Volltextsuchindex verwendet. Dies gilt unabhängig davon, ob sich die Abfrage auf solche indizierten Spalten bezieht oder ob sie den Operator `MATCH` verwendet.

Virtuelle Spalten

Zurzeit wird Parallel Query nicht für Tabellen verwendet, die eine virtuelle Spalte enthalten. Dies gilt unabhängig davon, ob die Abfrage auf virtuelle Spalten verweist.

Integrierte Caching-Mechanismen

Aurora enthält integrierte Caching-Mechanismen: den Bufferpool und den Abfrage-Cache. Der Aurora-Optimierer entscheidet, ob einer dieser Caching-Mechanismen oder Parallel Query für eine bestimmte Abfrage besser geeignet ist.

Wenn eine Parallelabfrage Zeilen filtert und Spaltenwerte transformiert/extrahiert, werden Daten als Tupel an den Hauptknoten zurückübertragen – nicht als Datenseiten. Das bedeutet also, dass bei Verwendung von Parallel Query dem Bufferpool keine Seiten hinzugefügt werden. Es werden auch keine Seiten entfernt, die bereits im Bufferpool sind.

Aurora überprüft die Anzahl der Tabellendatenseiten, die im Pufferpool vorhanden sind und welchen Anteil der Tabellendaten diese Zahl darstellt. Aurora verwendet diese Informationen, um zu bestimmen, ob es effizienter ist, parallele Abfragen zu verwenden (und die Daten im Pufferpool zu umgehen). Es ist auch möglich, dass Aurora den nicht-parallelen Abfrageverarbeitungspfad verwendet, der auf Daten aus dem Bufferpool zurückgreift. Welche Seiten im Cache abgelegt sind und wie sich datenintensive Abfragen auf das Caching und die Bereinigung auswirken, hängt von den Konfigurationseinstellungen des Bufferpools ab. Es kann deshalb schwierig vorherzusagen sein, ob eine Abfrage mit Parallel Query ausgeführt wird. Dies hängt von den Daten im Bufferpool ab, deren Zusammensetzung sich ständig ändert.

Außerdem begrenzt Aurora, wie viele Parallelabfragen gleichzeitig möglich sind. Parallel Query kommt nicht für jede Abfrage zum Einsatz. Deshalb befinden sich die Daten von Tabellen, die von mehreren Abfragen gleichzeitig genutzt werden, zu einem beträchtlichen Teil im Bufferpool. Dementsprechend nutzt Aurora diese Tabellen nicht oft für Parallelabfragen.

In einer Abfolge nicht-paralleler Abfragen gegen dieselbe Tabelle ist die erste Abfrage möglicherweise langsam, weil die Daten nicht im Bufferpool sind. Die zweite und nachfolgende Abfragen laufen bereits schneller ab, weil der Bufferpool inzwischen sozusagen "warmgelaufen" ist. In der Regel zeigen Parallelabfragen ab der ersten Tabellenabfrage gleichbleibende Leistung Für Leistungstests empfehlen sich Vergleichswerte für einen kalten und einen warmen Bufferpool. In einigen Fällen sind die Ergebnisse des warmen Bufferpools eine gute Vergleichsbasis für Parallelabfragezeiten. Berücksichtigen Sie in diesen Fällen Faktoren wie die Häufigkeit von Abfragen

für diese Tabelle. Überlegen Sie auch, ob es sich lohnt, die Daten für diese Tabelle im Pufferpool zu behalten.

Der Abfrage-Cache vermeidet die erneute Ausführung einer Abfrage, wenn eine identische Abfrage abgesendet wird und die zugrunde liegenden Tabellendaten nicht geändert wurden. Mit Parallel Query optimierte Abfragen können im Abfrage-Cache abgelegt werden. Wird die gleiche Abfrage noch einmal gestartet, liegt sofort ein Ergebnis vor.

> **Note**
>
> Bei Leistungsvergleichen können aufgrund des Abfrage-Cache künstlich niedrige Zeitangaben zustandekommen. Für Benchmark-ähnliche Aufgabenstellungen empfiehlt sich der `sql_no_cache`-Hinweis. Dieser verhindert, dass das Ergebnis aus dem Abfrage-Cache kommt. Auch dann nicht, wenn die gleiche Abfrage schon einmal ausgeführt wurde. Der Hinweis folgt in der Abfrage unmittelbar nach der `SELECT`-Anweisung. Viele Beispiele für parallele Abfragen in diesem Thema enthalten diesen Hinweis, um die Abfragezeiten zwischen den Versionen der Abfrage vergleichbar zu machen, für welche die parallele Abfrage aktiviert und deaktiviert ist.
> Entfernen Sie diesen Hinweis aus Ihrem Quellcode, bevor Sie parallele Abfragen in einer Produktionsumgebung verwenden.

Optimierungshinweise

Eine andere Möglichkeit, den Optimierer zu steuern, besteht in der Verwendung von Optimierungshinweisen, die in einzelnen Anweisungen angegeben werden können. Sie können beispielsweise eine Optimierung für eine Tabelle in einer Anweisung aktivieren und dann die Optimierung für eine andere Tabelle deaktivieren. Weitere Informationen zu diesen Hinweisen finden Sie unter Optimierungshinweise im MySQL-Referenzhandbuch.

Sie können SQL-Hinweise mit Aurora-MySQL-Abfragen verwenden, um die Leistung zu optimieren. Sie können auch Hinweise verwenden, um zu verhindern, dass Ausführungspläne für wichtige Abfragen aufgrund unvorhersehbarer Bedingungen geändert werden.

Wir haben die Funktion für SQL-Hinweise erweitert, damit Sie die Optimiererauswahl für Ihre Abfragepläne besser kontrollieren können. Diese Hinweise gelten für Abfragen, bei denen die Parallelabfrageoptimierung verwendet wird. Weitere Informationen finden Sie unter Aurora-MySQL-Hinweise.

Temporäre MyISAM-Tabellen

Die parallele Abfrageausführung ist nur möglich, wenn InnoDB-Tabellen vorliegen. Aurora MySQL verwendet MyISAM im Hintergrund für temporäre Tabellen. Interne Abfragephasen mit temporären Tabellen werden nie parallel abgefragt. Diese Abfragephasen erkennen Sie am Code `Using temporary` in der `EXPLAIN`-Ausgabe.

Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster

Sie können die Hochleistungsfunktion erweitertes Auditing in Amazon Aurora MySQL verwenden, um Datenbank-Aktivitäten zu überprüfen. Dafür müssen Sie die Sammlung der Prüfprotokolle aktivieren, indem Sie mehrere DB-Cluster-Parameter einstellen. Wenn erweitertes Auditing aktiviert ist, können Sie es verwenden, um eine beliebige Kombination von unterstützten Ereignissen zu protokollieren.

Sie können die Prüfprotokolle ansehen oder herunterladen, um die Prüfinformationen für jeweils eine DB-Instance zu überprüfen. Dazu können Sie die Verfahren in Überwachen von Amazon Aurora-Protokolldateien verwenden.

> **Tip**
>
> Für einen Aurora-DB-Cluster, der mehrere DB-Instances enthält, ist es möglicherweise praktischer, die Prüfprotokolle für alle Instances im Cluster zu untersuchen. Dazu können Sie CloudWatch Logs verwenden. Sie können eine Einstellung auf Cluster-Ebene aktivieren, um die Aurora-MySQL-Prüfprotokolldaten in einer Protokollgruppe in CloudWatch zu veröffentlichen. Anschließend können Sie die Prüfprotokolle über die CloudWatch-Oberfläche anzeigen, filtern und durchsuchen. Weitere Informationen finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.

Aktivieren von erweitertem Auditing

Verwenden Sie die in diesem Abschnitt beschriebenen Parameter, um erweitertes Auditing für Ihr DB-Cluster zu aktivieren und zu konfigurieren.

Verwenden Sie den Parameter `server_audit_logging` zum Aktivieren oder Deaktivieren von Advanced Auditing.

Mit dem Parameter `server_audit_events` können Sie angeben, welche Ereignisse protokolliert werden.

Verwenden Sie die Parameter `server_audit_incl_users` und `server_audit_excl_users`, um anzugeben, wer überprüft werden soll. Standardmäßig werden alle Benutzer überprüft. Weitere Informationen darüber, wie diese Parameter funktionieren, wenn ein oder beide leer bleiben oder in beiden die gleichen Benutzernamen angegeben sind, finden Sie unter server_audit_incl_users und server_audit_excl_users.

Konfigurieren Sie das erweiterte Auditing, indem Sie diese Parameter in der vom DB-Cluster verwendeten Parametergruppe festlegen. Sie können die in Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora gezeigte Prozedur verwenden, um DB-Cluster-Parameter mithilfe der AWS-Managementkonsole zu ändern. Mit dem AWS CLI-Befehl modify-db-cluster-parameter-group oder der Amazon-RDS-API-Operation ModifyDBClusterParameterGroup können Sie DB-Cluster-Parameter programmgesteuert ändern.

Das Ändern dieser Parameter erfordert keinen Neustart des DB-Clusters, wenn die Parametergruppe Ihrem Cluster zugeordnet ist. Wenn Sie die Parametergruppe zum ersten Mal mit dem Cluster verknüpfen, ist ein Neustart des Clusters erforderlich.

Themen

- server_audit_logging
- server_audit_events
- server_audit_incl_users
- server_audit_excl_users

server_audit_logging

Aktiviert oder deaktiviert erweitertes Auditing. Dieser Parameter ist standardmäßig auf OFF eingestellt. Setzten Sie ihn auf ON, um erweiterte Prüfungen zu aktivieren.

In den Protokollen werden keine Prüfungsdaten angezeigt, es sei denn, Sie definieren auch eine oder mehrere Arten von Ereignissen mit dem Parameter `server_audit_events`.

Um sich zu vergewissern, dass Prüfungsdaten für eine DB-Instance protokolliert werden, sehen Sie nach, ob einige Protokolldateien für diese Instance Namen nach dem Schema `audit/audit.log.`*`other_identifying_information`* haben. Befolgen Sie zum Anzeigen der Namen der Protokolldateien die Anleitung unter Anzeigen und Auflisten von Datenbank-Protokolldateien.

server_audit_events

Beinhaltet die durch Kommas getrennte Liste von Ereignissen zum Protokollieren. Ereignisse müssen alle in Großbuchstaben angegeben werden und es sollte kein Leerraum zwischen den Listenelementen bestehen, zum Beispiel: `CONNECT,QUERY_DDL`. Dieser Parameter ist standardmäßig auf eine leere Zeichenfolge eingestellt.

Sie können eine beliebige Kombination der folgenden Ereignisse protokollieren:

- CONNECT – Protokolliert sowohl erfolgreiche als auch fehlgeschlagene Verbindungen und Verbindungstrennungen. Dieses Ereignis beinhaltet Benutzerinformationen.
- QUERY – Protokolliert alle Abfragen in Klartext, einschließlich Abfragen, die aufgrund der Syntax oder Berechtigungsfehlern fehlschlagen.

> **Tip**
>
> Wenn dieser Ereignistyp aktiviert ist, enthalten die Prüfungsdaten Informationen über die kontinuierliche Überwachung und Zustandsprüfung, die Aurora automatisch durchführt. Wenn Sie nur an bestimmten Arten von Operationen interessiert sind, können Sie spezifischere Arten von Ereignissen verwenden. Sie können die CloudWatch-Oberfläche auch verwenden, um in den Protokollen nach Ereignissen zu suchen, die sich auf bestimmte Datenbanken, Tabellen oder Benutzer beziehen.

- QUERY_DCL – Ähnlich dem QUERY-Ereignis, aber gibt nur Data Control Language (DCL)-Abfragen zurück (GRANT, REVOKE usw.).
- QUERY_DDL – Ähnlich dem QUERY-Ereignis, aber gibt nur Data Definition Language (DDL)-Abfragen zurück (CREATE, ALTER usw.).
- QUERY_DML – Ähnlich dem QUERY-Ereignis, aber gibt nur Data Manipulation Language (DML)-Abfragen zurück (INSERT, UPDATE usw. und auch SELECT).
- TABLE – Protokolliert die Tabellen, die von der Ausführung von Abfragen betroffen sind.

> **Note**
>
> In Aurora gibt es keinen Filter, der bestimmte Abfragen aus den Prüfungsprotokollen ausschließt. Wenn Sie `SELECT`-Abfragen ausschließen möchten, müssen Sie alle DML-Anweisungen ausschließen.
> Wenn ein bestimmter Benutzer diese internen `SELECT`-Abfragen in den Prüfungsprotokollen meldet, können Sie diesen Benutzer ausschließen, indem Sie den DB-Cluster-Parameter server_audit_excl_users festlegen. Wenn dieser Benutzer jedoch auch in anderen Aktivitäten verwendet wird und nicht weggelassen werden kann, gibt es keine andere Option zum Ausschließen von `SELECT`-Abfragen.

server_audit_incl_users

Beinhaltet die durch Kommas getrennte Liste mit Benutzernamen für Benutzer, deren Aktivitäten überprüft werden. Zwischen den Listenelementen sollte kein Leerraum bestehen, zum Beispiel: `user_3,user_4`. Dieser Parameter ist standardmäßig auf eine leere Zeichenfolge eingestellt. Die maximale Länge beträgt 1024 Zeichen. Bestimmte Benutzernamen müssen mit entsprechenden Werten in der Spalte `User` der `mysql.user`-Tabelle übereinstimmen. Weitere Informationen zu Benutzernamen finden Sie unter Benutzernamen finden Sie unter Kontobenutzernamen und Passwörter in der MySQL-Dokumentation.

Wenn `server_audit_incl_users` und `server_audit_excl_users` keine Werte beinhalten (standardmäßige Einstellung), werden alle Benutzer überprüft.

Wenn Sie Benutzer zu `server_audit_incl_users` hinzufügen und für `server_audit_excl_users` keinen Wert eingeben, werden nur diese Benutzer überprüft.

Wenn Sie Benutzer zu `server_audit_excl_users` hinzufügen und für `server_audit_incl_users` keinen Wert eingeben, werden alle Benutzer überprüft, die nicht in `server_audit_excl_users` aufgeführt sind.

Wenn Sie dieselben Benutzer zu `server_audit_excl_users` und `server_audit_incl_users` hinzufügen, werden diese Benutzer überprüft. Wenn derselbe Benutzer in beiden Einstellungen aufgeführt ist, erhält `server_audit_incl_users` die höhere Priorität.

Verbinden und trennen Sie Ereignisse, die nicht von dieser Variable betroffen sind. Sie werden immer protokolliert, wenn dies festgelegt wurde. Ein Benutzer wird protokolliert, selbst wenn dieser Benutzer auch im Parameter `server_audit_excl_users` angegeben ist, da `server_audit_incl_users` höhere Priorität hat.

server_audit_excl_users

Beinhaltet die durch Kommas getrennte Liste mit Benutzernamen für Benutzer, deren Aktivitäten nicht überprüft werden. Zwischen den Listenelementen sollte kein Leerraum bestehen, zum Beispiel: `rdsadmin,user_1,user_2`. Dieser Parameter ist standardmäßig auf eine leere Zeichenfolge eingestellt. Die maximale Länge beträgt 1024 Zeichen. Bestimmte Benutzernamen müssen mit entsprechenden Werten in der Spalte `User` der `mysql.user`-Tabelle übereinstimmen. Weitere Informationen zu Benutzernamen finden Sie unter Benutzernamen finden Sie unter Kontobenutzernamen und Passwörter in der MySQL-Dokumentation.

Wenn `server_audit_incl_users` und `server_audit_excl_users` keine Werte beinhalten (standardmäßige Einstellung), werden alle Benutzer überprüft.

Wenn Sie Benutzer zu `server_audit_excl_users` hinzufügen und `server_audit_incl_users` leer lassen, werden nur die unter `server_audit_excl_users` aufgeführten Benutzer nicht überprüft. Alle anderen werden jedoch überprüft.

Wenn Sie dieselben Benutzer zu `server_audit_excl_users` und `server_audit_incl_users` hinzufügen, werden diese Benutzer überprüft. Wenn derselbe Benutzer in beiden Einstellungen aufgeführt ist, erhält `server_audit_incl_users` die höhere Priorität.

Verbinden und trennen Sie Ereignisse, die nicht von dieser Variable betroffen sind. Sie werden immer protokolliert, wenn dies festgelegt wurde. Ein Benutzer wird protokolliert, wenn dieser Benutzer auch im Parameter `server_audit_incl_users` festgelegt ist, da diese Einstellung eine höhere Priorität hat als `server_audit_excl_users`.

Anzeigen von Audit-Protokollen

Sie können Audit-Protokolle mithilfe der Konsole anzeigen und herunterladen. Wählen Sie auf der Seite Datenbanken die DB-Instance aus, um ihre Details anzuzeigen, und scrollen Sie dann in den Abschnitt Protokolle. Die von der Advanced-Auditing-Funktion erstellten Prüfprotokolle haben Namen nach dem Schema `audit/audit.log.`*`other_identifying_information`*.

Um eine Protokolldatei herunterzuladen, wählen Sie die Datei im Bereich Protokolle aus. Wählen Sie dann Herunterladen.

Sie können auch mit dem AWS CLI-Befehl describe-db-log-files eine Liste der Protokolldateien erhalten. Sie können den Inhalt einer Protokolldatei mit dem AWS CLI-Befehl download-db-log-file-portion herunterladen. Weitere Informationen erhalten Sie unter Anzeigen und Auflisten von Datenbank-Protokolldateien und Herunterladen einer Datenbank-Protokolldatei.

Details in Prüfprotokollen

Protokolldateien werden als CSV-Dateien (durch Kommata getrennte Werte) im UTF-8-Format dargestellt. Abfragen werden auch in einfache Anführungszeichen (') gesetzt.

Das Prüfprotokoll wird separat im lokalen Speicher jeder Aurora-MySQL-DB-Instance gespeichert. Jede Instance verteilt Schreibvorgänge auf vier Protokolldateien gleichzeitig. Die maximale Größe einer Protokolldatei ist 100 MB. Wenn dieses nicht konfigurierbare Limit erreicht ist, rotiert Aurora die Datei und generiert eine neue Datei.

> **Tip**
>
> Protokolldateieinträge folgen keiner sequenziellen Reihenfolge. Um die Einträge zu sortieren, verwenden Sie den Wert des Zeitstempels. Sie müssen eventuell alle Protokolldateien überprüfen, um die aktuellen Ereignisse zu sehen. Für mehr Flexibilität beim Sortieren und Durchsuchen der Protokolldaten aktivieren Sie die Einstellung zum Hochladen der Prüfprotokolle auf CloudWatch und sehen Sie sie sich über die CloudWatch-Oberfläche an. Um Prüfdaten mit mehreren Feldtypen und mit Ausgabe im JSON-Format anzuzeigen, können Sie auch die Funktion Datenbankaktivitäts-Streams verwenden. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.

Die Prüfprotokolldateien beinhalten die folgenden durch Kommata getrennten Informationen in Zeilen in der festgelegten Reihenfolge:

Feld	Beschreibung
timestamp	Der präzise Unix-Zeitstempel für das protokollierte Ereignis mit Mikrosekunden.
serverhost	Der Name der Instance, für die das Ereignis protokolliert wird.
username	Der verbundene Benutzername des Benutzers.
host	Der Host, von dem sich der Benutzer verbunden hat.
connectionid	Die Nummer der Verbindungs-ID für den protokollierte Vorgang.
queryid	Die Nummer der Abfragen-ID, die verwendet werden kann, um Ereignisse für relationale Tabellenereignisse und zugehörige Abfragen zu finden. Für `TABLE`-Ereignisse werden mehrere Zeilen hinzugefügt.
operation	Der dokumentierte Aktionstyp. Mögliche Werte sind: `CONNECT`, `QUERY`, `READ`, `WRITE`, `CREATE`, `ALTER`, `RENAME` und `DROP`.
Datenbank	Die aktive Datenbank, wie vom `USE`-Befehl eingestellt.
Objekt	Bei `QUERY`-Ereignissen gibt dieser Wert die Abfrage an, die die Datenbank ausgeführt hat. Bei `TABLE`-Ereignissen gibt er den Tabellennamen an.

Feld	Beschreibung
retcode	Der zurückgegebene Code des protokollierten Vorgangs.

Replikation mit Amazon Aurora My SQL

Die Aurora-MySQL-Replikationsfunktionen sind für die hohe Verfügbarkeit und Leistung Ihres Clusters entscheidend. Ein Aurora erleichtert das Erstellen und Skalieren von Clustern mit bis zu 15 Aurora-Replikaten.

Alle Replicas arbeiten mit denselben zugrunde liegenden Daten. Wenn einige Datenbank-Instances offline gehen, bleiben andere verfügbar, um die Verarbeitung von Abfragen fortzusetzen oder bei Bedarf als Writer zu übernehmen. Aurora verteilt Ihre schreibgeschützten Verbindungen automatisch auf mehrere Datenbank-Instances und hilft einem Aurora-Cluster, abfrageintensive Workloads zu unterstützen.

In den folgenden Themen finden Sie Informationen darüber, wie die Aurora-MySQL-Replikation funktioniert, und wie Sie die Replikationseinstellungen für beste Verfügbarkeit und Leistung anpassen.

Themen

- Verwendung von Aurora-Replicas
- Replikationsoptionen für Amazon Aurora MySQL
- Performance-Überlegungen zur Amazon Aurora MySQL-Replikation
- Konfigurieren von Replikationsfiltern mit Aurora MySQL
- Überwachung der Amazon Aurora MySQL-Replikation
- Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg
- Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation)
- Verwenden der GTID-basierten Replikation

Verwendung von Aurora-Replicas

Aurora-Replicas sind unabhängige Endpunkte in einem Aurora-DB-Cluster, die die beste Methode für das Skalieren von Leseoperationen und Erhöhen der Verfügbarkeit darstellen. Es können bis zu 15 Aurora-Replikate über die Availability Zones verteilt werden, über die sich ein DB-Cluster innerhalb einer AWS-Region erstreckt. Obwohl das DB-Cluster-Volume aus mehreren Kopien der Daten in Ihrem DB-Cluster besteht, werden die Daten im Cluster-Volume als ein zusammengefasstes einzelnes logisches Volume der primären Instance und den Aurora Replicas im DB-Cluster dargestellt. Weitere Informationen über Aurora-Replicas finden Sie unter Aurora-Replikate.

Aurora Replicas funktionieren für das Skalieren von Lesevorgängen, da sie in Ihrem Cluster-Volume vollständig für Lesevorgänge bereit stehen. Schreibvorgänge werden von der primären Instance verwaltet. Da das Cluster-Volume zwischen allen Instances in Ihrem Aurora MySQL-DB-Cluster geteilt wird, ist kein zusätzlicher Arbeitsaufwand erforderlich, um eine Kopie Ihrer Daten für jedes Aurora-Replica zu erstellen. Im Gegensatz dazu müssen MySQL-Read Replicas in einem einzelnen Thread alle Schreiboperationen aus der Quell-DB-Instance in ihrem lokalen Datenspeicher wiedergeben. Dies kann sich auf die Leistungsfähigkeit der unterstützten Lesevorgänge im Datenverkehr mit großem Volumen in Ihren MySQL-Lesereplikaten auswirken.

Mit Aurora MySQL wird beim Löschen eines Aurora-Replica der Instance-Endpunkt sofort entfernt und das Aurora-Replica vom Leser-Endpunkt entfernt. Wenn Anweisungen vorhanden sind, die auf dem Aurora-Replica ausgeführt werden, das gerade gelöscht wird, besteht eine Übergangsfrist von drei Minuten. Vorhandene Anweisungen können während der Nachfrist geordnet beendet werden. Nach Ablauf der Nachfrist wird das Aurora-Replikat geschlossen und gelöscht.

Important

Aurora-Replicas für Aurora MySQL verwenden immer die standardmäßige Transaktionsisolationsebene `REPEATABLE READ` für Vorgänge in InnoDB-Tabellen. Sie können den Befehl `SET TRANSACTION ISOLATION LEVEL` verwenden, um die Transaktionsebene nur für die primäre Instance eines Aurora MySQL-DB-Clusters zu ändern. Diese Beschränkung vermeidet Sperren auf Benutzerebene in Aurora-Replicas und ermöglicht eine Skalierung der Aurora-Replicas, um tausende aktive Benutzerverbindungen bei gleichzeitig minimaler Replica-Verzögerung zu ermöglichen.

Note

Durch DDL-Anweisungen, die auf der primären Instance ausgeführt werden, können Datenbankverbindungen auf den verknüpften Aurora-Replikaten unterbrochen werden. Wenn eine Aurora-Replica-Verbindung aktiv ein Datenbankobjekt (z. B. eine Tabelle) verwendet und dieses Objekt auf der primären Instance mithilfe einer DDL-Anweisung geändert wird, führt das zu einer Unterbrechung der Aurora-Replica-Verbindung.

> **Note**
>
> Die Region China (Ningxia) unterstützt keine regionsübergreifenden Lesereplikate.

Replikationsoptionen für Amazon Aurora MySQL

Sie können eine Replikation zwischen allen folgenden Optionen einrichten:

- Zwei Aurora-MySQL-DB-Cluster in verschiedenen AWS-Regionen, indem Sie ein regionsübergreifendes Lesereplikat eines Aurora-MySQL-DB-Clusters erstellen.

  Weitere Informationen finden Sie unter Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg.
- Zwei Aurora-MySQL-DB-Cluster in der gleichen AWS-Region, indem Sie mit dem MySQL-Binärprotokoll (binlog) eine Replikation einrichten.

  Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).
- Eine RDS for MySQL-DB-Instance als Quelle und ein Aurora MySQL-DB-Cluster, indem Sie ein Aurora-Read Replica einer RDS for MySQL-DB-Instance erstellen.

  Sie können diesen Ansatz verwenden, um bestehende und laufende Datenänderungen in Aurora MySQL während der Migration zu Aurora zu bringen. Weitere Informationen finden Sie unter Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat).

  Sie können diesen Ansatz auch verwenden, um die Skalierbarkeit von Leseabfragen für Ihre Daten zu erhöhen. Sie tun dies, indem Sie die Daten mit einer oder mehreren DB-Instances in einem schreibgeschützten Aurora MySQL-Cluster abfragen. Weitere Informationen finden Sie unter Skalieren von Lesevorgängen für Ihre MySQL-Datenbank mit Amazon Aurora.
- Ein Aurora-MySQL-DB-Cluster in einer AWS-Region und bis zu fünf schreibgeschützte Aurora-MySQL-DB-Cluster in verschiedenen Regionen durch Erstellen einer globalen Aurora-Datenbank.

  Sie können eine Aurora globale Datenbank verwenden, um Anwendungen mit einem weltweiten Fußabdruck zu unterstützen. Der primäre Aurora MySQL-DB-Cluster verfügt über eine Writer-Instance und bis zu 15 Aurora Replikate. Die schreibgeschützten sekundären Aurora MySQL-DB-

Cluster können jeweils aus bis zu 16 Aurora Replikaten bestehen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.

> **Note**
>
> Durch einen Neustart der primären Instance eines Amazon Aurora-DB-Clusters werden auch automatisch die Aurora-Replicas für diesen DB-Cluster neu gestartet, um einen Einstiegspunkt wiederherzustellen, der die Lese-/Schreibkonsistenz innerhalb des DB-Clusters hinweg gewährleistet.

Performance-Überlegungen zur Amazon Aurora MySQL-Replikation

Die folgenden Funktionen helfen Ihnen bei der Optimierung der Leistung der Aurora MySQL-Replikation.

Die Funktion Replica-Protokollkompression reduziert automatisch die Netzwerkbandbreite für Replikationsbenachrichtigungen. Weil jede Benachrichtigung an alle Aurora-Replicas gesendet wird, sind die Vorteile bei umfangreicheren Clustern größer. Diese Funktion beinhaltet einen gewissen CPU-Overhead im Schreiber-Knoten, um die Kompression durchzuführen. In Aurora MySQL Version 2 und Version 3 ist sie immer aktiviert.

Die Funktion Binärprotokollfilterung reduziert automatisch die Netzwerkbandbreite für Replikationsbenachrichtigungen. Da Aurora-Replicas die in den Replikationsbenachrichtigungen enthaltene Binärprotokollinformation nicht verwenden, sind diese Daten in den an diese Knoten gesendeten Benachrichtigungen nicht enthalten.

In Aurora MySQL Version 2 steuern Sie diese Funktion durch eine Änderung des `aurora_enable_repl_bin_log_filtering`-Parameters. Dieser Parameter ist standardmäßig aktiviert. Da diese Optimierung transparent sein soll, können Sie diese Einstellung nur während der Diagnose oder Fehlersuche bei Problemen im Zusammenhang mit der Replikation deaktivieren. Beispielsweise können Sie so das Verhalten eines älteren Aurora MySQL-Clusters anzupassen, bei dem diese Funktion nicht verfügbar war.

Die Binlog-Filterung ist in Aurora MySQL Version 3 immer aktiviert.

Konfigurieren von Replikationsfiltern mit Aurora MySQL

Sie können Replikationsfilter verwenden, um anzugeben, welche Datenbanken und Tabellen mit einem Lesereplikat repliziert werden. Replikationsfilter können Datenbanken und Tabellen in die Replikation einbeziehen oder sie von der Replikation ausschließen.

Im Folgenden finden Sie einige Anwendungsfälle für Replikationsfilter:

- Reduzieren der Größe eines Lesereplikats. Mit Replikationsfiltern können Sie die Datenbanken und Tabellen ausschließen, die für das Lesereplikat nicht benötigt werden.
- Ausschließen von Datenbanken und Tabellen von Lesereplikaten aus Sicherheitsgründen.
- Replizieren verschiedener Datenbanken und Tabellen für spezifische Anwendungsfälle bei verschiedenen Lesereplikaten. Beispielsweise könnten Sie bestimmte Lesereplikate für Analysen oder Sharding verwenden.
- Für einen DB-Cluster mit Read Replicas in verschiedenen AWS-Regionen, um unterschiedliche Datenbanken oder Tabellen in verschiedenen AWS-Regionen zu replizieren.
- Um anzugeben, welche Datenbanken und Tabellen mit einem Aurora-MySQL-DB-Cluster repliziert werden, der als Replikat in einer eingehenden Replikationstopologie konfiguriert ist. Weitere Informationen zu dieser Konfiguration finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Themen

- Einrichten der Parameter der Replikationsfilter für Aurora MySQL
- Einschränkungen der Replikationsfilter für Aurora MySQL
- Beispiele für Replikationsfilter bei Aurora MySQL
- Anzeigen der Replikationsfilter für ein Lesereplikat

Einrichten der Parameter der Replikationsfilter für Aurora MySQL

Zur Konfiguration von Replikationsfiltern legen Sie die folgenden Parameter fest:

- `binlog-do-db` – Repliziert Änderungen in die angegebenen Binärprotokolle. Wenn Sie diesen Parameter für einen Binlog-Quell-Cluster festlegen, werden nur die im Parameter angegebenen Binärprotokolle repliziert.

- `binlog-ignore-db` – Repliziert keine Änderungen in die angegebenen Binärprotokolle. Wenn der Parameter `binlog-do-db` für einen Binlog-Quell-Cluster festgelegt ist, wird dieser Parameter nicht ausgewertet.
- `replicate-do-db` – Repliziert Änderungen der angegebenen Datenbanken. Wenn Sie diesen Parameter für einen Binlog-Replikat-Cluster festlegen, werden nur die im Parameter angegebenen Datenbanken repliziert.
- `replicate-ignore-db` – Repliziert keine Änderungen der angegebenen Datenbanken. Wenn der Parameter `replicate-do-db` für einen Binlog-Replikat-Cluster festgelegt ist, wird dieser Parameter nicht ausgewertet.
- `replicate-do-table` – Repliziert Änderungen der angegebenen Tabellen. Wenn Sie diesen Parameter für ein Lesereplikat festlegen, werden nur die im Parameter angegebenen Tabellen repliziert. Wenn der Parameter `replicate-do-db` oder `replicate-ignore-db` festgelegt ist, müssen Sie die Datenbank, welche die angegebenen Tabellen enthält, in die Replikation mit dem Binlog-Replikat-Cluster einbeziehen.
- `replicate-ignore-table` – Repliziert keine Änderungen der angegebenen Tabellen. Wenn der Parameter `replicate-do-table` für einen Binlog-Replikat-Cluster festgelegt ist, wird dieser Parameter nicht ausgewertet.
- `replicate-wild-do-table` – Repliziert Tabellen basierend auf den angegebenen Namensmustern für Datenbanken und Tabellen. Die Platzhalterzeichen `%` und `_` werden unterstützt. Wenn der Parameter `replicate-do-db` oder `replicate-ignore-db` festgelegt ist, müssen Sie die Datenbank, welche die angegebenen Tabellen enthält, in die Replikation mit dem Binlog-Replikat-Cluster einbeziehen.
- `replicate-wild-ignore-table` – Repliziert keine Tabellen basierend auf den angegebenen Namensmustern für Datenbanken und Tabellen. Die Platzhalterzeichen `%` und `_` werden unterstützt. Wenn der Parameter `replicate-do-table` oder `replicate-wild-do-table` für einen Binlog-Replikat-Cluster festgelegt ist, wird dieser Parameter nicht ausgewertet.

Die Parameter werden in der angegebenen Reihenfolge ausgewertet. Weitere Informationen zur Funktionsweise dieser Parameter finden Sie in der MySQL-Dokumentation:

- Allgemeine Informationen finden Sie unter Optionen und Variablen für Replikatserver.
- Informationen darüber, wie Filterparameter für die Datenbankreplikation ausgewertet werden, finden Sie unter Optionen zur Auswertung der Replikation auf Datenbankebene und für die binäre Protokollierung.

- Informationen darüber, wie Filterparameter für die Tabellenreplikation ausgewertet werden, finden Sie unter Optionen zur Auswertung der Replikation auf Tabellenebene.

Standardmäßig hat jeder dieser Parameter einen leeren Wert. Sie können diese Parameter auf jedem Binlog-Cluster verwenden, um Replikationsfilter festzulegen, zu ändern und zu löschen. Wenn Sie einen dieser Parameter festlegen, trennen Sie die einzelnen Filter durch ein Komma voneinander.

Sie können die Platzhalterzeichen `%` und `_` in den Parametern `replicate-wild-do-table` und `replicate-wild-ignore-table` verwenden. Der Platzhalter `%` entspricht einer beliebigen Anzahl von Zeichen, und der Platzhalter `_` entspricht nur einem Zeichen.

Das binäre Protokollierungsformat der Quell-DB-Instance ist wichtig für die Replikation, da es den Datensatz der Datenänderungen bestimmt. Die Einstellung des Parameters `binlog_format` bestimmt, ob die Replikation zeilenbasiert oder anweisungsbasiert ist. Weitere Informationen finden Sie unter Konfigurieren der Aurora MySQL-Binärprotokollierung für Single-AZ-Datenbanken.

> **Note**
>
> Alle DDL-Anweisungen (Data Definition Language) werden unabhängig von der Einstellung `binlog_format` für die Quell-DB-Instance als Anweisungen repliziert.

Einschränkungen der Replikationsfilter für Aurora MySQL

Folgende Einschränkungen gelten für Replikationsfilter bei Aurora MySQL:

- Replikationsfilter werden nur für Aurora MySQL Version 3 unterstützt.
- Jeder Filterparameter für die Replikation hat ein Limit von 2 000 Zeichen.
- Kommas werden in Replikationsfiltern nicht unterstützt.
- Die Replikationsfilterung unterstützt keine XA-Transaktionen.

  Weitere Informationen finden Sie unter Einschränkungen bei XA-Transaktionen in der MySQL-Dokumentation.

Beispiele für Replikationsfilter bei Aurora MySQL

Um die Replikationsfilter für ein Lesereplikat zu konfigurieren, ändern Sie die Parameter der Replikationsfilter in der DB-Cluster-Parametergruppe, die dem Lesereplikat zugeordnet ist.

> **Note**
>
> Eine Standard-DB-Cluster-Parametergruppe kann nicht modifiziert werden. Erstellen Sie eine neue Parametergruppe und ordnen Sie diese der Lesereplika zu, wenn die Lesereplika eine Standardparametergruppe verwendet. Weitere Informationen zu DB-Cluster-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Sie können Parameter in einer DB-Cluster-Parametergruppe mithilfe der AWS-Managementkonsole-, AWS CLI- oder der RDS-API festlegen. Weitere Informationen zum Festlegen von Parametern finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora. Wenn Sie Parameter in einer Parametergruppe festlegen, verwenden alle DB-Cluster, die der Parametergruppe zugeordnet sind, diese Parametereinstellungen. Wenn Sie Parameter der Replikationsfilter in einer Parametergruppe festlegen, stellen Sie sicher, dass die Parametergruppe nur Replica-Clustern zugeordnet ist. Lassen Sie die Parameter der Replikationsfilter für Quell-DB-Instances leer.

In den folgenden Beispielen werden die Parameter mithilfe von festgeleg AWS CLI. Diese Beispiele legen `ApplyMethod` auf `immediate` fest, sodass die Parameteränderungen unmittelbar nach Abschluss des CLI-Befehls erfolgen. Wenn Sie möchten, dass eine ausstehende Änderung nach dem Neustart des Lesereplikats angewendet wird, legen Sie `ApplyMethod` auf `pending-reboot` fest.

In den folgenden Beispielen werden Replikationsfilter festgelegt:

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example Einschließen von Datenbanken in die Replikation

Das folgende Beispiel schließt die Datenbanken `mydb1` und `mydb2` in die Replikation ein.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example Einschließen von Tabellen in die Replikation

Das folgende Beispiel schließt die Tabellen `table1` und `table2` in der Datenbank `mydb1` in die Replikation ein.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example Einschließen von Tabellen in die Replikation mit Platzhalterzeichen

Das folgende Beispiel schließt Tabellen mit Namen, die mit `order` und `return` beginnen, in Datenbank `mydb` in die Replikation ein.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
```

```
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example Ausschließen von Datenbanken von der Replikation

Das folgende Beispiel schließt die Datenbanken `mydb5` und `mydb6` von der Replikation aus.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6,ApplyMethod=immediate"
```

Example Ausschließen von Tabellen von der Replikation

Das folgende Beispiel schließt die Tabellen `table1` in Datenbank `mydb5` und `table2` in Datenbank `mydb6` von der Replikation aus.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example Ausschließen von Tabellen von der Replikation mit Platzhalterzeichen

Das folgende Beispiel schließt Tabellen mit Namen, die mit `order` und `return` beginnen, in Datenbank `mydb7` von der Replikation aus.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Anzeigen der Replikationsfilter für ein Lesereplikat

Sie können die Replikationsfilter für ein Lesereplikat wie folgt anzeigen:

- Überprüfen Sie die Einstellungen der Parameter der Replikationsfilter in der dem Lesereplikat zugeordneten Parametergruppe.

  Detaillierte Anweisungen finden Sie unter Anzeigen von Parameterwerten für eine DB-Parametergruppe in Amazon Aurora.
- Stellen Sie in einem MySQL-Client eine Verbindung zum Read-Replikat her und führen Sie die`SHOW REPLICA STATUS` Anweisung aus.

  In der Ausgabe werden in den folgenden Feldern die Replikationsfilter für das Lesereplikat angezeigt:

- `Binlog_Do_DB`
- `Binlog_Ignore_DB`
- `Replicate_Do_DB`
- `Replicate_Ignore_DB`
- `Replicate_Do_Table`
- `Replicate_Ignore_Table`
- `Replicate_Wild_Do_Table`
- `Replicate_Wild_Ignore_Table`

Weitere Informationen zu diesen Feldern finden Sie unter Überprüfen des Replikationsstatus in der MySQL-Dokumentation.

Überwachung der Amazon Aurora MySQL-Replikation

Skalieren von Lesevorgängen und hohe Verfügbarkeit hängen von der minimalen Verzögerungszeit ab. Sie können überwachen, wie stark ein Aurora-Replica gegenüber der primären Instance Ihrer Aurora MySQL-DB-Cluster verzögert ist, indem Sie die `AuroraReplicaLag`-Metrik von Amazon CloudWatch überwachen. Die `AuroraReplicaLag`-Metrik wird in jedem Aurora-Replikat aufgezeichnet.

Die primäre DB-Instance zeichnet auch die Amazon CloudWatch-Metriken `AuroraReplicaLagMaximum` und `AuroraReplicaLagMinimum` auf. Die `AuroraReplicaLagMaximum`-Metrik zeichnet die maximale Verzögerung zwischen der primären DB-Instance und jedem Aurora-Replikat im DB-Cluster auf. Die `AuroraReplicaLagMinimum`-Metrik zeichnet die minimale Verzögerung zwischen der primären DB-Instance und jedem Aurora-Replikat im DB-Cluster auf.

Wenn Sie den aktuellen Wert für die Aurora-Replica-Verzögerung benötigen, können Sie die `AuroraReplicaLag`-Metrik in Amazon CloudWatch überprüfen. Die Aurora-Replica-Verzögerung wird auch für jedes Aurora-Replikat Ihres DB-Clusters von Aurora MySQL in der `information_schema.replica_host_status`-Tabelle aufgezeichnet. Weitere Informationen zu dieser Tabelle finden Sie unter information_schema.replica_host_status.

Weitere Informationen über die Überwachung von RDS-Instances und CloudWatch-Metriken finden Sie unter Überwachung von Metriken in einem Amazon-Aurora-Cluster.

Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg

Sie können einen Amazon-Aurora-MySQL-DB-Cluster als Lesereplikat in einer anderen AWS-Region als der Ihres Quell-DB-Clusters erstellen. Durch diesen Ansatz können Sie das Potenzial von Notfallwiederherstellungen steigern, Lesevorgänge in einer Ihren Nutzern näher liegenden AWS-Region skalieren und einfacher zwischen einer AWS-Region und anderen Regionen migrieren.

Sie können Lesereplikate sowohl von verschlüsselten als auch von unverschlüsselten DB-Clustern erstellen. Das Lesereplikat muss verschlüsselt sein, wenn der Quell-DB-Cluster verschlüsselt ist.

Für jeden Quell-DB-Cluster können Sie über höchstens fünf regionsübergreifende DB-Cluster-Lesereplikate verfügen.

> **Note**
>
> Als Alternative zu regionsübergreifenden Lesereplikaten können Sie Lesevorgänge mit minimaler Verzögerung skalieren, indem Sie eine globale Aurora-Datenbank verwenden. Eine globale Aurora-Datenbank hat einen primären Aurora-DB-Cluster in einer AWS-Region und bis zu 10 sekundäre schreibgeschützte DB-Cluster in verschiedenen Regionen. Jeder sekundäre DB-Cluster kann bis zu 16 (statt 15) Aurora-Replikate enthalten. Die Replikation vom primären DB-Cluster zu allen sekundären wird von der Aurora-Speicherschicht und nicht von der Datenbank-Engine durchgeführt, so dass die Verzögerung für die Replikation von Änderungen in der Regel weniger als 1 Sekunde beträgt. Wenn Sie die Datenbank-Engine aus dem Replikationsprozess heraushalten, verarbeitet die Datenbank-Engine nur Workloads. Dies bedeutet auch, dass Sie die Aurora-MySQL-Binärprotokoll-Replikation (Binary Logging) nicht konfigurieren oder verwalten müssen. Weitere Informationen hierzu finden Sie unter Verwenden von Amazon Aurora Global Database.

Wenn Sie eine Aurora-MySQL-DB-Cluster-Read-Replica in einer anderen AWS-Region erstellen, sollten Sie Folgendes beachten:

- Sowohl Ihr Quell-DB-Cluster als auch Ihr regionsübergreifender Lesereplikat-DB-Cluster kann bis zu 15 Aurora-Replikate zusammen mit der primären Instance im DB-Cluster haben. Dank dieser Funktionalität können Lesevorgänge sowohl für die Quell-AWS-Region als auch für die Ziel-AWS-Region der Replikation skaliert werden.

- In einem regionsübergreifenden Szenario besteht eine höhere Verzögerung zwischen dem Quell-DB-Cluster und der Read Replica aufgrund der längeren Netzwerkkanäle zwischen den AWS-Regionen.
- Für Daten, die für eine regionsübergreifende Replikation weitergeleitet werden, fallen Amazon-RDS-Datenübertragungskosten an. Die folgenden regionsübergreifenden Replikationsaktionen generieren Gebühren für die übermittelten Daten aus der Quell-AWS-Region:
  - Wenn Sie das Lesereplikat erstellen, erstellt Amazon RDS einen Snapshot des Quell-Clusters und leitet den Snapshot an die AWS-Region weiter, die das Lesereplikat enthält.
  - Für jede in den Quell-Datenbanken durchgeführte Datenänderung übermittelt Amazon RDS Daten aus der Quellregion an die AWS-Region, die das Lesereplikat enthält.

  Weitere Informationen zu den Kosten von Amazon-RDS-Datenübertragungen finden Sie unter Amazon Aurora – Preise.
- Sie können mehrere gleichzeitige Erstellungs- oder Löschaktionen für Lesereplikate ausführen, die auf den gleichen Quell-DB-Cluster verweisen. Sie müssen jedoch das Limit von höchstens fünf Lesereplikaten für jeden Quell-DB-Cluster einhalten.
- Damit die Replikation effektiv durchgeführt werden kann, sollte jedes Lesereplikat über dieselbe Menge an Ressourcen für Datenverarbeitung und Speicher wie der Quell-DB-Cluster verfügen. Wenn Sie den Quell-DB-Cluster skalieren, sollten Sie auch die Lesereplikate skalieren.

Themen

- Bevor Sie beginnen
- Erstellen eines regionsübergreifenden Lesereplikat-DB-Clusters für Aurora MySQL
- Hochstufen eines Lesereplikats zu einem DB-Cluster für Aurora MySQL
- Fehlerbehebung bei regionsübergreifenden Replikaten für Amazon Aurora MySQL

Bevor Sie beginnen

Bevor Sie einen Aurora-MySQL-DB-Cluster erstellen können, der ein regionsübergreifendes Lesereplikat darstellt, müssen Sie die Binärprotokollierung für Ihren Quell-Aurora-MySQL-DB-Cluster aktivieren. Die regionsübergreifende Replikation für Aurora MySQL verwendet die MySQL-Binärreplikation, um Änderungen im regionsübergreifenden Lesereplikat-DB-Cluster wiederzugeben.

Aktualisieren Sie den `binlog_format`-Parameter für Ihren Quell-DB-Cluster, um die Binärprotokollierung in einem Aurora-MySQL-DB-Cluster zu aktivieren. Der Parameter

`binlog_format` ist ein Parameter auf Cluster-Ebene, der sich in der Cluster-Standardparametergruppe befindet. Wenn der DB-Cluster die Standard-DB-Cluster-Parametergruppe verwendet, müssen Sie eine neue DB-Cluster-Parametergruppe erstellen, um die `binlog_format`-Einstellungen ändern zu können. Wir empfehlen Ihnen, das `binlog_format` auf `MIXED` einzustellen. Jedoch können Sie `binlog_format` auch auf `ROW` oder auf `STATEMENT` einstellen, wenn Sie ein spezifisches Format des Binärprotokolls benötigen. Starten Sie Ihren Aurora-DB-Cluster neu, damit die Änderungen übernommen werden.

Weitere Informationen zur Verwendung der binären Protokollierung mit Aurora MySQL finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation). Weitere Informationen zum Ändern der Aurora MySQL-Konfigurationsparameter finden Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter und Parametergruppen für Amazon Aurora.

Erstellen eines regionsübergreifenden Lesereplikat-DB-Clusters für Aurora MySQL

Sie können mithilfe der AWS-Managementkonsole, der AWS Command Line Interface (AWS CLI) oder der Amazon-RDS-API ein regionsübergreifendes Lesereplikat eines Aurora-DB-Clusters erstellen. Sie können regionsübergreifende Lesereplikate sowohl aus verschlüsselten als auch aus unverschlüsselten DB-Clustern erstellen.

Wenn Sie mithilfe der AWS-Managementkonsole ein regionsübergreifendes Lesereplikat für Aurora MySQL erstellen, erstellt Amazon RDS einen DB-Cluster in der Ziel-AWS-Region und anschließend automatisch eine DB-Instance als primäre Instance für diesen DB-Cluster.

Wenn Sie ein regionsübergreifendes Lesereplikat mithilfe der AWS CLI oder RDS-API erstellen, müssen Sie zuerst den DB-Cluster in der Ziel-AWS-Region erstellen und anschließend warten, bis er aktiviert wird. Sobald es aktiviert ist, erstellen Sie eine DB-Instance, die die primäre Instance für diesen DB-Cluster ist.

Die Replikation beginnt, wenn die primäre Instance des Lesereplikat-DB-Clusters verfügbar wird.

Verwenden Sie die folgenden Prozeduren, um ein regionsübergreifendes Lesereplikat für einen Aurora MySQL-DB-Cluster zu erstellen. Diese Prozeduren funktionieren für das Erstellen von Lesereplikaten aus verschlüsselten oder unverschlüsselten DB-Clustern.

Konsole

So erstellen Sie mithilfe der AWS-Managementkonsole einen Aurora-MySQL-DB-Cluster, der ein regionsübergreifendes Lesereplikat ist

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der rechten oberen Ecke der AWS-Managementkonsole die AWS-Region aus, die Ihr Quell-DB-Cluster hostet.
3. Wählen Sie im Navigationsbereich Datenbanken aus.
4. Wählen Sie den DB-Cluster aus, um ein regionsübergreifendes Lesereplikat dafür zu erstellen.
5. Wählen Sie unter Aktionen die Option Regionsübergreifende Read Replica erstellen aus.
6. Wählen Sie auf der Seite Regionsübergreifende Read Replica erstellen die Optionseinstellungen für Ihren regionsübergreifenden Lesereplikat-DB-Cluster aus, wie in der folgenden Tabelle beschrieben.

Option	Beschreibung
Zielregion	Wählen Sie die AWS-Region zum Hosten des neuen regionsübergreifenden Lesereplikat-DB-Clusters aus.
Zieladressen-DB-Subnetzgruppe	Wählen Sie die DB-Subnetzgruppe aus, die für den regionsübergreifenden Lesereplikat-DB-Cluster verwendet werden soll.
Öffentlich zugänglich	Wählen Sie Ja aus, um dem regionsübergreifenden Lesereplikat-DB-Cluster eine öffentliche IP-Adresse zuzuweisen, andernfalls wählen Sie Nein aus.
Verschlüsselung	Wählen Sie Enable Encryption (Verschlüsselung aktivieren) aus, um die Verschlüsselung im Ruhezustand für diesen DB-Cluster zu aktivieren. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
AWS KMS key	Diese Option ist nur verfügbar, wenn Verschlüsselung auf Verschlüsselung aktivieren festgelegt wurde.

Option	Beschreibung
	Wählen Sie die AWS KMS key, die für die Verschlüsselung dieses DB-Clusters verwendet werden soll. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
DB-Instance-Klasse	Wählen Sie eine DB-Instance-Klasse aus, die die Verarbeitungs- und Speicheranforderungen der primären Instance im DB-Cluster definiert. Weitere Informationen zu den Optionen für DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
Multi-AZ-Bereitstellung	Wählen Sie Ja aus, um ein Lesereplikat des neuen DB-Clusters in einer anderen Availability Zone in der Ziel-AWS-Region für den Failover-Support zu erstellen. Weitere Informationen über die Multi-AZ-Bereitstellung finden Sie unter Regionen und Availability Zones.
Lese-Replikat-Quelle	Wählen Sie den Quell-DB-Cluster aus, um ein regionsübergreifendes Lesereplikat dafür zu erstellen.

Option	Beschreibung
DB-Instance-Kennung	Geben Sie einen Namen für die primäre Instance in Ihrem regionsübergreifenden Lesereplikat-DB-Cluster ein. Dieser Bezeichner wird in der Endpunktadresse für die primäre Instance des neuen DB-Clusters verwendet. Für den DB-Instance-Bezeichner gelten folgende Einschränkungen: • Sie darf 1 bis 63 alphanumerische Zeichen oder Bindestriche enthalten. • Das erste Zeichen muss ein Buchstabe sein. • Es darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten. • Er muss für alle DB-Instances, jedes AWS-Konto und jede AWS-Region eindeutig sein. Da der regionsübergreifende Lesereplikat-DB-Cluster aus einem Snapshot des Quell-DB-Clusters erstellt wird, stimmen der Masterbenutzername und das Masterpasswort für das Lesereplikat mit denen des Quell-DB-Clusters überein.

Option	Beschreibung
DB-Cluster-Kennung	Geben Sie einen Namen für den regionsübergreifen den Lesereplikat-DB-Cluster ein, der für das Replikat in Ihrem Konto in der Ziel-AWS-Region eindeutig ist. Dieser Bezeichner wird in der Cluster-Endpunktad resse für den DB-Cluster verwendet. Weitere Informati onen zum Cluster-Endpunkt finden Sie unter Amazon-Aurora-Endpunktverbindungen. Für den DB-Cluster-Bezeichner gelten folgende Einschränkungen: • Sie darf 1 bis 63 alphanumerische Zeichen oder Bindestriche enthalten. • Das erste Zeichen muss ein Buchstabe sein. • Es darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten. • Es muss für alle DB-Cluster in jedem AWS-Konto und jeder AWS-Region eindeutig sein.
Priorität	Wählen Sie eine Failover-Priorität für die primäre Instance des DB-Clusters aus. Diese Priorität bestimmt die Reihenfolge, in der Aurora-Replikate bei der Wiederherstellung nach einem Ausfall der primären Instance hochgestuft werden. Wenn Sie keinen Wert auswählen, wird als Standard tier-1 eingestellt. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.

Option	Beschreibung
Datenbankport	Geben Sie den Port an, über den Anwendungen und Dienstprogramme auf die Datenbank zugreifen können. Aurora-DB-Cluster verwenden standardm äßig den MySQL-Standardport 3306. Die Firewalls einiger Unternehmen blockieren Verbindungen mit diesem Port. Wenn die Firewall Ihres Unternehmens den Standardport blockiert, wählen Sie einen anderen Port für den neuen DB-Cluster aus.
Verbesserte Überwachung	Wählen Sie Erweiterte Überwachung aktivieren aus, um die Erfassung von Metriken in Echtzeit für das Betriebssystem zu aktivieren, in dem Ihr DB-Cluste r ausgeführt wird. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Übe rwachung).
Überwachungsrolle	Nur verfügbar, wenn Erweiterte Überwachung auf Erweiterte Überwachung aktivieren gesetzt ist. Wählen Sie die IAM-Rolle aus, die Sie erstellt haben, damit Amazon RDS für Sie mit Amazon CloudWatch Logs kommunizieren kann, oder wählen Sie Standard aus, damit RDS eine Rolle namens `rds-monitoring-role` für Sie erstellt. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Übe rwachung).
Granularität	Nur verfügbar, wenn Erweiterte Überwachung auf Erweiterte Überwachung aktivieren gesetzt ist. Mit ihr können Sie die Zeitspanne zwischen den Erfassung en der Kennzahlen des DB-Clusters in Sekunden festlegen.

Option	Beschreibung
Kleinere Versions-Updates automatisch aktivieren	Diese Einstellung wird für Aurora MySQL-DB-Cluster nicht verwendet. Weitere Informationen über Engine-Updates für Aurora MySQL finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.

7. Wählen Sie Create (Erstellen) aus, um ein regionsübergreifendes Lesereplikat für Aurora zu erstellen.

AWS CLI

So erstellen Sie mithilfe der CLI einen Aurora MySQL-DB-Cluster, der ein regionsübergreifendes Lesereplikat ist:

1. Rufen Sie den AWS CLI-Befehl create-db-cluster in der AWS-Region auf, in der Sie den Lesereplikat-DB-Cluster erstellen möchten. Fügen Sie die Option `--replication-source-identifier` ein und legen Sie den Amazon-Ressourcennamen (ARN) des Quell-DB-Clusters fest, für den Sie ein Lesereplikat erstellen möchten.

   Für die regionsübergreifende Replikation, bei der der durch `--replication-source-identifier` identifizierte DB-Cluster verschlüsselt ist, geben Sie die Optionen `--kms-key-id` und `--storage-encrypted` an.

   > **Note**
   >
   > Sie können eine regionsübergreifende Replikation aus einem unverschlüsselten DB-Cluster zu einem verschlüsselten Lesereplikat einrichten, indem Sie `--storage-encrypted` festlegen und einen Wert für `--kms-key-id` angeben.

   Sie können die Parameter `--master-username` und `--master-user-password` nicht angeben. Diese Werte werden aus dem Quell-DB-Cluster übernommen.

   Im folgenden Code-Beispiel wird ein Lesereplikat in der Region us-east-1 aus einem unverschlüsselten DB-Cluster-Snapshot in der Region us-west-2 erstellt. Der Befehl wird in der

Region us-east-1 aufgerufen. In diesem Beispiel wird die Option `--manage-master-user-password` zum Generieren des Hauptbenutzerpassworts und zum Verwalten dieses Passworts in Secrets Manager angegeben. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager. Alternativ können Sie die Option `--master-password` verwenden, um das Passwort selbst festzulegen und zu verwalten.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

Für Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

Im folgenden Code-Beispiel wird ein Lesereplikat in der Region us-east-1 aus einem verschlüsselten DB-Cluster-Snapshot in der Region us-west-2 erstellt. Der Befehl wird in der Region us-east-1 aufgerufen.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.08.0 \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster \
  --kms-key-id my-us-east-1-key \
  --storage-encrypted
```

Für Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora-mysql ^
  --engine-version 8.0.mysql_aurora.3.08.0 ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster ^
  --kms-key-id my-us-east-1-key ^
  --storage-encrypted
```

Die Option `--source-region` ist für die regionsübergreifende Replikation zwischen den Region AWS GovCloud (USA-Ost) und AWS GovCloud (USA West) erforderlich, in denen der durch `--replication-source-identifier` identifizierte DB-Cluster verschlüsselt ist. Geben Sie für `--source-region` die AWS-Region des Quell-DB-Clusters an.

Wenn `--source-region` nicht festgelegt ist, geben Sie einen `--pre-signed-url`-Wert an. Eine vorsignierte URL ist eine URL, die eine mit der Signaturversion 4 signierte Anforderung für den Befehl `create-db-cluster` enthält, der in der Quell-AWS-Region aufgerufen wird. Weitere Informationen über die Option `pre-signed-url` finden Sie unter create-db-cluster in der AWS CLI-Befehlsreferenz.

2. Überprüfen Sie die Verfügbarkeit des DB-Clusters, indem Sie den AWS CLI-Befehl describe-db-clusters verwenden, wie im folgenden Beispiel gezeigt.

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

Wenn der Befehl **`describe-db-clusters`** den Status `available` anzeigt, erstellen Sie die primäre Instance für den DB-Cluster, damit die Replikation gestartet werden kann. Verwenden Sie dazu den AWS CLI-Befehl create-db-instance, wie im folgenden Beispiel gezeigt.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier sample-replica-cluster \
  --db-instance-class db.r5.large \
  --db-instance-identifier sample-replica-instance \
  --engine aurora-mysql
```

Für Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier sample-replica-cluster ^
  --db-instance-class db.r5.large ^
  --db-instance-identifier sample-replica-instance ^
  --engine aurora-mysql
```

Wenn die DB-Instance erstellt wurde und verfügbar ist, wird die Replikation gestartet. Sie können feststellen, ob die DB-Instance verfügbar ist, indem Sie den AWS CLI-Befehl describe-db-instances aufrufen.

RDS-API

So erstellen Sie mithilfe der API einen Aurora MySQL-DB-Cluster, der ein regionsübergreifendes Lesereplikat ist:

1. Rufen Sie die RDS-API-Aktion CreateDBCluster in der AWS-Region auf, in der Sie einen Read-Replica-DB-Cluster erstellen möchten. Fügen Sie den Parameter `ReplicationSourceIdentifier` ein und legen Sie den Amazon-Ressourcennamen (ARN) des Quell-DB-Clusters fest, für den Sie ein Lesereplikat erstellen möchten.

   Für die regionsübergreifende Replikation, bei der der durch `ReplicationSourceIdentifier` identifizierte DB-Cluster verschlüsselt ist, geben Sie den Parameter `KmsKeyId` an und legen den Parameter `StorageEncrypted` auf `true` fest.

   > **Note**
   >
   > Sie können eine regionsübergreifende Replikation aus einem unverschlüsselten DB-Cluster zu einem verschlüsselten Lesereplikat einrichten, indem Sie für `StorageEncrypted` den Wert **`true`** festlegen und einen Wert für `KmsKeyId` angeben. In diesem Fall müssen Sie nicht festlege `PreSignedUrl`.

   Sie müssen die Parameter `MasterUsername` und `MasterUserPassword` nicht mit einschließen, da diese Werte aus dem Quell-DB-Cluster bezogen werden.

Im folgenden Code-Beispiel wird ein Lesereplikat in der Region us-east-1 aus einem unverschlüsselten DB-Cluster-Snapshot in der Region us-west-2 erstellt. Die Aktion wird in der Region us-east-1 aufgerufen.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

Im folgenden Code-Beispiel wird ein Lesereplikat in der Region us-east-1 aus einem verschlüsselten DB-Cluster-Snapshot in der Region us-west-2 erstellt. Die Aktion wird in der Region us-east-1 aufgerufen.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &KmsKeyId=my-us-east-1-key
  &StorageEncrypted=true
  &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBCluster
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-
west-2%252Frds%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
```

```
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-
amz-content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

Geben Sie für die regionsübergreifende Replikation zwischen den Regionen AWS GovCloud (USA-Ost) und AWS GovCloud (USA West), in denen der durch `ReplicationSourceIdentifier` identifizierte DB-Cluster verschlüsselt ist, außerdem den Parameter `PreSignedUrl` an. Die vorsignierte URL muss eine gültige Anforderung für die API-Operation `CreateDBCluster` sein, die in der Quell-AWS-Region mit dem zu replizierenden verschlüsselten DB-Cluster ausgeführt werden kann. Der KMS-Schlüsselbezeichner wird zur Verschlüsselung des Lesereplikats verwendet und muss ein für die Ziel-AWS-Region gültiger KMS-Schlüssel sein. Verwenden Sie stattdessen den AWS CLI-Befehl create-db-cluster mit der Option `--source-region`, um eine vorsignierte URL automatisch (und nicht manuell) zu generieren.

2. Überprüfen Sie die Verfügbarkeit des DB-Clusters, indem Sie die RDS-API-Operation DescribeDBClusters verwenden, wie im folgenden Beispiel gezeigt.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=DescribeDBClusters
  &DBClusterIdentifier=sample-replica-cluster
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T002223Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
```

```
  &X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

Wenn die Ergebnisse von `DescribeDBClusters` den Status `available` anzeigen, erstellen Sie die primäre Instance für den DB-Cluster, damit die Replikation gestartet werden kann. Verwenden Sie dafür die RDS-API-Operation CreateDBInstance, wie im folgenden Beispiel gezeigt.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBInstance
  &DBClusterIdentifier=sample-replica-cluster
  &DBInstanceClass=db.r5.large
  &DBInstanceIdentifier=sample-replica-instance
  &Engine=aurora-mysql
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T003808Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

Wenn die DB-Instance erstellt wurde und verfügbar ist, wird die Replikation gestartet. Sie können feststellen, ob die DB-Instance verfügbar ist, indem Sie den AWS CLI-Befehl DescribeDBInstances aufrufen.

Anzeigen von regionsübergreifenden Amazon Aurora MySQL-Replikaten

Sie können die regionsübergreifenden Replikationsbeziehungen Ihres Amazon-Aurora-MySQL-DB-Clusters anzeigen, indem Sie den AWS CLI-Befehl describe-db-clusters oder die RDS-API-Operation DescribeDBClusters aufrufen. In der Antwort sind die DB-Clusterkennungen aller regionsübergreifenden Lesereplikat-DB-Cluster im Feld `ReadReplicaIdentifiers` aufgeführt. Das Element `ReplicationSourceIdentifier` gibt Aufschluss über den ARN des Quell-DB-Clusters, der die Replikationsquelle ist.

Hochstufen eines Lesereplikats zu einem DB-Cluster für Aurora MySQL

Sie können ein Aurora MySQL-Lesereplikat zu einem eigenständigen DB-Cluster hochstufen. Wenn Sie ein Aurora MySQL-Lesereplikat hochstufen, werden ihre DB-Instances neu gestartet, bevor sie verfügbar werden.

Typischerweise stufen Sie ein Aurora MySQL-Lesereplikat zu einem eigenständigen DB-Cluster als Datenwiederherstellungsschema hoch, wenn der Quell-DB-Cluster ausfällt.

Erstellen Sie dazu zuerst ein Lesereplikat und überwachen Sie anschließend den Quell-DB-Cluster auf Fehler. Im Fall eines Ausfalls machen Sie Folgendes:

1. Stufen Sie das Lesereplikat hoch.
2. Leiten Sie den Datenverkehr der Datenbank an den hochgestuften DB-Cluster weiter.
3. Erstellen Sie ein Ersatz-Lesereplikat mit dem hochgestuften DB-Cluster als Quelle.

Wenn Sie ein Lesereplikat hochstufen, wird das Lesereplikat zu einem eigenständigen Aurora-DB-Cluster. Das Hochstufen kann einige Minuten oder mehr in Anspruch nehmen, abhängig von der Größe des Lesereplikats. Nachdem Sie das Lesereplikat zu einem neuen DB-Cluster hochgestuft haben, ist es wie bei jedem anderen DB-Cluster. Beispielsweise können Sie daraus Lesereplikate erstellen und zeitpunktbezogene Wiederherstellungsoperationen durchführen. Sie können auch Aurora-Replicas für den DB-Cluster erstellen.

Da der hochgestufte DB-Cluster kein Lesereplikat mehr ist, können Sie ihn nicht mehr als Replikationsziel verwenden.

Die folgenden Schritte zeigen den allgemeinen Vorgang für das Hochstufen eines Lesereplikats zu einem DB-Cluster:

1. Halten Sie alle Schreibvorgänge von Transaktionen im Lesereplikat-Quell-DB-Cluster an und warten Sie anschließend, bis alle Updates für das Lesereplikat abgeschlossen wurden. Datenbank-Updates werden im Lesereplikat durchgeführt, nachdem Sie im Quell-DB-Cluster aufgetreten sind; diese Replikationsverzögerung kann beträchtlich variieren. Verwenden Sie die Metrik `ReplicaLag`, um zu bestimmen, wann alle Aktualisierungen am Lesereplikat vorgenommen wurden. Die Metrik `ReplicaLag` zeichnet die Zeit auf, die eine Lesereplikat-DB-Instance hinter der Quell-DB-Instance zurückbleibt. Wenn die Metrik `ReplicaLag` den Wert `0` erreicht, hat das Lesereplikat den Stand der Quell-DB-Instance erreicht.
2. Stufen Sie das Lesereplikat mit der Option Hochstufen in der Amazon-RDS-Konsole, dem AWS CLI- Befehl promote-read-replica-db-cluster oder der Amazon-RDS-API Operation PromoteReadReplicaDBCluster hoch.

   Sie wählen eine Aurora MySQL-DB-Instance aus, um das Lesereplikat hochzustufen. Nachdem das Lesereplikat hochgestuft wurde, wird der Aurora MySQL-DB-Cluster zu einem eigenständigen DB-Cluster hochgestuft. Die DB-Instance mit der höchsten Failover-Priorität wird zu der primären

DB-Instance für den DB-Cluster hochgestuft. Die anderen dB-Instances werden zu Aurora-Replicas.

> **Note**
>
> Das Hochstufen kann einige Minuten in Anspruch nehmen. Wenn Sie ein Lesereplikat hochstufen, wird die Replikation gestoppt und die DB-Instances werden neu gestartet. Sobald der Neustart abgeschlossen ist, steht das Lesereplikat als neuer DB-Cluster zur Verfügung.

Konsole

So stufen Sie ein Aurora MySQL-Lesereplikat zu einem DB-Cluster hoch:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der Konsole Instances aus.

   Der Bereich Instance wird angezeigt.
3. Wählen Sie im Bereich Instances das Lesereplikat aus, das Sie hochstufen möchten.

   Die Lesereplikate werden als Aurora MySQL-DB-Instances angezeigt.
4. Wählen Sie unter Aktionen Promote read replica (Read Replica hochstufen) aus.
5. Wählen Sie auf der Bestätigungsseite Read Replica hochstufen aus.

AWS CLI

Verwenden Sie den AWS CLI-Befehl promote-read-replica-db-cluster, um ein Lesereplikat zu einem eigenständigen DB-Cluster hochzustufen.

Example

Für Linux, macOS oder Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier mydbcluster
```

Für Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier mydbcluster
```

RDS-API

Rufen Sie PromoteReadReplicaDBCluster auf, um ein Lesereplikat zu einem DB-Cluster hochzustufen.

Fehlerbehebung bei regionsübergreifenden Replikaten für Amazon Aurora MySQL

Im Folgenden finden Sie eine Liste der häufigen Fehlermeldungen, die Sie beim Erstellen eines regionsübergreifenden Amazon Aurora-Lesereplikats erhalten könnten, und wie Sie die angegebenen Fehler beheben können.

Für Quell-Cluster [DB-Cluster-ARN] sind Binärprotokolle nicht aktiviert

Um dieses Problem zu lösen, aktivieren Sie die Binärprotokollierung für das Quell-DB-Cluster. Weitere Informationen finden Sie unter Bevor Sie beginnen.

Quell-Cluster [DB-Cluster-ARN] verfügt über keine Cluster-Parametergruppe, die zu einem Writer synchron ist

Diese Fehlermeldung erhalten Sie, wenn Sie den DB-Cluster-Parameter `binlog_format` aktualisiert haben, aber die primäre Instance für das DB-Cluster nicht neu gestartet haben. Starten Sie die primäre Instance (die schreibende Instance) für das DB-Cluster neu und versuchen Sie es erneut.

Quell-Cluster [DB-Cluster-ARN] verfügt bereits über ein Lesereplikat in dieser Region

Sie können über bis zu fünf regionsübergreifende DB-Cluster verfügen, die Lesereplikate für jeden Quell-DB-Cluster in jeder AWS-Region darstellen. Wenn in einer bestimmten AWS-Region bereits die maximale Anzahl von Lesereplikaten für einen DB-Cluster vorhanden ist, müssen Sie ein vorhandenes löschen, bevor Sie einen neuen regionsübergreifenden DB-Cluster in dieser Region erstellen können.

Für DB-Cluster [DB-Cluster-ARN] ist ein Upgrade der Datenbank-Engine für die Unterstützung einer regionsübergreifenden Replikation erforderlich

Führen Sie ein Upgrade der Datenbank-Engine-Version für alle Instances im Quell-DB-Cluster auf die aktuelle Datenbank-Engine-Version durch und versuchen Sie erneut, eine regionsübergreifende Lesereplikat-DB zu erstellen.

Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation)

Da Amazon Aurora MySQL mit MySQL kompatibel ist, können Sie eine Replikation zwischen einer MySQL-Datenbank und einem Amazon Aurora MySQL-DB-Cluster einrichten. Bei diesem Replikationstyp wird die binäre MySQL-Protokoll-Replikation verwendet, die auch als Binärprotokoll-Replikation bezeichnet wird. Wenn Sie die binäre Protokollreplikation mit Aurora verwenden, sollte Ihre MySQL-Datenbank MySQL-Version 5.5 oder höher verwenden. Sie können eine Replikation einrichten, bei der Ihr Aurora MySQL DB-Cluster die Replikationsquelle oder das Replica ist. Sie können mit einer Amazon-RDS-MySQL-DB-Instance, einer MySQL-Datenbank außerhalb von Amazon RDS oder einem anderen Aurora MySQL-DB-Cluster replizieren.

> **Note**
>
> Sie können die an bestimmte Arten von Aurora-Clustern gesendete oder davon empfangene Binlog-Replikation nicht verwenden. Insbesondere ist die Binlog-Replikation nicht für Aurora Serverless v1-Cluster verfügbar. Wenn die Anweisungen `SHOW MASTER STATUS` und `SHOW SLAVE STATUS` (Aurora-MySQLVersion 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) keine Ausgabe zurückgeben, überprüfen Sie, ob der von Ihnen verwendete Cluster die Binlog-Replikation unterstützt.

Sie können eine Replikation auch mit einer RDS-for-MySQL-DB-Instance oder einem Aurora-MySQL-DB-Cluster in einer anderen AWS-Region vornehmen. Wenn Sie eine Replikation über AWS-Regionen hinweg durchführen, stellen Sie sicher, dass Ihre DB-Cluster und DB-Instances öffentlich zugänglich sind. Wenn sich die Aurora MySQL-DB-Cluster in privaten Subnetzen in Ihrer VPC befinden, verwenden Sie VPC-Peering zwischen den AWS-Regionen. Weitere Informationen finden Sie unter Ein DB-Cluster in einer VPC, auf den eine EC2-Instance in einer anderen VPC zugreift.

Erstellen Sie einen DB-Cluster von Aurora MySQL als Lesereplikat in einer anderen AWS-Region als der des Quell-DB-Clusters, um die Replikation zwischen einem DB-Cluster von Aurora MySQL und einem DB-Cluster von Aurora MySQL in einer anderen AWS-Region zu konfigurieren. Weitere Informationen finden Sie unter Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg.

Mit Auror-MySQL-Version 2 und 3 können Sie eine Replikation zwischen Aurora MySQL und einer externen Quelle oder einem Ziel durchführen, die bzw. das globale Transaktionskennungen (Global Transaction Identifiers, GTIDs) für die Replikation verwendet. Stellen Sie sicher, dass die

Einstellungen der GTID-bezogenen Parameter im Aurora MySQL-DB-Cluster mit dem GTID-Staus der externen Datenbank kompatibel sind. Weitere Informationen zur Vorgehensweise finden Sie unter Verwenden der GTID-basierten Replikation. In Aurora-MySQL-Version 3.01 und höher können Sie auswählen, wie GTIDs Transaktionen zugewiesen werden, die von einer Quelle repliziert werden, die keine GTIDs verwendet. Informationen zur gespeicherten Prozedur, die diese Einstellung steuert, finden Sie unter mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL Version 3).

Warning

Wenn Sie zwischen Aurora MySQL und MySQL replizieren, dürfen Sie nur InnoDB-Tabellen verwenden. Wenn Sie MyISAM-Tabellen replizieren möchten, müssen Sie diese mit dem folgenden Befehl in das InnoDB-Format konvertieren, bevor Sie die Replikation einrichten.

```
alter table <schema>.<table_name> engine=innodb, algorithm=copy;
```

In den folgenden Abschnitten richten Sie die Replikation ein, halten die Replikation an, skalieren Lesevorgänge für Ihre Datenbank, optimieren die Binlog-Replikation und richten das erweiterte Binlog ein.

Themen

- Einrichten einer binären Protokollreplikation für Aurora MySQL
- Anhalten einer binären Protokollreplikation für Aurora MySQL
- Skalieren von Lesevorgängen für Ihre MySQL-Datenbank mit Amazon Aurora
- Optimieren einer binären Protokollreplikation für Aurora MySQL
- Einrichten eines erweiterten Binärprotokolls für Aurora MySQL

Einrichten einer binären Protokollreplikation für Aurora MySQL

Das Einrichten einer MySQL-Replikation mit Aurora MySQL umfasst die folgenden Schritte, die ausführlich erklärt werden:

Inhalt

- 1. Aktivieren der Binärprotokollierung für die Replikationsquelle
- 2. Beibehaltung der Binärprotokolle für die Replikationsquelle, bis diese nicht mehr erforderlich sind
- 3. Erstellen einer Kopie oder eines Dumps Ihrer Replikationsquelle

- 4. Laden des Dumps in das Replikatziel (falls erforderlich)
- 5. Erstellen Sie einen Replikationsbenutzer auf Ihrer Replikationsquelle
- 6. Aktivieren der Replikation für das Replikat-Ziel
  - Festlegen einer Position zum Stoppen der Replikation zu einer Read Replica
- 7. Überwachen Ihres Replikats
- Synchronisierung von Passwörtern zwischen Replikationsquelle und -ziel

1. Aktivieren der Binärprotokollierung für die Replikationsquelle

Im Folgenden finden Sie die Anweisungen zum Aktivieren der Binärprotokollierung an der Replikationsquelle für Ihre Datenbank-Engine.

Datenbank-Engine	Anleitungen
Aurora MySQL	So aktivieren Sie die Binärprotokollierung für einen Aurora-MySQL-DB-Cluster Legen Sie den Parameter `binlog_format` auf `ROW`, `STATEMENT` oder `MIXED` fest. `MIXED` wird empfohlen, sofern kein bestimmtes Format für das Binärprotokoll notwendig ist. (Der Standardwert ist `OFF`.) Um den`binlog_format` -Parameter zu ändern, erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe und ordnen Sie diese benutzerdefinierte Parametergruppe Ihrem DB-Cluster zu. Sie können die Parameter in einer Standard-DB-Clusterparametergruppe nicht ändern. Wenn Sie den Parameter `binlog_format` von `OFF` zu einem anderen Wert ändern, müssen Sie Ihren Aurora-DB-Cluster neu starten, damit die Änderung wirksam wird. Weitere Informationen erhalten Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter und Parametergruppen für Amazon Aurora.
RDS für MySQL	So aktivieren Sie die Binärprotokollierung für eine Amazon-RDS-DB-Instance Sie können die Binärprotokollierung nicht direkt für eine Amazon-RDS-DB-Instance aktivieren, aber Sie können diese Funktion aktivieren, indem Sie Folgendes tun:

Datenbank-Engine	Anleitungen
	• Deaktivieren Sie die automatischen Backups für die DB-Instance. Sie können automatische Backups aktivieren, wenn Sie eine DB-Instance erstellen, oder Sie können Backups aktivieren, indem Sie eine bestehende DB-Instance ändern. Weitere Informationen finden Sie unter Erstellen einer DB-Instance im Amazon-RDS-Benutzerhandbuch. • Erstellen Sie ein Lesereplikat für die DB-Instance. Weitere Informationen finden Sie unter Arbeiten mit Read Replicas im Amazon-RDS-Benutzerhandbuch.

Datenbank-Engine	Anleitungen
MySQL (extern)	So richten Sie eine verschlüsselte Replikation ein Wenn Sie Daten sicher mit Aurora-MySQL-Version 2 replizieren möchten, verwenden Sie eine verschlüsselte Replikation. **Note** Wenn Sie keine verschlüsselte Replikation benötigen, können Sie diese Schritte überspringen. Folgende Voraussetzungen gelten für die Verwendung einer verschlüsselten Replikation: • Secure Sockets Layer (SSL) muss in der externen MySQL-Quelldatenbank aktiviert sein. • Ein Clientschlüssel und ein Clientzertifikat müssen für das Aurora MySQL-DB-Cluster vorbereitet werden. Während der verschlüsselten Replikation dient das Aurora MySQL-DB-Cluster als Client für den MySQL-Datenbankserver. Die Zertifikate und Schlüssel für den Aurora MySQL-Client befinden sich in Dateien im PEM-Format. 1. Stellen Sie sicher, dass alle Vorbereitungen für die verschlüsselte Replikation getroffen wurden: • Wenn SSL auf dem externen Server mit der MySQL-Quelldatenbank nicht aktiviert ist und Sie keinen Clientschlüssel und kein Clientzertifikat vorbereitet haben, aktivieren Sie SSL auf dem MySQL-Datenbankserver und generieren Sie den Clientschlüssel und das Clientzertifikat. • Wenn SSL auf der externen Quelle aktiviert ist, geben Sie einen Clientschlüssel und ein Clientzertifikat für das Aurora MySQL-DB-Cluster an. Wenn Sie diese Werte nicht haben, erstellen Sie ein einen neuen Schlüssel und ein neues Zertifikat für das Aurora MySQL-DB-Cluster. Sie benötigen zur

Datenbank-Engine	Anleitungen

Signierung des Clientzertifikats den Zertifizierungsstellenschlüssel, den Sie zum Konfigurieren von SSL auf der externen MySQL-Quelldatenbank verwendet haben.

Weitere Informationen finden Sie unter Creating SSL Certificates and Keys Using openssl in der MySQL-Dokumentation.

Sie benötigen das Zertifizierungsstellenzertifikat, den Clientschlüssel und das Clientzertifikat.

2. Stellen Sie als Masterbenutzer über SSL eine Verbindung zum Aurora MySQL-DB-Cluster her.

   Informationen zum Herstellen einer Verbindung mit einem Aurora MySQL-DB-Cluster über SSL finden Sie unter TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

3. Führen Sie die gespeicherte Prozedur `mysql.rds_import_binlog_ssl_material` aus, um die SSL-Informationen in das Aurora MySQL-DB-Cluster zu importieren.

   Fügen Sie die Informationen aus den PEM-Dateien für das Aurora MySQL-DB-Cluster in die korrekte JSON-Nutzlast für den Parameter `ssl_material_value` ein.

   Im folgenden Beispiel werden SSL-Informationen in ein Aurora MySQL-DB-Cluster importiert. Der Code in den PEM-Dateien ist in der Regel länger als in diesem Beispiel.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
```

Datenbank-Engine	Anleitungen

```
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICA
TE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE
 KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pc
jqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSF
JuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJ
tjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMu
cxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

Weitere Informationen erhalten Sie unter mysql.rds_import_binlog_ssl_material und TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

> **Note**
>
> Nach dem Ausführen der Prozedur werden die SSL-Informationen in Dateien gespeichert. Um die Dateien später zu löschen, können Sie die mysql.rds_remove_binlog_ssl_material gespeicherte Prozedur ausführen.

So aktivieren Sie die Binärprotokollierung für eine externe MySQL-Datenbank

Datenbank-Engine	Anleitungen

1. Halten Sie mit dieser Befehlszeile den mysql-Service an.

```
sudo service mysqld stop
```

2. Bearbeiten Sie die `my.cnf`-Datei (diese Datei befindet sich üblicherweise unter `/etc`).

```
sudo vi /etc/my.cnf
```

Fügen Sie die Optionen `log_bin` und `server_id` zum Abschnitt `[mysqld]` hinzu. Die Option `log_bin` bietet eine Dateinamenkennung für Binärprotokolldateien. Die Option `server_id` stellt eine eindeutige Kennung für den Server für Quelle-Replica-Beziehungen bereit.

Wenn keine verschlüsselte Replikation erforderlich ist, muss die externe MySQL-Datenbank mit aktivierten Binärprotokollen und aktiviertem SSL gestartet werden.

Dies sind die relevanten Einträge für unverschlüsselte Daten in der Datei `/etc/my.cnf` .

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

Wenn die verschlüsselte Replikation erforderlich ist, muss die externe MySQL-Datenbank mit aktivierten Binärprotokollen und aktiviertem SSL gestartet werden.

Die Einträge in der Datei `/etc/my.cnf` müssen dann die Speicherorte der PEM-Dateien für den MySQL-Datenbankserver enthalten.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
```

Datenbank-Engine	Anleitungen

```
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

Zusätzlich muss die Option `sql_mode` für Ihre MySQL-DB-Instance auf 0 gesetzt oder in Ihrer my.cnf-Datei enthalten sein.

Zeichnen Sie die Position der Binärprotokolle der externen MySQL-Datenbank auf, während Sie mit der Datenbank verbunden sind.

```
mysql> SHOW MASTER STATUS;
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
+------------------+----------+--------------+------------------
+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
 Executed_Gtid_Set |
+------------------+----------+--------------+------------------
+-------------------+
| mysql-bin.000031 |      107 |              |                  |
              |
+------------------+----------+--------------+------------------
+-------------------+
1 row in set (0.00 sec)
```

Weitere Informationen finden Sie unter Setting the replication source configuration in der MySQL-Dokumentation.

3. Starten Sie den mysql-Service.

```
sudo service mysqld start
```

2. Beibehaltung der Binärprotokolle für die Replikationsquelle, bis diese nicht mehr erforderlich sind

Wenn Sie die binäre MySQL-Protokollreplikation verwenden, verwaltet Amazon RDS den Replikationsvorgang nicht. Daher müssen Sie sicherstellen, dass die Binärprotokolldateien an Ihre Replikationsquelle so lange aufbewahrt werden, bis die Änderungen auf das Replica angewandt wurden. Durch diese Pflege können Sie Ihre Quelldatenbank im Fall eines Ausfalls wiederherstellen.

Verwenden Sie folgende Anweisungen zur Beibehaltung von Binärprotokollen für Ihre Datenbank-Engine.

Datenbank-Engine	Anleitungen
Aurora MySQL	So bewahren Sie Binärprotokolle für einen Aurora MySQL-DB-Cluster auf Sie haben keinen Zugriff auf die Binärprotokolldateien eines Aurora-MySQL-DB-Clusters. Daher müssen Sie für die Aufbewahrung der Binärprotokolldateien an Ihrer Replikationsquelle ein ausreichend langes Zeitfenster festlegen, um sicherzustellen, dass die Änderungen auf Ihr Replica angewandt wurden, bevor die Binärprotokolldatei von Amazon RDS gelöscht wird. Sie können diese Binärprotokolldateien in einem Aurora MySQL-DB-Cluster für bis zu 90 Tage aufbewahren. Wenn Sie eine Replikation mit einer MySQL-Datenbank oder einer RDS für MySQL-DB-Instance als Replica einrichten und die Datenbank, für die Sie eine Replica erstellen, sehr groß ist, wählen Sie ein großes Zeitfenster, um Binärprotokolldateien aufzubewahren, bis die erste Kopie der Datenbank in der Replica abgeschlossen ist und die Replica-Verzögerung den Wert 0 erreicht hat. Verwenden Sie zur Festlegung der Aufbewahrungszeit für Binärprotokolldateien die Prozedur mysql.rds_set_configuration und legen Sie einen Konfigurationsparameter für `'binlog retention hours'` zusammen mit der Stundenanzahl für die Aufbewahrung der Binärprotokolldateien im DB-Cluster fest. Der maximal zulässige Wert für Aurora MySQL Version 2.11.0 und höher sowie Version 3 ist 2 160 (90 Tage). Beim folgenden Beispiel wird der Aufbewahrungszeitraum für binäre Protokolle auf 6 Tage festgelegt: `CALL mysql.rds_set_configuration('binlog retention hours', 144);`

Datenbank-Engine	Anleitungen
	Nachdem die Replikation gestartet wurde, können Sie nach dem Ausführen des Befehls `SHOW SLAVE STATUS` (Aurora-MySQL-Version 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) für das Read Replica im Feld `Seconds behind master` überprüfen, ob Änderungen auf Ihr Replica angewandt wurden. Wenn der Wert im Feld `Seconds behind master` 0 ist, gibt es keine Replikatverzögerung. Wenn keine Replica-Verzögerung vorliegt, reduzieren Sie den Aufbewahrungszeitraum für die Binärprotokolldateien, indem Sie den Konfigurationsparameter `binlog retention hours` auf ein kleineres Zeitfenster einstellen. Wenn diese Einstellung nicht angegeben wird, ist der Standardwert für Aurora MySQL 24 (1 Tag). Wenn Sie einen Wert für `'binlog retention hours'` angeben, der den maximal zulässigen Wert überschreitet, dann wird von Aurora MySQL der maximale Wert verwendet.
RDS für MySQL	So bewahren Sie Binärprotokolle einer Amazon-RDS-DB-Instance auf Sie können binäre Protokolldateien auf einer Amazon-RDS-DB-Instance aufbewahren, indem Sie die Stundenanzahl für die Aufbewahrung einstellen, so wie es bereits für einen Aurora-MySQL-DB-Cluster im vorherigen Anschnitt erläutert wurde. Sie können auch Binärprotokolldateien für eine Amazon-RDS-DB-Instance aufbewahren, indem Sie ein Lesereplikat von dieser DB-Instance erstellen. Dieses Lesereplikat ist temporär und lediglich dazu gedacht, Binärprotokolldateien aufzubewahren. Nachdem das Lesereplikat erstellt wurde, rufen Sie die Prozedur mysql.rds_stop_replication für das Lesereplikat auf. Während die Replikation angehalten wird, löscht Amazon RDS keine der Binärprotokolldateien für die Replikationsquelle. Nachdem Sie die Replikation mit Ihrem permanenten Replikat eingerichtet haben, können Sie das Read-Replikat löschen, sobald die Replikat-Verzögerung (Feld `Seconds behind master`) zwischen Ihrer Replikationsquelle und Ihrem permanenten Replikat den Wert 0 erreicht hat.

Datenbank-Engine	Anleitungen
MySQL (extern)	So bewahren Sie Binärprotokolle in einer externen MySQL-Datenbank Da die Binärprotokolldateien in einer externen MySQL-Datenbank nicht von Amazon RDS verwaltet werden, werden diese aufbewahrt, bis Sie sie selbst löschen. Nachdem die Replikation gestartet wurde, können Sie nach dem Ausführen des Befehls `SHOW SLAVE STATUS` (Aurora-MySQL-Version 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) für das Read Replica im Feld `Seconds behind master` überprüfen, ob Änderungen auf Ihr Replica angewandt wurden. Wenn der Wert im Feld `Seconds behind master` 0 ist, gibt es keine Replikatverzögerung. Wenn keine Replica-Verzögerung besteht, können Sie alte Binärprotokolldateien löschen.

3. Erstellen einer Kopie oder eines Dumps Ihrer Replikationsquelle

Sie verwenden einen Snapshot, Klon oder Dump Ihrer Replikationsquelle, um die Baseline-Kopie Ihrer Daten in Ihr Replikat zu kopieren. Dann starten Sie die Replikation ab diesem Punkt.

Verwenden Sie folgende Anleitungen zum Erstellen einer Kopie oder eines Dumps der Replikationsquelle für Ihre Datenbank-Engine.

Datenbank-Engine	Anleitungen
Aurora MySQL	So erstellen Sie eine Kopie eines DB-Clusters von Aurora MySQL Verwenden Sie eine der folgenden Methoden: • Stellen Sie aus einem DB-Cluster-Snapshot wieder her: 1. Erstellen Sie einen DB-Cluster-Snapshot Ihres Amazon Aurora-DB-Clusters. Weitere Informationen finden Sie unter Erstellen eines DB-Cluster-Snapshots. 2. Erstellen Sie ein neues Aurora-DB-Cluster, indem Sie eine Wiederherstellung aus einem kürzlich erstellten DB-Cluster-Snapshot durchführen.

Datenbank-Engine	Anleitungen

Stellen Sie sicher, dass Sie dieselbe DB-Parametergruppe für Ihr wiederher gestelltes DB-Cluster wie für Ihr ursprüngliches DB-Cluster aufbewahren. Dadurch wird sichergestellt, dass die Binärprotokollierung für die Kopie des DB-Clusters aktiviert ist. Weitere Informationen finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.

- Klonen Sie Ihren DB-Cluster. Weitere Informationen finden Sie unter Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster.

So bestimmen Sie den Namen und die Position der Binlog-Datei

Verwenden Sie eine der folgenden Methoden:

- In AWS-Managementkonsole:
  1. Wählen Sie Datenbanken und anschließend die primäre Instance (Schreiber) für Ihren neuen Aurora-DB-Cluster aus, um Details anzuzeigen.
  2. Scrollen Sie zu Recent Events (Aktuelle Ereignisse). Eine Ereignismeldung zeigt den Namen und die Position der Binärprotokolldatei an. Die Ereignismeldung liegt im folgenden Format vor.

     ```
     Binlog position from crash recovery is binlog-file-name  binlog-position
     ```

  3. Speichern Sie den Binärprotokolldateinamen- und die -positionswerte für den Beginn einer Replikation.
- Rufen Sie den AWS CLI-Befehl describe-events wie im folgenden Beispiel auf.

```
aws rds describe-events

{
    "Events": [
        {
            "EventCategories": [],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:
db:sample-restored-instance",
```

Datenbank-Engine	Anleitungen

```
            "Date": "2016-10-28T19:43:46.862Z",
            "Message": "Binlog position from crash recovery is mysql-
bin-changelog.000003 4278",
            "SourceIdentifier": "sample-restored-instance"
        }
    ]
}
```

- Überprüfen Sie das MySQL-Fehlerprotokoll auf die letzte Position der MySQL-Bin log-Datei.

So erstellen Sie einen Dump eines DB-Clusters von Aurora MySQL

Wenn Ihr Replikatziel eine externe MySQL-Datenbank oder eine DB-Instance von RDS für MySQL ist, müssen Sie eine Dump-Datei aus dem Aurora-DB-Cluster erstellen.

Stellen Sie sicher, dass der Befehl `mysqldump` für die Kopie des von Ihnen erstellte n Quell-DB-Clusters ausgeführt wird. Dadurch werden Sperrüberlegungen beim Erstellen des Dumps vermieden. Wenn der Dump direkt auf dem Quell-DB-Cluster erstellt würde, müssten die Quelltabellen gesperrt werden, um gleichzeitige Schreibvo rgänge in diese während des Dump-Vorgangs zu vermeiden.

1. Verbinden Sie sich mit Ihrem DB-Cluster mithilfe eines SQL-Clients.
2. Führen Sie den Befehl `mysqldump` aus. Zum Beispiel:

```
PROMPT> mysqldump --databases database_name --single-transaction
--order-by-primary -r backup.sql -u local_user s -p
```

3. Nachdem Sie die Dump-Datei erstellt haben, können Sie die DB-Cluster-Kopie löschen.

Datenbank-Engine	Anleitungen
RDS für MySQL	So erstellen Sie einen Snapshot Ihrer Amazon-RDS-DB-Instance Erstellen Sie ein Lesereplikat Ihrer Amazon-RDS-DB-Instance. Weitere Informationen finden Sie unter Erstellen einer Read Replica im Amazon Relational Database Service-Benutzerhandbuch. 1. Verbinden Sie sich mit Ihrem Lesereplikat und halten Sie die Replikation mithilfe der Prozedur mysql.rds_stop_replication an. 2. Während die Read Replica angehalten ist, stellen Sie eine Verbindung zur Read Replica her und führen Sie den Befehl `SHOW SLAVE STATUS` (Aurora-MySQL-Version 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) aus. Rufen Sie den aktuellen Namen der Binärprotokolldatei aus dem Feld `Relay_Master_Log_File` und die Position aus dem Feld `Exec_Master_Log_Pos` ab. Speichern Sie diese Werte für den Start einer Replikation. 3. Solange das Lesereplikat im Status Stopped (Angehalten) verweilt, erstellen Sie einen DB-Snapshot dieses Lesereplikats. Weitere Informationen finden Sie unter Erstellen eines DB-Snapshots im Amazon Relational Database Service-Benutzerhandbuch. 4. Löschen Sie das Read Replica.

Datenbank-Engine	Anleitungen

MySQL (extern)	So erstellen Sie einen Dump einer externen MySQL-Datenbank
	1. Bevor Sie einen Dump erstellen, müssen Sie sicherstellen, dass der Speicherort Ihres Binärprotokolls für den Dump auf dem aktuellen Stand der Daten Ihrer Quell-Instance ist. Hierfür müssen Sie als Erstes alle Schreibvorgänge für diese Instance mithilfe des folgenden Befehls anhalten:
	`mysql> FLUSH TABLES WITH READ LOCK;`
	2. Erstellen Sie einen Dump Ihrer MySQL-Datenbank mithilfe des Befehls `mysqldump` , wie im Folgenden gezeigt:
	`PROMPT> sudo mysqldump --databases database_name --master-data=2 --single-transaction \`
	`--order-by-primary -r backup.sql -u local_user -p`
	3. Nachdem Sie den Dump erstellt haben, entsperren Sie die Tabellen in Ihrer MySQL-Datenbank mit dem folgenden Befehl:
	`mysql> UNLOCK TABLES;`

4. Laden des Dumps in das Replikatziel (falls erforderlich)

Wenn Sie Daten aus einem Dump einer Amazon-RDS-externen MySQL-Datenbank laden möchten, können Sie eine EC2-Instance erstellen, um die Dump-Dateien dorthin zu kopieren. Anschließend können Sie die Daten aus dieser EC2-Instance in den DB-Cluster oder die DB-Instance laden. Bei Verwendung dieses Ansatzes können Sie die Dumpdateien komprimieren, bevor Sie sie in eine EC2-Instance kopieren, um Netzwerkkosten zu minimieren, die in Verbindung mit dem Kopieren von Dateien nach Amazon RDS entstehen. Sie können die Dumpdatei oder -dateien auch verschlüsseln, um sie beim Übermitteln im Netzwerk zu schützen.

> **Note**
>
> Wenn Sie einen neuen DB-Cluster von Aurora MySQL als Replikatziel erstellen, müssen Sie keine Dump-Datei laden:

- Sie können aus einem DB-Cluster-Snapshot wiederherstellen, um einen neuen DB-Cluster zu erstellen. Weitere Informationen finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.
- Sie können Ihren Quell-DB-Cluster klonen, um einen neuen DB-Cluster zu erstellen. Weitere Informationen finden Sie unter Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster.
- Sie können die Daten aus einem DB-Instance-Snapshot zu einem neuen DB-Cluster migrieren. Weitere Informationen finden Sie unter Migrieren von Daten zu einem Amazon Aurora MySQL-DB-Cluster.

Verwenden Sie folgende Anleitungen, um den Dump Ihrer Replikationsquelle in das Replikatziel für Ihre Datenbank-Engine zu laden.

Datenbank-Engine	Anleitungen

Aurora MySQL	So laden Sie einen Dump in einen DB-Cluster von Aurora MySQL
	1. Kopieren Sie die Ausgabe des Befehls `mysqldump` von der Replikationsquelle an einen Speicherort, der auch eine Verbindung zum Aurora MySQL-DB-Cluster herstellen kann.
	2. Verbinden Sie sich mithilfe des `mysql`-Befehls mit Ihrem Aurora MySQL-DB-Cluster. Im Folgenden wird ein Beispiel gezeigt.
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. Führen Sie in der `mysql`-Eingabeaufforderung den Befehl `source` aus und geben Sie den Namen Ihrer Datenbank-Dumpdatei ein, um die Daten in den Aurora MySQL-DB-Cluster zu laden, zum Beispiel:
	`mysql> source backup.sql;`

RDS für MySQL	So laden Sie einen Dump in eine DB-Instance von Amazon RDS

Datenbank-Engine	Anleitungen

	1. Kopieren Sie die Ausgabe des Befehls `mysqldump` von Ihrer Replikationsquelle an einen Speicherort, der auch eine Verbindung mit Ihrer MySQL-DB-Instance herstellen kann.
	2. Verbinden Sie sich mit Ihrer MySQL-DB-Instance mithilfe des `mysql`-Befehls. Im Folgenden wird ein Beispiel gezeigt.
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. Führen Sie in der `mysql`-Eingabeaufforderung den Befehl `source` aus und geben Sie den Namen Ihrer Datenbank-Dumpdatei mit, um deren Daten in die My-DB-Instance zu laden, zum Beispiel:
	`mysql> source backup.sql;`

MySQL (extern)	So laden Sie einen Dump in eine externe MySQL-Datenbank
	Sie können keinen DB-Snapshot oder DB-Cluster-Snapshot in eine externe MySQL-Datenbank laden. Stattdessen müssen Sie die Ausgabe des `mysqldump` -Befehls verwenden.
	1. Kopieren Sie die Ausgabe des Befehls `mysqldump` von der Replikationsquelle an einen Speicherort, der auch eine Verbindung zur MySQL-Datenbank herstellen kann.
	2. Verbinden Sie sich mit Ihrer MySQL-Datenbank mithilfe des `mysql`-Befehls. Im Folgenden wird ein Beispiel gezeigt.
	`PROMPT> mysql -h host_name -port=3306 -u db_master_user -p`
	3. Führen Sie an der `mysql`-Eingabeaufforderung den Befehl `source` aus und übergeben Sie den Namen der Datenbank-Dumpdatei, deren Daten in die MySQL-Datenbank geladen werden sollen. Im Folgenden wird ein Beispiel gezeigt.
	`mysql> source backup.sql;`

5. Erstellen Sie einen Replikationsbenutzer auf Ihrer Replikationsquelle

Erstellen Sie eine Benutzer-ID an der Quelle, die ausschließlich für die Replikation verwendet wird. Das folgende Beispiel bezieht sich auf RDS für MySQL oder externe MySQL-Quelldatenbanken.

```
mysql> CREATE USER 'repl_user'@'domain_name' IDENTIFIED BY 'password';
```

Für Aurora-MySQL-Quelldatenbanken ist der DB-Cluster-Parameter `skip_name_resolve` auf 1 (0N) gesetzt und kann nicht geändert werden. Sie müssen also eine IP-Adresse für den Host anstelle eines Domainnamens verwenden. Weitere Informationen finden Sie unter skip_name_resolve in der MySQL-Dokumentation.

```
mysql> CREATE USER 'repl_user'@'IP_address' IDENTIFIED BY 'password';
```

Der Benutzer benötigt die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE`. Gewähren Sie dem Benutzer diese Berechtigungen.

Wenn die Replikation verschlüsselt durchgeführt werden muss, fordern Sie für den Replikationsbenutzer eine SSL-Verbindung an. Sie können beispielsweise mit einer der folgenden Anweisungen SSL-Verbindungen für das Benutzerkonto `repl_user` erforderlich machen.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'IP_address';
```

```
GRANT USAGE ON *.* TO 'repl_user'@'IP_address' REQUIRE SSL;
```

> **Note**
>
> Wenn `REQUIRE SSL` nicht angegeben wird, kann die Replikationsverbindung ohne entsprechende Meldung auf eine unverschlüsselte Verbindung zurückgesetzt werden.

6. Aktivieren der Replikation für das Replikat-Ziel

Bevor Sie die Replikation aktivieren, empfehlen wir, manuell einen Snapshot des Aurora-MySQL-DB-Clusters oder des RDS für MySQL-DB-Instance Replica-Ziels zu erstellen. Wenn ein Problem auftritt und Sie den Replikationsvorgang mit dem DB-Cluster oder dem DB-Instance-Replica-Ziel wieder einrichten müssen, können Sie das DB-Cluster bzw. die DB-Instance aus diesem Snapshot wiederherstellen, anstatt erneut Daten in das Replica-Ziel zu importieren.

Verwenden Sie folgende Anweisungen zum Aktivieren der Replikation für Ihre Datenbank-Engine.

Datenbank-Engine	Anleitungen

Aurora MySQL	So aktivieren Sie die Replikation aus einem Aurora-MySQL-DB-Cluster
	1. Suchen Sie den Startpunkt für die Replikation. Sie benötigen den Namen der Binärprotokolldatei und die Binärprotokollposition.
	Wenn Ihr DB-Cluster-Replikatziel auf Folgendem erstellt wurde:
	• DB-Cluster-Snapshot oder -Klon – Rufen Sie den Namen und die Position der Binlog-Datei aus den letzten Ereignissen für Ihren neuen DB-Cluster ab, wie unter 3. Erstellen einer Kopie oder eines Dumps Ihrer Replikationsquelle gezeigt.
	• DB-Snapshot – Sie haben den Binärprotokoll-Dateinamen und die -position aus dem Befehl `SHOW SLAVE STATUS` (Aurora MySQL Version 2) oder `SHOW REPLICA STATUS` (Aurora MySQL Version 3) abgerufen, als Sie den Snapshot Ihrer Replikationsquelle erstellt haben.
	2. Stellen Sie eine Verbindung mit dem DB-Cluster her und rufen Sie die folgenden Prozeduren auf, um eine Replikation mit Ihrer Replikationsquelle unter Verwendung des Namens und des Speicherorts der Binärprotokolldatei aus dem vorherigen Schritt zu starten:
	• mysql.rds_set_external_source (Aurora-MySQL-Version 3)
	• mysql.rds_set_external_master (Aurora-MySQL-Version 2)
	• mysql.rds_start_replication (alle Versionen)
	Das folgende Beispiel gilt für Aurora MySQL Version 3.
	`CALL mysql.rds_set_external_source ('mydbinstance.123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);`
	`CALL mysql.rds_start_replication;`
	Zur Verwendung der SSL-Verschlüsselung legen Sie den endgültigen Wert auf 1 statt auf `0` fest.

Datenbank-Engine	Anleitungen

RDS für MySQL	So aktivieren Sie die Replikation aus einer Amazon-RDS-DB-Instance
	1. Wenn das DB-Instance-Replica-Ziel aus einem DB-Snapshot erstellt wurde, benötigen Sie den Namen der Binärprotokolldatei und die Position in der Datei, ab der die Replikation durchgeführt werden soll. Sie haben diese Werte aus dem Befehl `SHOW SLAVE STATUS` (Aurora-MySQL-Version 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) abgerufen, als Sie den Snapshot Ihrer Replikationsquelle erstellt haben.
	2. Stellen Sie eine Verbindung zur DB-Instance her und rufen Sie die Prozeduren mysql.rds_set_external_master (Aurora-MySQL-Version 2) odermysql.rds_set_external_source (Aurora-MySQL-Version 3) und mysql.rds_start_replication auf, um eine Replikation mit Ihrer Replikationsquelle zu starten. Verwenden Sie den Namen und den Speicherort der Binärprotokolldatei aus dem vorherigen Schritt. Im Folgenden wird ein Beispiel gezeigt.
	`CALL mysql.rds_set_external_master ('mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);`
	`CALL mysql.rds_start_replication;`
	Zur Verwendung der SSL-Verschlüsselung legen Sie den endgültigen Wert auf 1 statt auf `0` fest.

Datenbank-Engine	Anleitungen

MySQL (extern)	So aktivieren Sie die Replikation aus einer externen MySQL-Datenbank
	1. Erhalten Sie den Namen und die Position der Binärprotokolldatei, die der Ausgangspunkt für eine Replikation sind. Sie haben diese Werte aus dem Befehl `SHOW SLAVE STATUS` (Aurora-MySQL-Version 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) abgerufen, als Sie den Snapshot Ihrer Replikationsquelle erstellt haben. Wenn das externe MySQL-Replica-Ziel durch den `mysqldump` -Befehl mit der Option `--master-data=2` mit Daten gefüllt wurde, sind der Name und die Position der Binärprotokolldatei in den ausgegebenen Informationen enthalten. Im Folgenden wird ein Beispiel gezeigt.
	`-- -- Position to start replication or point-in-time recovery from -- -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;`
	2. Stellen Sie eine Verbindung zum externen MySQL-Replica-Ziel her und führen Sie die Befehle `CHANGE MASTER TO` und `START SLAVE` (Aurora-MySQL-Version 2) oder `START REPLICA` (Aurora-MySQL-Version 3) aus, um eine Replikation mit der Replikationsquelle mithilfe des Namens und der Position der Binärprotokolldatei aus dem vorherigen Schritt zu starten. Beispiel:
	`CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; -- And one of these statements depending on your engine version: START SLAVE; -- Aurora MySQL version 2 START REPLICA; -- Aurora MySQL version 3`

Wenn die Replikation fehlschlägt, kann dies zu einem starken Anstieg der unbeabsichtigten I/O auf dem Replikat führen, was die Leistung beeinträchtigen kann. Wenn die Replikation fehlschlägt oder nicht mehr benötigt wird, können Sie das gespeicherte Verfahren mysql.rds_reset_external_master (Aurora-MySQL-Version 2) oder mysql.rds_reset_external_source (Aurora-MySQL-Version 3) zum Entfernen der Replikationskonfiguration durchführen.

Festlegen einer Position zum Stoppen der Replikation zu einer Read Replica

In der Aurora MySQL-Version 3.04 und höher können Sie die Replikation starten und dann an einem bestimmten Ort in der Binär-Protokolldatei mit der gespeicherten mysql.rds_start_replication_until (Aurora-MySQL-Version 3)-Prozedur anhalten.

Starten Sie die Replikation für ein Lesereplikat und stoppen Sie die Replikation an einer bestimmten Position wie folgt:

1. Verwenden Sie einen MySQL-Client, um als Master-Benutzer eine Verbindung zum Replikat-DB-Cluster von Aurora MySQL herzustellen.
2. Führen Sie die gespeicherte Prozedur mysql.rds_start_replication_until (Aurora-MySQL-Version 3) aus.

   Das folgende Beispiel initiiert die Replikation und repliziert die Änderungen, bis die Position `120` in der Binärprotokolldatei `mysql-bin-changelog.000777` erreicht wird. Beispiel: In einem Szenario der Notfallwiederherstellung liegt die Position `120` unmittelbar vor der Katastrophe.

   ```
   call mysql.rds_start_replication_until(
     'mysql-bin-changelog.000777',
     120);
   ```

Die Replikation stoppt automatisch, sobald der Stopppunkt erreicht ist. Das folgende RDS-Ereignis wird generiert: `Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure`.

Wenn Sie die GTID-basierte Replikation verwenden, nutzen Sie die gespeicherte Prozedur mysql.rds_start_replication_until_gtid (Aurora-MySQL-Version 3) anstelle der gespeicherten Prozedur mysql.rds_start_replication_until (Aurora-MySQL-Version 3). Weitere Informationen zu GTID-basierten Replikationen finden Sie unter Verwenden der GTID-basierten Replikation.

7. Überwachen Ihres Replikats

Wenn Sie eine MySQL-Replikation mit einem Aurora MySQL-DB-Cluster eingerichtet haben, müssen Sie Failover-Ereignisse für den Aurora MySQL-DB-Cluster überwachen, wenn dieser ein Replica-Ziel ist. Im Falle eines Failovers kann dann das DB-Cluster (das Replica-Ziel) auf einem neuen Host mit einer anderen Netzwerkadresse erneut erstellt werden. Weitere Informationen zum Überwachen von Failover-Ereignissen finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

Sie können außerdem überwachen, wie weit das Replica-Ziel hinter der Replikationsquelle zurückliegt, indem Sie eine Verbindung zum Replica-Ziel herstellen und den Befehl `SHOW SLAVE STATUS` (Aurora-MySQL-Version 2) oder `SHOW REPLICA STATUS` (Aurora-MySQL-Version 3) ausführen. In der Befehlsausgabe wird im Feld `Seconds Behind Master` angezeigt, wie weit das Replica-Ziel hinter der Quelle zurückliegt.

> **Important**
>
> Wenn Sie Ihren DB-Cluster aktualisieren und eine benutzerdefinierte Parametergruppe angeben, müssen Sie den Cluster nach Abschluss des Upgrades unbedingt manuell neu starten. Dadurch verwendet der Cluster Ihre neuen benutzerdefinierten Parametereinstellungen und die Binlog-Replikation wird neu gestartet.

Synchronisierung von Passwörtern zwischen Replikationsquelle und -ziel

Wenn Sie Benutzerkonten und Kennwörter auf der Replikationsquelle mithilfe von SQL-Anweisungen ändern, werden diese Änderungen automatisch auf das Replikationsziel repliziert.

Wenn Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API verwenden, um das Hauptkennwort auf der Replikationsquelle zu ändern, werden diese Änderungen nicht automatisch auf das Replikationsziel repliziert. Wenn Sie den Master-Benutzer und das Master-Kennwort zwischen dem Quell- und dem Zielsystem synchronisieren möchten, müssen Sie die gleiche Änderung auf dem Replikationsziel selbst vornehmen.

Anhalten einer binären Protokollreplikation für Aurora MySQL

Folgen Sie diesen im folgenden Verlauf detailliert beschriebenen Schritten, um eine binäre Protokollreplikation mit einer MySQL-DB-Instance, einer externen MySQL-Datenbank oder einem anderen Aurora-DB-Cluster anzuhalten.

1. Anhalten der binären Protokollreplikation für das Replikat-Ziel

2. Deaktivieren der Binärprotokollierung für die Replikationsquelle

1. Anhalten der binären Protokollreplikation für das Replikat-Ziel

Im Folgenden finden Sie die Anweisungen zum Anhalten der binären Protokollreplikation für Ihre Datenbank-Engine.

Datenbank-Engine	Anleitungen
Aurora MySQL	So können Sie die binären Protokollreplikation in einem Aurora MySQL-DB-Cluster-Replica-Ziel anhalten Stellen Sie eine Verbindung mit dem als Replica-Ziel verwendeten Aurora-DB-Cluster her und rufen Sie die Prozedur mysql.rds_stop_replication auf.
RDS für MySQL	So können Sie eine binären Protokollreplikation in einer Amazon-RDS-DB-Instance anhalten Stellen Sie eine Verbindung mit der als Replica-Ziel verwendeten RDS-DB-Instance her und rufen Sie die Prozedur mysql.rds_stop_replication auf.
MySQL (extern)	So halten Sie die binären Protokollreplikation für eine externe MySQL-Datenbank an Stellen Sie eine Verbindung mit der MySQL-Datenbank her und führen Sie den Befehl `STOP SLAVE` (Version 5.7) oder `STOP REPLICA` (Version 8.0) aus.

2. Deaktivieren der Binärprotokollierung für die Replikationsquelle

Im Folgenden finden Sie die Anweisungen zum Deaktivieren der Binärprotokollierung an der Replikationsquelle für Ihre Datenbank-Engine.

Datenbank-Engine	Anleitungen
Aurora MySQL	So deaktivieren Sie die Binärprotokollierung für einen Amazon-Aurora-DB-Cluster 1. Stellen Sie eine Verbindung mit dem als Replikationsquelle verwendeten Aurora-DB-Cluster her.

Datenbank-Engine	Anleitungen
	2. Verwenden Sie das Verfahren mysql.rds_set_configuration und geben Sie den Konfigurationsparameter `binlog retention hours` mit dem Wert `NULL` an, wie im folgenden Beispiel gezeigt. `CALL mysql.rds_set_configuration('binlog retention hours', NULL);` **Note** Sie können den Wert `0` nicht für `binlog retention hours` verwenden. 3. Stellen Sie den Parameter `binlog_format` für die Replikationsquelle auf `OFF` ein. Der`binlog_format` -Parameter befindet sich in der benutzerdefinierten Parametergruppe, die Ihrem DB-Cluster zugeordnet ist. Nachdem Sie den Parameterwert `binlog_format` geändert haben, starten Sie Ihren DB-Cluster neu, damit die Änderungen übernommen werden. Weitere Informationen erhalten Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter und Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.
RDS für MySQL	So deaktivieren Sie die Binärprotokollierung für eine Amazon-RDS-DB-Instance Sie können die Binärprotokollierung nicht direkt für eine Amazon-RDS-DB-Instance deaktivieren, aber Sie können diese Funktion deaktivieren, indem Sie Folgendes tun: 1. Deaktivieren Sie die automatischen Backups für die DB-Instance. Sie können die automatischen Backups deaktivieren, indem Sie eine bestehende DB-Instance ändern und Backup Retention Period (Aufbewahrungszeitraum für Sicherungen) auf den Wert 0 setzen. Weitere Informationen finden Sie unter Ändern einer Amazon-RDS-DB-Instance und unter Arbeiten mit Sicherungen im Amazon Relational Database Service-Benutzerhandbuch. 2. Löschen Sie alle Lesereplikate für die DB-Instance. Weitere Informationen finden Sie unter Arbeiten mit Read Replicas von MariaDB, MySQL und PostgreSQL DB-Instances im Amazon Relational Database Service-Benutzerhandbuch.

Datenbank-Engine	Anleitungen
MySQL (extern)	So deaktivieren Sie die Binärprotokollierung für eine externe MySQL-Datenbank Verbinden Sie sich mit der MySQL-Datenbank und rufen Sie den Befehl `STOP REPLICATION` auf. 1. Halten Sie aus einer Command-Shell den Service `mysqld` an. ``` sudo service mysqld stop ``` 2. Bearbeiten Sie die `my.cnf`-Datei (diese Datei befindet sich üblicherweise unter `/etc`). ``` sudo vi /etc/my.cnf ``` Löschen Sie die Optionen `log_bin` und `server_id` aus dem Abschnitt `[mysqld]`. Weitere Informationen finden Sie unter Setting the replication source configuration in der MySQL-Dokumentation. 3. Starten Sie den mysql-Service. ``` sudo service mysqld start ```

Skalieren von Lesevorgängen für Ihre MySQL-Datenbank mit Amazon Aurora

Sie können Amazon Aurora mit Ihrer MySQL-DB-Instance verwenden, um die Möglichkeiten der Skalierung von Lesevorgängen von Amazon Aurora zu nutzen und den Lese-Workload für Ihre MySQL-DB-Instance zu erweitern. Erstellen Sie einen Amazon-Aurora MySQL-DB-Cluster und machen Sie ihn zum Read Replica Ihrer MySQL-DB-Instance, um Aurora für die Skalierung von Lesevorgängen Ihrer MySQL-DB-Instance zu verwenden. Dies gilt für eine RDS für MySQL-DB-Instance oder eine MySQL-Datenbank, die außerhalb von Amazon RDS ausgeführt wird.

Weitere Informationen zum Erstellen eines Amazon Aurora-DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Wenn Sie eine Replikation zwischen Ihrer MySQL-DB-Instance und Ihrem Amazon Aurora-DB-Cluster einrichten, folgen Sie sicherheitshalber bitte diesen Anweisungen:

- Verwenden Sie die Amazon Aurora-DB-Cluster-Endpunkt-Adresse, wenn Sie Ihr Amazon Aurora MySQL-DB-Cluster referenzieren. Bei einem Failover verwendet die Aurora-Replica, die zur primären Instance for das Aurora MySQL-DB-Cluster hochgestuft wird, weiterhin die DB-Cluster-Endpunkt-Adresse.
- Bewahren Sie die Binärprotokolle auf Ihrer Schreiber-Instance auf, bis Sie sichergestellt haben, dass sie auf das Aurora Replica angewandet wurden. Durch das Aufbewahren können Sie sicherstellen, dass Sie Ihre Schreiber-Instance im Fall eines Ausfalls wiederherstellen können.

Important

Wenn Sie eine von Ihnen verwaltete Replikation verwenden, sind Sie für die Überwachung und Lösung jeglicher auftretender Probleme bei der Replikation verantwortlich. Weitere Informationen finden Sie unter Diagnose und Lösung bei Verzögerungen zwischen Read Replicas (Lesereplikaten).

Note

Die erforderlichen Berechtigungen zum Starten einer Replikation in einem Amazon-Aurora-MySQL-DB-Cluster sind beschränkt und für Ihren Amazon-RDS-Hauptbenutzer nicht verfügbar. Aus diesem Grund müssen Sie die Prozedurenmysql.rds_set_external_master (Aurora-MySQL-Version 2), mysql.rds_set_external_source (Aurora-MySQL-Version 3) und mysql.rds_start_replication verwenden, um eine Replikation zwischen Ihrem Aurora-MySQL-DB-Cluster und Ihrer MySQL-DB-Instance einzurichten.

Starten der Replikation zwischen einer externen Quell-Instance und einem MySQL-DB-Cluster

1. Legen Sie die Quell-MySQL-DB-Instance als schreibgeschützt fest:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. Führen Sie den Befehl `SHOW MASTER STATUS` in der Quell-MySQL-DB-Instance aus, um den Speicherort des Binärprotokolls zu bestimmen. Sie erhalten eine Ausgabe, ähnlich der im folgenden Beispiel:

```
File                          Position
-----------------------------------
 mysql-bin-changelog.000031      107
-----------------------------------
```

3. Kopieren Sie die Datenbank mithilfe von aus der externen MySQL-DB-Instance in das Amazon Aurora MySQL-DB-Cluster `mysqldump`. Bei sehr großen Datenbanken empfiehlt es sich, das Verfahren zu verwenden, das im Abschnitt zum Importieren von Daten in eine Datenbank von Amazon RDS für MySQL mit reduzierter Ausfallzeit im Benutzerhandbuch zu Amazon Relational Database Service beschrieben ist.

   Für Linux, macOS oder Unix:

```
mysqldump \
    --databases <database_name> \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -p local_password | mysql \
        --host aurora_cluster_endpoint_address \
        --port 3306 \
        -u RDS_user_name \
        -p RDS_password
```

   Für Windows:

```
mysqldump ^
    --databases <database_name> ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -p local_password | mysql ^
        --host aurora_cluster_endpoint_address ^
        --port 3306 ^
        -u RDS_user_name ^
```

```
        -p RDS_password
```

> **Note**
>
> Stellen Sie sicher, dass kein Leezeichen zwischen der Option `-p` und dem eingegebenen Passwort vorhanden ist.

Verwenden Sie die Optionen `--host`, `--user (-u)` `--port` und `-p` im Befehl `mysql`, um den Host-Namen, Benutzernamen, Port und das Passwort für die Verbindung mit Ihrem Aurora-DB-Cluster anzugeben. Der Host-Name ist der DNS-Name aus dem Amazon Aurora-DB-Cluster-Endpunkt, z. B, `mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com`. Sie finden den Endpunktwert in den Cluster-Details in der Amazon-RDS-Managementkonsole.

4. Legen Sie die Quell-MySQL-DB-Instance als wieder beschreibbar fest:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Weitere Informationen zum Erstellen von Backups zur Verwendung mit der Replikation finden Sie unter Backing up a source or replica by making it read only in der MySQL-Dokumentation.

5. Fügen Sie in der Amazon-RDS-Management Console die IP-Adresse des Servers, der die Quell-MySQL-Datenbank hostet, zu der VPC-Sicherheitsgruppe für das Amazon Aurora-DB-Cluster hinzu. Weitere Informationen zum Ändern einer VPC-Sicherheitsgruppe finden Sie unter Sicherheitsgruppen für Ihre VPC im Amazon Virtual Private Cloud-Benutzerhandbuch.

   Es könnte sein, dass Sie Ihr lokales Netzwerk so konfigurieren müssen, dass es Verbindungen von der IP-Adresse Ihres Amazon Aurora-DB-Clusters zulässt, damit Sie mit Ihrer Quell-MySQL-Datenbank kommunizieren können. Verwenden Sie den Befehl `host`, um die IP-Adresse des Amazon Aurora-DB-Clusters herauszufinden.

```
host aurora_endpoint_address
```

   Der Hostname ist der DNS-Name aus dem Amazon Aurora-DB-Cluster-Endpunkt.

6. Verbinden Sie sich mithilfe eines Clients Ihrer Wahl mit der externen MySQL-Instance und erstellen Sie einen MySQL-Benutzer, der für die Replikation verwendet werden soll. Dieses

Konto wird ausschließlich für die Replikation verwendet und muss auf Ihre Domäne beschränkt sein, um die Sicherheit zu erhöhen. Im Folgenden wird ein Beispiel gezeigt.

```
CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
```

7. Erteilen Sie Ihrem Replikationsbenutzer für die externe MySQL-Instance die Sonderrechte `REPLICATION CLIENT` und `REPLICATION SLAVE`. Erteilen Sie beispielsweise die Sonderrechte `REPLICATION CLIENT` und `REPLICATION SLAVE` in allen Datenbank für den '`repl_user`'-Benutzer für Ihre Domäne, mit dem folgenden Befehl.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com'
       IDENTIFIED BY 'password';
   ```

8. Machen Sie einen manuellen Snapshot des Aurora MySQL-DB-Clusters, der das Read Replica sein soll, bevor Sie eine Replikation einrichten. Wenn Sie erneut eine Replikation mit dem DB-Cluster als Read Replica einrichten müssen, können Sie das Aurora MySQL-DB-Cluster aus diesem Snapshot wiederherstellen, anstatt die Daten aus Ihrer-MySQL-DB-Instance in ein neues Aurora MySQL-DB-Cluster importieren zu müssen.
9. Legen Sie das Amazon Aurora-DB-Cluster als Replica fest. Verbinden Sie sich als Master-Benutzer mit dem DB-Cluster von Amazon Aurora und bestimmen Sie die MySQL-Quelldatenbank mithilfe der Prozeduren mysql.rds_set_external_master (Aurora-MySQL-Version 2) oder mysql.rds_set_external_source (Aurora-MySQL-Version 3) und mysql.rds_start_replication als Replikationsquelle.

   Verwenden Sie den Namen und die Position der Binlog-Datei, die Sie in Schritt 2 festgelegt haben. Im Folgenden wird ein -Beispiel gezeigt.

   ```
   For Aurora MySQL version 2:
   CALL mysql.rds_set_external_master ('mymasterserver.example.com', 3306,
       'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);

   For Aurora MySQL version 3:
   CALL mysql.rds_set_external_source ('mymasterserver.example.com', 3306,
       'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);
   ```

10. Rufen Sie auf dem Amazon-Aurora-DB-Cluster die mysql.rds_start_replication-Prozedur auf, um die Replikation zu starten.

    ```
    CALL mysql.rds_start_replication;
    ```

Nachdem Sie die Replikation zwischen Ihrer Quell-MySQL-DB-Instance und Ihrem Amazon Aurora-DB-Cluster erneut eingerichtet haben, können Sie Aurora-Replicas zu Ihrem Amazon Aurora-DB-Cluster hinzufügen. Sie können sich anschließend mit dem Aurora Replicas verbinden, um die Lesevorgänge Ihrer Daten zu skalieren. Weitere Informationen über das Erstellen einer Aurora-Replica finden Sie unter Hinzufügen von Aurora-Replicas zu einem DB-Cluster.

Optimieren einer binären Protokollreplikation für Aurora MySQL

Im Folgenden erfahren Sie, wie Sie die Leistung der binären Protokollreplikation optimieren und damit zusammenhängende Probleme in Aurora MySQL beheben können.

> **Tip**
>
> In dieser Erörterung wird davon ausgegangen, dass Sie mit dem Mechanismus der binären MySQL-Protokollreplikation und dessen Funktionsweise vertraut sind. Hintergrundinformationen finden Sie unter Replication Implementation (Implementierung der Replikation) in der MySQL-Dokumentation.

Multithread-Replikation für binäre Protokolle

Bei der Multithread-Replikation für binäre Protokolle liest ein SQL-Thread Ereignisse aus dem Relay-Log und stellt sie in die Warteschlange, damit SQL-Worker-Threads angewendet werden können. Die SQL-Worker-Threads werden vom Koordinator-Thread verwaltet. Die binären Protokollereignisse werden nach Möglichkeit parallel angewendet. Der Grad der Parallelität hängt von Faktoren wie Version, Parametern, Schemadesign und Workload-Eigenschaften ab.

Die Multithread-Replikation für binäre Protokolle wird in Aurora-MySQL-Version 3 und Aurora-MySQL-Version 2.12.1 und höher unterstützt. Damit ein Multithread-Replikat Binlog-Ereignisse effizient parallel verarbeiten kann, müssen Sie die Quelle für die Multithread-Replikation für binäre Protokolle konfigurieren, und die Quelle muss eine Version verwenden, die die Parallelitätsinformationen in ihren Binärprotokolldateien enthält.

Wenn eine DB-Instance von Aurora MySQL für die Verwendung der binären Protokollreplikation konfiguriert ist, verwendet die Replikat-Instance standardmäßig die Single-Thread-Replikation für Aurora-MySQL-Versionen niedriger als 3.04. Zum Aktivieren der Multithread-Replikation aktualisieren Sie den Parameter `replica_parallel_workers` in Ihrer benutzerdefinierten Parametergruppe auf einen Wert größer als 1.

Für Aurora-MySQL-Version 3.04 und höher erfolgt die Replikation standardmäßig mit mehreren Threads, wobei `replica_parallel_workers` auf 4 gesetzt ist. Sie können diesen Parameter in Ihrer benutzerdefinierten Parametergruppe ändern.

Zum Erhöhen der Ausfallsicherheit Ihrer Datenbank gegen unerwartete Unterbrechungen empfehlen wir, die GTID-Replikation in der Quelle und GTIDs im Replikat zuzulassen. Zum Zulassen der GTID-Replikation setzen Sie `gtid_mode` sowohl in der Quelle als auch im Replikat auf `ON_PERMISSIVE`. Weitere Informationen zu GTID-basierten Replikationen finden Sie unter Verwenden der GTID-basierten Replikation.

Die folgenden Konfigurationsoptionen helfen Ihnen bei der Optimierung der Multithread-Replikation. Informationen zur Verwendung finden Sie unter Replikations- und binäre Protokollierungsoptionen und -Variablen im MySQL-Verweishandbuch. Weitere Informationen zur Multithread-Replikation finden Sie im MySQL-Blog Verbesserung des parallelen Anwenders durch schreibsatzbasierte Abhängigkeitsverfolgung.

Optimale Parameterwerte hängen von mehreren Faktoren ab. Beispielsweise wird die Leistung für die Binärprotokollreplikation von Ihren Datenbank-Workload-Merkmalen und der DB-Instance-Klasse beeinflusst, auf der das Replikat ausgeführt wird. Daher empfehlen wir Ihnen, alle Änderungen an diesen Konfigurationsparametern gründlich zu testen, bevor Sie neue Parametereinstellungen auf eine Produktions-Instance anwenden:

- `binlog_format recommended value` – festlegen auf `ROW`
- `binlog_group_commit_sync_delay`
- `binlog_group_commit_sync_no_delay_count`
- `binlog_transaction_dependency_history_size`
- `binlog_transaction_dependency_tracking` – empfohlener Wert ist `WRITESET`
- `replica_preserve_commit_order`
- `replica_parallel_type` – empfohlener Wert ist `LOGICAL_CLOCK`
- `replica_parallel_workers`
- `replica_pending_jobs_size_max`
- `transaction_write_set_extraction` – empfohlener Wert ist `XXHASH64`

Ihr Schema und Ihre Workload-Eigenschaften sind Faktoren, die sich parallel auf die Replikation auswirken. Zu den häufigsten Faktoren zählen folgende:

- Fehlen von Primärschlüsseln – RDS kann keine Schreibsatzabhängigkeit für Tabellen ohne Primärschlüssel einrichten. Mit dem `ROW`-Format kann eine einzelne mehrzeilige Anweisung mit einem einzigen vollständigen Tabellenscan in der Quelle ausgeführt werden. Dies führt jedoch zu einem vollständigen Tabellenscan pro Zeile, die im Replikat geändert wurde. Das Fehlen von Primärschlüsseln verringert den Replikationsdurchsatz erheblich.
- Vorhandensein von Fremdschlüsseln – Wenn Fremdschlüssel vorhanden sind, kann Amazon RDS die Schreibsatzabhängigkeit nicht für die Parallelität der Tabellen mit der FK-Beziehung verwenden.
- Größe der Transaktionen – Wenn sich eine einzelne Transaktion über Dutzende oder Hunderte von Megabyte oder Gigabyte erstreckt, verbringen der Koordinator-Thread und einer der Worker-Threads möglicherweise viel Zeit damit, nur diese Transaktion zu verarbeiten. Während dieser Zeit können alle anderen Worker-Threads inaktiv bleiben, nachdem sie die Verarbeitung ihrer vorherigen Transaktionen abgeschlossen haben.

In Aurora-MySQL-Version 3.06 und höher können Sie die Leistung für binäre Protokollreplikate verbessern, wenn Sie Transaktionen für große Tabellen mit mehr als einem sekundären Index replizieren. Diese Funktion führt einen Thread-Pool ein, um sekundäre Indexänderungen parallel auf ein Binlog-Replikat anzuwenden. Die Funktion wird durch den DB-Cluster-Parameter `aurora_binlog_replication_sec_index_parallel_workers` gesteuert, der die Gesamtzahl der parallelen Threads steuert, die für die Anwendung der sekundären Indexänderungen verfügbar sind. Der Parameter ist standardmäßig auf `0` (deaktiviert) eingestellt. Für die Aktivierung dieser Funktion ist kein Instance-Neustart erforderlich. Zum Aktivieren dieser Funktion beenden Sie die laufende Replikation, legen die gewünschte Anzahl paralleler Worker-Threads fest und starten die Replikation dann erneut.

Optimieren einer Binlog-Replikation

In Aurora MySQL 2.10 und höher wendet Aurora automatisch eine Optimierung an, die als Binlog-I/O-Cache bekannt ist, auf die Binärlog-Replikation. Durch Zwischenspeichern der zuletzt festgeschriebenen Binlog-Ereignisse soll diese Optimierung die Leistung des Binlog-Dump-Threads verbessern und gleichzeitig die Auswirkungen auf Vordergrundtransaktionen auf der Binlog-Quell-Instance begrenzen.

> **Note**
>
> Dieser für diese Funktion verwendete Speicher ist unabhängig von der `binlog_cache` MySQL-Einstellung.

Diese Funktion gilt nicht für Aurora DB-Instances, die die `db.t2`- und `db.t3`-Instance-Klassen verwenden.

Sie müssen keine Konfigurationsparameter anpassen, um diese Optimierung zu aktivieren. Insbesondere wenn Sie den Konfigurationsparameter `aurora_binlog_replication_max_yield_seconds` in früheren Aurora-MySQL-Versionen auf einen Wert ungleich null eingestellt hatten, setzen Sie ihn für derzeit verfügbare Versionen auf null zurück.

Die Statusvariablen `aurora_binlog_io_cache_reads` und `aurora_binlog_io_cache_read_requests` helfen Ihnen zu überwachen, wie oft die Daten aus dem Binlog-E/A-Cache gelesen werden.

- `aurora_binlog_io_cache_read_requests` zeigt die Anzahl der Binlog-I/O-Leseanfragen aus dem Cache an.
- `aurora_binlog_io_cache_reads` zeigt die Anzahl der Binlog-I/O-Lesevorgänge an, die Informationen aus dem Cache abrufen.

Die folgende SQL-Abfrage berechnet den Prozentsatz der Binlog-Leseanforderungen, die die zwischengespeicherten Informationen nutzen. In diesem Fall ist es umso besser, je näher das Verhältnis an 100 liegt.

```
mysql> SELECT
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
  / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
  * 100
  as binlog_io_cache_hit_ratio;
+---------------------------+
| binlog_io_cache_hit_ratio |
+---------------------------+
|         99.99847949080622 |
+---------------------------+
```

Die Binlog-I/O-Cache-Funktion enthält auch neue Metriken im Zusammenhang mit den Binlog-Dump-Threads. Dump-Threads sind die Threads, die erstellt werden, wenn neue Binlog-Replikate mit der Binlog-Quell-Instance verbunden sind.

Die Metriken des Dump-Threads werden alle 60 Sekunden mit dem Präfix in das Datenbankprotokoll gedruck `[Dump thread metrics]`. Die Metriken umfassen Informationen für jedes Binlog-Replikat wie `Secondary_id`, `Secondary_uuid`, den Namen der Binlog-Datei und die Position, die jedes Replikat liest. Zu den Metriken gehört auch `Bytes_behind_primary`, das den Abstand in Byte zwischen Replikationsquelle und Replikat darstellt. Diese Metrik misst die Verzögerung des Replikat-I/O-Threads. Diese Zahl unterscheidet sich von der Verzögerung des Replikat-SQL-Applier-Threads, der durch die `seconds_behind_master` Metrik auf dem Binlog-Replikat dargestellt wird. Sie können feststellen, ob Binlog-Replikate die Quelle aufholen oder zurückfallen, indem Sie überprüfen, ob die Entfernung abnimmt oder zunimmt.

In-Memory-Relay-Protokoll

In Aurora-MySQL-Version 3.10 und höher führt Aurora eine Optimierung ein, die als In-Memory-Relay-Protokoll bekannt ist, um den Replikationsdurchsatz zu verbessern. Diese Optimierung verbessert die E/A-Leistung des Relay-Protokolls, indem dessen gesamte Zwischeninhalte im Speicher zwischengespeichert werden. Dadurch wird die Verzögerung beim Festschreiben reduziert, indem die E/A-Operationen im Speicher minimiert werden, da der Inhalt des Relay-Protokolls im Speicher weiterhin leicht zugänglich ist.

Standardmäßig ist die In-Memory-Relay-Protokollfunktion automatisch für von Aurora verwaltete Replikationsszenarien (einschließlich Blau/Grün-Bereitstellungen, Aurora-Aurora-Replikationen und regionsübergreifende Replikate) aktiviert, wenn das Replikat eine der folgenden Konfigurationen erfüllt:

- Singlethread-Replikationsmodus (replica_parallel_workers = 0)
- Multithread-Replikation mit aktiviertem GTID-Modus:
  - Automatische Positionierung aktiviert
  - GTID-Modus im Replikat auf ON gesetzt
- Dateibasierte Replikation mit replica_preserve_commit_order = ON

Die In-Memory-Relay-Protokollfunktion wird für Instance-Klassen unterstützt, die größer als t3.large sind, ist aber für Aurora Serverless-Instances nicht verfügbar. Der zirkuläre Puffer des Relay-Protokolls hat eine feste Größe von 128 MB. Zum Überwachen des Speicherverbrauchs dieser Funktion können Sie die folgende Abfrage ausführen:

```
SELECT event_name, current_alloc FROM sys.memory_global_by_current_bytes WHERE
 event_name = 'memory/sql/relaylog_io_cache';
```

Die In-Memory-Relay-Protokollfunktion wird durch den Parameter aurora_in_memory_relaylog gesteuert, der entweder auf DB-Cluster- oder Instance-Ebene festgelegt werden kann. Sie können diese Funktion dynamisch aktivieren oder deaktivieren, ohne Ihre Instance neu starten zu müssen:

1. Stoppen Sie die laufende Replikation.
2. Setzen Sie aurora_in_memory_relaylog in der Parametergruppe auf ON (zum Aktivieren) oder OFF (zum Deaktivieren).
3. Starten Sie die Replikation neu.

Beispiel:

```
CALL mysql.rds_stop_replication;
set aurora_in_memory_relaylog to ON to enable or OFF to disable in cluster parameter
 group
CALL mysql.rds_start_replication;
```

Selbst wenn aurora_in_memory_relaylog auf ON gesetzt ist, kann die In-Memory-Relay-Protokollfunktion unter bestimmten Bedingungen immer noch deaktiviert sein. Verwenden Sie den folgenden Befehl, um den aktuellen Status der Funktion zu überprüfen:

```
SHOW GLOBAL STATUS LIKE 'Aurora_in_memory_relaylog_status';
```

Wenn die Funktion unerwartet deaktiviert wird, können Sie den Grund ermitteln, indem Sie Folgendes ausführen:

```
SHOW GLOBAL STATUS LIKE 'Aurora_in_memory_relaylog_disabled_reason';
```

Dieser Befehl gibt eine Meldung zurück, in der erklärt wird, warum die Funktion derzeit deaktiviert ist.

Einrichten eines erweiterten Binärprotokolls für Aurora MySQL

Das erweiterte Binärprotokoll reduziert den durch die Aktivierung des Binärprotokolls verursachten Rechenleistungs-Overhead, der in bestimmten Fällen bis zu 50 % betragen kann. Mit einem erweiterten Binärprotokoll kann dieser Overhead auf etwa 13 % reduziert werden. Damit der Overhead reduziert wird, schreibt das erweiterte Binärprotokoll die Binär- und Transaktionsprotokolle parallel in den Speicher, wodurch die zum Zeitpunkt des Transaktions-Commits geschriebenen Daten minimiert werden.

Die Verwendung des erweiterten Binärprotokolls verbessert außerdem die Datenbankwiederherstellungszeit nach Neustarts und Failovers um bis zu 99 % im Vergleich zum Community-MySQL-Binärprotokoll. Das erweiterte Binärprotokoll ist mit bestehenden binlogbasierten Workloads kompatibel und Sie interagieren damit genauso wie mit dem Community-MySQL-Binärprotokoll.

Das erweiterte Binärprotokoll ist in Aurora-MySQL-Version 3.03.1 und höher verfügbar.

Themen

- Konfiguration erweiterter Binärprotokollparameter
- Sonstige zugehörige Parameter
- Unterschiede zwischen erweitertem Binärprotokoll und Community-MySQL-Binärprotokoll
- Amazon-CloudWatch-Metriken für das erweiterte Binärprotokoll
- Beschränkungen für das erweiterte Binärprotokoll

Konfiguration erweiterter Binärprotokollparameter

Sie können zwischen Community-MySQL-Binärprotokoll und erweitertem Binärprotokoll wechseln, indem Sie die Parameter für das erweiterte Binärprotokoll aktivieren/deaktivieren. Die bestehenden Binärprotokollbenutzer können die Binärprotokolldateien weiterhin lesen und verarbeiten, ohne dass es zu Lücken in der Binärprotokolldateifolge kommt.

Stellen Sie die folgenden Parameter ein, um das erweiterte Binärprotokoll zu aktivieren:

Parameter	Standard	Beschreibung
`binlog_format`	–	Legen Sie den Parameter `binlog_format` auf das binäre Protokollierungsformat Ihrer Wahl fest, um das erweiterte Binärprotokoll zu aktivieren. Stellen Sie sicher, dass der `binlog_format parameter` nicht auf AUS eingestellt ist. Weitere Informationen finden Sie

Parameter	Standard	Beschreibung
		unter Konfiguration der binären Protokollierung mit Aurora MySQL.
`aurora_enhanced_binlog`	`0`	Legen Sie den Wert dieses Parameters in der Parameter gruppe des DB-Clusters, die mit dem Aurora-MySQL-Cluster verknüpft ist, auf 1 fest. Wenn Sie den Wert dieses Parameters ändern, müssen Sie die Writer-Instance neu starten, wenn für den Wert `DBClusterParameterGroupStatus pending-reboot` angezeigt wird.
`binlog_backup`	`1`	Deaktivieren Sie diesen Parameter, um das erweiterte Binärprotokoll zu aktivieren. Legen Sie dazu den Wert dieses Parameters auf `0` fest.
`binlog_replication_globaldb`	`1`	Deaktivieren Sie diesen Parameter, um das erweiterte Binärprotokoll zu aktivieren. Legen Sie dazu den Wert dieses Parameters auf `0` fest.

Important

Sie können die Parameter `binlog_backup` und `binlog_replication_globaldb` nur deaktivieren, wenn Sie das erweiterte Binärprotokoll verwenden.

Stellen Sie die folgenden Parameter ein, um das erweiterte Binärprotokoll zu deaktivieren:

Parameter	Beschreibung
`aurora_enhanced_binlog`	Legen Sie den Wert dieses Parameters in der Parametergruppe des DB-Clusters, die mit dem Aurora-MySQL-Cluster verknüpft ist, auf `0` fest. Sobald Sie den Wert dieses Parameters ändern, müssen Sie die Writer-Instance neu starten, wenn für den Wert `DBClusterParameterGroupStatus` `pending-reboot` angezeigt wird.
`binlog_backup`	Aktivieren Sie diesen Parameter, wenn Sie das erweiterte Binärprotokoll deaktivieren. Legen Sie dazu den Wert dieses Parameters auf 1 fest.
`binlog_replication_globaldb`	Aktivieren Sie diesen Parameter, wenn Sie das erweiterte Binärprotokoll deaktivieren. Legen Sie dazu den Wert dieses Parameters auf 1 fest.

Um zu überprüfen, ob das erweiterte Binärprotokoll aktiviert ist, verwenden Sie den folgenden Befehl im MySQL-Client:

```
mysql>show status like 'aurora_enhanced_binlog';

+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| aurora_enhanced_binlog | ACTIVE |
+-------------------------+--------+
1 row in set (0.00 sec)
```

Wenn das erweiterte Binärprotokoll aktiviert ist, wird in der Ausgabe `ACTIVE` für `aurora_enhanced_binlog` angezeigt.

Sonstige zugehörige Parameter

Wenn Sie das erweiterte Binärprotokoll aktivieren, sind folgende Parameter betroffen:

- Der Parameter `max_binlog_size` ist sichtbar, kann aber nicht geändert werden. Sein Standardwert `134217728` wird automatisch in `268435456` geändert, wenn das erweiterte Binärprotokoll aktiviert ist.
- Im Gegensatz zum Community-MySQL-Binärprotokoll fungiert `binlog_checksum` nicht als dynamischer Parameter, wenn das erweiterte Binärprotokoll aktiviert ist. Damit die Änderung an diesem Parameter wirksam wird, müssen Sie den DB-Cluster manuell neu starten, auch wenn für `ApplyMethod` `immediate` festgelegt ist.
- Der Wert, den Sie für den Parameter `binlog_order_commits` festlegen, hat keinen Einfluss auf die Reihenfolge der Commits, wenn das erweiterte Binärprotokoll aktiviert ist. Die Commits werden immer ohne weitere Auswirkungen auf die Leistung angeordnet.

Unterschiede zwischen erweitertem Binärprotokoll und Community-MySQL-Binärprotokoll

Das erweiterte Binärprotokoll interagiert anders mit Klonen, Backups und der globalen Aurora-Datenbank als das Community-MySQL-Binärprotokoll. Bevor Sie das erweiterte Binärprotokoll verwenden, sollten Sie sich mit den folgenden Unterschieden vertraut machen.

- Erweiterte Binärprotokolldateien aus dem Quell-DB-Cluster sind in einem geklonten DB-Cluster nicht verfügbar.
- Erweiterte Binärprotokolldateien sind in Aurora-Backups nicht enthalten. Daher sind erweiterte Binärprotokolldateien aus dem Quell-DB-Cluster nach der Wiederherstellung eines DB-Clusters trotz einer für ihn festgelegten Aufbewahrungsfrist nicht verfügbar.
- Bei Verwendung mit einer globalen Aurora-Datenbank werden die erweiterten Binärprotokolldateien des primären DB-Clusters nicht auf den DB-Cluster in den sekundären Regionen repliziert.

Beispiele

Die folgenden Beispiele veranschaulichen die Unterschiede zwischen dem erweiterten Binärprotokoll und dem Community-MySQL-Binärprotokoll.

Auf einem wiederhergestellten oder geklonten DB-Cluster

Wenn das erweiterte Binärprotokoll aktiviert ist, sind die historischen Binärprotokolldateien im wiederhergestellten oder geklonten DB-Cluster nicht verfügbar. Wenn das Binärprotokoll aktiviert

ist, beginnt der neue DB-Cluster nach einem Wiederherstellungs- oder Klonvorgang, seine eigene Reihenfolge von Binärprotokolldateien zu schreiben, beginnend mit 1 (mysql-bin-changelog 000001).

Um das erweiterte Binärprotokoll nach einem Wiederherstellungs- oder Klonvorgang zu aktivieren, legen Sie die erforderlichen DB-Cluster-Parameter auf dem wiederhergestellten oder geklonten DB-Cluster fest. Weitere Informationen finden Sie unter Konfiguration erweiterter Binärprotokollparameter.

Example Beispiel: Ausführung des Klon- oder Wiederherstellungsvorgangs, wenn das erweiterte Binärprotokoll aktiviert ist

Quell-DB-Cluster:

```
mysql> show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog turned on
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

In einem wiederhergestellten oder geklonten DB-Cluster werden Binärprotokolldateien nicht gesichert, wenn das erweiterte Binärprotokoll aktiviert ist. Um Unterbrechungen in den Binärprotokolldaten zu vermeiden, sind die Binärprotokolldateien, die vor dem Aktivieren des erweiterten Binärprotokolls geschrieben wurden, ebenfalls nicht verfügbar.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> New sequence of Binlog files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example Beispiel: Ausführung des Klon- oder Wiederherstellungsvorgangs, wenn das erweiterte Binärprotokoll deaktiviert ist

DB-Quellcluster:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Das erweiterte Binärprotokoll wird nach `mysql-bin-changelog.000003` deaktiviert. In einem wiederhergestellten oder geklonten DB-Cluster sind Binärprotokolldateien verfügbar, die nach dem Deaktivieren des erweiterten Binärprotokoll geschrieben wurden.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

In einer Amazon Aurora Global Database

In einer Amazon Aurora Global Database werden die Binärprotokolldaten des primären DB-Clusters nicht auf den sekundären DB-Cluster repliziert. Nach einem regionsübergreifenden Failover-Prozess sind die Binärprotokolldaten im neu hochgestuften primären DB-Cluster nicht verfügbar. Wenn das Binärprotokoll aktiviert ist, beginnt der neu hochgestufte DB-Cluster mit seiner eigenen Reihenfolge von Binärprotokolldateien, beginnend mit 1 (mysql-bin-changelog 000001).

Um das erweiterte Binärprotokoll nach einem Failover zu aktivieren, müssen Sie die erforderlichen DB-Cluster-Parameter auf dem sekundären DB-Cluster festlegen. Weitere Informationen finden Sie unter Konfiguration erweiterter Binärprotokollparameter.

Example Beispiel: Ausführen eines globalen Datenbank-Failovers, wenn das erweiterte Binärprotokoll aktiviert ist

Alter primärer DB-Cluster (vor dem Failover):

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog enabled
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Neuer primärer DB-Cluster (nach dem Failover):

Binärprotokolldateien werden nicht in sekundäre Regionen repliziert, wenn das erweiterte Binärprotokoll aktiviert ist. Um Unterbrechungen in den Binärprotokolldaten zu vermeiden, sind die Binärprotokolldateien, die vor dem Aktivieren des erweiterten Binärprotokolls geschrieben wurden, nicht verfügbar.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
```

```
| mysql-bin-changelog.000001 |       156 | No        | --> Fresh sequence of Binlog
 files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example Beispiel: Ausführen eines globalen Datenbank-Failovers, wenn das erweiterte Binärprotokoll deaktiviert ist

Quell-DB-Cluster:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Wiederhergestellter oder geklonter DB-Cluster:

Das erweiterte Binärprotokoll wird nach `mysql-bin-changelog.000003` deaktiviert. Binärprotokolldateien, die nach dem Deaktivieren des erweiterten Binärprotokolls geschrieben wurden, werden repliziert und sind im neu hochgestuften DB-Cluster verfügbar.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
```

```
3 rows in set (0.00 sec)
```

Amazon-CloudWatch-Metriken für das erweiterte Binärprotokoll

Die folgenden Amazon-CloudWatch-Metriken werden nur veröffentlicht, wenn das erweiterte Binärprotokoll aktiviert ist.

CloudWatch-Metrik	Beschreibung	Einheiten
ChangeLogBytesUsed	Die Menge des vom Speicherp latzes, der vom erweiterten Binärprotokoll belegt wird.	Bytes
ChangeLogReadIOPs	Die Anzahl von Lese-I/O-Operationen, die im erweitert en Binärprotokoll in einem fünfminütigen Intervalls durchgeführt wurden.	Anzahl pro 5 Minuten
ChangeLogWriteIOPs	Die Anzahl von Schreib-I/O-Operationen, die im erweitert en Binärprotokoll in einem fünfminütigen Intervalls durchgeführt wurden.	Anzahl pro 5 Minuten

Beschränkungen für das erweiterte Binärprotokoll

Die folgenden Einschränkungen gelten für DB-Cluster von Amazon Aurora, wenn das erweiterte Binärprotokoll aktiviert ist.

- Das erweiterte Binärprotokoll wird nur in Aurora-MySQL-Version 3.03.1 und höher unterstützt.
- Die auf dem primären DB-Cluster geschriebenen erweiterten Binärprotokolldateien werden nicht in die geklonten oder wiederhergestellten DB-Cluster kopiert.
- Bei Verwendung mit Amazon Aurora Global Database werden die erweiterten Binärprotokolldateien des primären DB-Clusters nicht auf sekundäre DB-Cluster repliziert. Daher sind die historischen Binärprotokolldaten nach dem Failover-Prozess im neuen primären DB-Cluster nicht verfügbar.
- Die folgenden Binärprotokoll-Konfigurationsparameter werden ignoriert:

  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`
  - `binlog_max_flush_queue_time`
- Sie können eine beschädigte Tabelle in einer Datenbank nicht löschen oder umbenennen. Um diese Tabellen zu löschen, können Sie sich an Support wenden.
- Der Binärprotokoll-I/O-Cache ist deaktiviert, wenn das erweiterte Binärprotokoll aktiviert ist. Weitere Informationen finden Sie unter Optimieren einer binären Protokollreplikation für Aurora MySQL.

> **Note**
>
> Das erweiterte Binärprotokoll bietet ähnliche Verbesserungen der Leseleistung wie der Binärprotokoll-I/O-Cache und deutliche Verbesserungen der Schreibleistung.

- Die Rückverfolgungsfunktion wird nicht unterstützt. Das erweiterte Binärprotokoll kann in einem DB-Cluster unter den folgenden Bedingungen nicht aktiviert werden:
  - Für den DB-Cluster ist derzeit die Rückverfolgungsfunktion aktiviert.
  - Für den DB-Cluster wurde die zuvor aktivierte Rückverfolgungsfunktion inzwischen deaktiviert.
  - Der DB-Cluster wurde aus einem Quell-DB-Cluster oder einem Snapshot mit aktivierter Rückverfolgungsfunktion wiederhergestellt.

Verwenden der GTID-basierten Replikation

Der folgende Inhalt beschreibt, wie Sie globale Transaktionskennungen (Global Transaction Identifiers, GTIDs) mit Binärprotokoll (binlog)-Replikation zwischen einem Aurora-MySQL-Cluster und einer externen Quelle verwenden.

> **Note**
>
> Bei Aurora können Sie diese Funktion nur mit Aurora-MySQL-Clustern verwenden, welche die binlog-Replikation in eine externe oder aus einer externen MySQL-Datenbank nutzen. Bei der anderen Datenbank kann es sich um eine Amazon-RDS-MySQL-Instance, eine lokale MySQL-Datenbank oder einen Aurora-DB-Cluster in einer anderen AWS-Region handeln. Weitere Informationen zum Konfigurieren dieser Art von Replikation finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Wenn Sie die binlog-Replikation verwenden und nicht mit der GTID-basierten Replikation mit MySQL vertraut sind, finden Sie unter Replication with Global Transaction Identifiers in der MySQL-Dokumentation Hintergrundinformationen.

Die GTID-basierte Replikation wird für Aurora-MySQL-Version 2 und 3 unterstützt.

Themen

- Übersicht über globale Transaktionskennungen (GTIDs)
- Parameter für die GTID-basierte Replikation
- Aktivieren der GTID-basierten Replikation für einen Aurora-MySQL-Cluster
- Deaktivieren einer GTID-basierten Replikation für einen Aurora MySQL-DB-Cluster

Übersicht über globale Transaktionskennungen (GTIDs)

Globale Transaktionskennungen (GTIDs) sind eindeutige IDs, die für festgeschriebene MySQL-Transaktionen generiert werden. Sie können GTIDs verwenden, um die Fehlerbehebung für die binlog-Replikation zu erleichtern.

> **Note**
>
> Wenn Aurora Daten unter den DB-Instances in einem Cluster synchronisiert, steht dieser Replikationsmechanismus in keinem Zusammenhang mit dem Binärprotokoll (binlog).
> Bei Aurora MySQL ist die GTID-basierte Replikation nur anwendbar, wenn Sie auch eine binlog-Replikation verwenden, um eine Replikation aus einer externen MySQL-kompatiblen Datenbank in einen oder aus einem Aurora MySQL-DB-Cluster durchführen.

MySQL verwendet für die binlog-Replikation zwei verschiedene Arten von Transaktionen:

- GTID-Transaktionen – Transaktionen, die durch eine GTID gekennzeichnet sind.
- Anonyme Transaktionen – Transaktionen, denen keine GTID zugeordnet ist.

In einer Replikationskonfiguration sind GTIDs bei allen DB-Instances eindeutig. GTIDs vereinfachen die Replikationskonfiguration, weil Sie nicht auf die Protokolldateipositionen verweisen müssen, wenn Sie diese verwenden. GTIDs erleichtern das Verfolgen von replizierten Transaktionen und legen fest, ob die Quell-Instance und Replikate konsistent sind.

Sie verwenden die GTID-basierte Replication gewöhnlich mit Aurora, wenn Sie eine Replikation aus einer externen MySQL-kompatiblen Datenbank in einen Aurora-Cluster durchführen. Sie können diese Replikationskonfiguration als Teil einer Migration von einer lokalen oder einer Amazon-RDS-Datenbank in Aurora MySQL einrichten. Wenn die externe Datenbank bereits GTIDs verwendet, kann der Replikationsvorgang durch Aktivieren der GTID-basierten Replikation für den Aurora-Cluster vereinfacht werden.

Sie konfigurieren die GTID-basierte Replikation für einen Aurora MySQL-Cluster, indem Sie zuerst die relevanten Konfigurationsparameter in einer DB-Clusterparametergruppe festlegen. Sie ordnen diese Parametergruppe dann dem Cluster zu.

Parameter für die GTID-basierte Replikation

Mit den folgenden Parametern konfigurieren Sie die GTID-basierte Replikation.

Parameter	Zulässige Werte	Beschreibung
`gtid_mode`	`OFF`, `OFF_PERMISSIVE` , `ON_PERMISSIVE` , `ON`	`OFF` gibt an, dass neue Transaktionen anonyme Transaktionen sind (d. h. keine GTIDs haben). Eine Transaktion muss anonym sein, um repliziert werden zu können. `OFF_PERMISSIVE` gibt an, dass neue Transaktionen anonyme Transaktionen sind und alle Transaktionen repliziert werden können. `ON_PERMISSIVE` gibt an, dass neue Transaktionen GTID-Transaktionen sind und alle Transaktionen repliziert werden können. `ON` gibt an, dass neue Transaktionen GTID-Transaktionen sind. Eine Transaktion muss eine GTID-Transaktion sein, um repliziert zu werden.
`enforce_gtid_consistency`	`OFF`, `ON`, `WARN`	`OFF` erlaubt es Transaktionen, gegen die GTID-Konsistenz zu verstoßen. `ON` verhindert das Verstoßen von Transaktionen gegen die GTID-Konsistenz.

Parameter	Zulässige Werte	Beschreibung
		`WARN` erlaubt es Transaktionen, gegen die GTID-Konsistenz zu verstoßen, generiert aber eine Warnung, wenn ein Verstoß auftritt.

> **Note**
>
> In der AWS-Managementkonsole erscheint der Parameter `gtid_mode` als `gtid-mode`.

Bei einer GTID-basierten Replikation verwenden Sie diese Einstellungen für die DB-Cluster-Parametergruppe für Ihren Aurora MySQL-DB-Cluster:

- `ON` und `ON_PERMISSIVE` gelten nur für die ausgehende Replikation von einem Aurora-MySQL-Cluster. Beide Werte bewirken, dass Ihr Aurora-DB-Cluster GTIDs für Transaktionen verwenden, die zu einer externen Datenbank repliziert werden. `ON` erfordert, dass die externe Datenbank ebenfalls die GTID-basierte Replikation verwendet. Mit `ON_PERMISSIVE` ist die GTID-basierte Replikation auf der externen Datenbank optional.
- Wenn `OFF_PERMISSIVE` eingestellt ist, bedeutet dies, dass Ihr Aurora-DB-Cluster die eingehende Replikation von einer externen Datenbank akzeptieren kann. Dies ist ungeachtet davon möglich, ob die externe Datenbank eine GTID-basierte Replikation verwendet oder ob nicht.
- Wenn `OFF` eingestellt ist, bedeutet dies, dass Ihr Aurora-DB-Cluster nur eingehende Replikation von externen Datenbanken akzeptiert, die keine GTID-basierte Replikation verwenden.

> **Tip**
>
> Eingehende Replikation ist das geläufigste binlog-Replikationsszenario für Aurora MySQL-Cluster. Für eine eingehende Replikation empfehlen wir, dass Sie den GTID-Modus auf einzustelle `OFF_PERMISSIVE`. Diese Einstellung ermöglicht eine eingehende Replikation aus externen Datenbanken ungeachtete der GTID-Einstellungen an der Replikationsquelle.

Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Aktivieren der GTID-basierten Replikation für einen Aurora-MySQL-Cluster

Wenn die GTID-basierte Replikation für einen Aurora MySQL-DB-Cluster aktiviert ist, gelten die GTID-Einstellungen sowohl für eine eingehende als auch für eine ausgehende binlog-Replikation.

So aktivieren Sie eine GTID-basierte Replikation für einen Aurora MySQL-Cluster

1. Erstellen oder bearbeiten Sie eine DB-Clusterparametergruppe unter Verwendung der folgenden Parametereinstellungen:
   - `gtid_mode` – `ON` oder `ON_PERMISSIVE`
   - `enforce_gtid_consistency` – `ON`
2. Ordnen Sie die DB-Clusterparametergruppe dem Aurora MySQL-Cluster zu. Befolgen Sie hierzu die Verfahren unter Parametergruppen für Amazon Aurora.
3. (Optional) Geben Sie an, wie GTIDs Transaktionen zugewiesen werden, die sie nicht enthalten. Rufen Sie dazu die Gespeicherte Prozedur in mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL Version 3) auf.

Deaktivieren einer GTID-basierten Replikation für einen Aurora MySQL-DB-Cluster

Sie können eine GTID-basierte Replikation für einen Aurora-MySQL-DB-Cluster deaktivieren. Dies bedeutet, dass der Aurora-Cluster keine ein- oder ausgehende binlog-Replikation mit externen Datenbanken ausführen kann, die .

> **Note**
>
> Im folgenden Verfahren bezieht sich Lesereplikat auf das Replikationsziel in einer Aurora-Konfiguration mit binlog-Replikation in eine externe oder aus einer externen Datenbank. Es bezieht sich nicht auf die schreibgeschützten Aurora Replica-DB-Instances. Wenn ein Aurora-Cluster beispielsweise eine eingehende Replikation aus einer externen Quelle akzeptiert, dann fungiert die primäre Aurora-Instance als Lesereplikat der binlog-Replikation.

Weitere Informationen zu den in diesem Abschnitt erwähnten gespeicherten Prozeduren finden Sie unter Referenz für gespeicherte Aurora-MySQL-Prozeduren.

GTID-basierte Replikation für einen Aurora-MySQL-DB-Cluster deaktivieren

1. Führen Sie die folgenden Schritte für die Aurora-Replikate aus:

Für Version 3

```
CALL mysql.rds_set_source_auto_position(0);
```

Für Version 2

```
CALL mysql.rds_set_master_auto_position(0);
```

2. Setzen Sie den Wert für `gtid_mode` auf `ON_PERMISSIVE` zurück.

   a. Stellen Sie sicher, das bei der DB-Clusterparametergruppe, die dem Aurora MySQL-Cluster zugeordnet ist, `gtid_mode` auf `ON_PERMISSIVE` eingestellt ist.

      Weitere Informationen zum Einstellen von Konfigurationsparametern unter Verwendung von Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

   b. Starten Sie den Aurora MySQL-DB-Cluster neu.

3. Setzen Sie den Wert für `gtid_mode` auf `OFF_PERMISSIVE` zurück.

   a. Stellen Sie sicher, das bei der DB-Clusterparametergruppe, die dem Aurora MySQL-Cluster zugeordnet ist, `gtid_mode` auf `OFF_PERMISSIVE` eingestellt ist.

   b. Starten Sie den Aurora MySQL-DB-Cluster neu.

4. Warten Sie, bis alle GTID-Transaktionen auf die primäre Aurora-Instance angewendet wurden. Um zu überprüfen, ob diese angewendet werden, führen Sie die folgenden Schritte aus:

   a. Führen Sie auf der primären Aurora-Instance den Befehl `SHOW MASTER STATUS` aus.

      Die Ausgabe sollte in etwa wie folgt aussehen.

```
File                          Position
-------------------------------------
mysql-bin-changelog.000031      107
-------------------------------------
```

      Notieren Sie die Datei und Position in Ihrer Ausgabe.

   b. Verwenden Sie bei jedem Lesereplikat die Datei- und Positionsinformationen der Quell-Instance im vorherigen Schritt, um die folgende Abfrage auszuführen.

Für Version 3

```
SELECT SOURCE_POS_WAIT('file', position);
```

Für Version 2

```
SELECT MASTER_POS_WAIT('file', position);
```

Führen Sie die folgende Anweisung aus, wenn der Dateiname `mysql-bin-changelog.000031` und die Position `107` lautet:

Für Version 3

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

Für Version 2

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. Setzen Sie die GTID-Parameter zurück, um die GTID-basierte Replikation zu deaktivieren:

   a. Die DB-Clusterparametergruppe, die dem Aurora MySQL-Cluster zugeordnet ist, muss über die folgenden Parametereinstellungen verfügen:

      - `gtid_mode` – `OFF`
      - `enforce_gtid_consistency` – `OFF`

   b. Starten Sie den Aurora MySQL-DB-Cluster neu.

Verwendung der lokalen Schreibweiterleitung in einem DB-Cluster von Amazon Aurora MySQL

Lokale (clusterinterne) Schreibweiterleitung ermöglicht Ihren Anwendungen, Lese-/Schreibtransaktionen direkt auf einem Aurora-Replikat durchzuführen. Diese Transaktionen werden dann zum Commit an die Writer-DB-Instance weitergeleitet. Sie können die lokale Schreibweiterleitung verwenden, wenn Ihre Anwendungen Lesen-nach-Schreiben-Konsistenz erfordern. Dies ist die Fähigkeit, den letzten Schreibvorgang in einer Transaktion zu lesen.

Lesereplikate erhalten asynchron Updates vom Writer. Ohne Schreibweiterleitung müssen Sie alle Lesevorgänge, die eine Lesen-nach-Schreiben-Konsistenz erfordern, auf der Writer-DB-Instance abwickeln. Oder Sie müssen eine komplexe benutzerdefinierte Anwendungslogik entwickeln, um aus Gründen der Skalierbarkeit mehrere Lesereplikate nutzen zu können. Ihre Anwendungen müssen den gesamten Lese- und Schreibverkehr vollständig aufteilen und zwei Gruppen von Datenbankverbindungen aufrechterhalten, um den Datenverkehr an den richtigen Endpunkt zu senden. Dieser Entwicklungsaufwand erschwert das Anwendungsdesign, wenn die Abfragen Teil einer einzelnen logischen Sitzung bzw. Transaktion innerhalb der Anwendung sind. Da die Replikationsverzögerung zwischen den Lesereplikaten unterschiedlich sein kann, ist es außerdem schwierig, eine globale Lesekonsistenz für alle Instances in der Datenbank zu erreichen.

Durch die Schreibweiterleitung müssen diese Transaktionen nicht aufgeteilt oder ausschließlich an den Writer gesendet werden, was die Anwendungsentwicklung vereinfacht. Diese neue Funktion macht es einfach, Leseskalierung für Workloads zu erreichen, die den letzten Schreibvorgang in einer Transaktion lesen müssen und nicht empfindlich auf Schreiblatenz reagieren.

Die lokale Schreibweiterleitung unterscheidet sich von der globalen Schreibweiterleitung, bei der Schreibvorgänge von einem sekundären DB-Cluster an den primären DB-Cluster in einer globalen Aurora-Datenbank weitergeleitet werden. Sie können die lokale Schreibweiterleitung in einem DB-Cluster verwenden, der Teil einer globalen Aurora-Datenbank ist. Weitere Informationen finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

Die Schreibweiterleitung erfordert Aurora-MySQL-Version 3.04 oder höher.

Themen

- Aktivieren der lokalen Schreibweiterleitung
- Überprüfen, ob die Schreibweiterleitung für einen DB-Cluster aktiviert ist

- Anwendung und SQL-Kompatibilität mit Schreibweiterleitung
- Isolationsstufen für die Schreibweiterleitung
- Lesekonsistenz für die Schreibweiterleitung
- Ausführen von Multipart-Anweisungen mit Schreibweiterleitung
- Transaktionen mit Schreibweiterleitung
- Konfigurationsparameter für die Schreibweiterleitung
- Amazon-CloudWatch-Metriken und Aurora-MySQL-Statusvariablen für die Schreibweiterleitung
- Identifizieren weitergeleiteter Transaktionen und Abfragen

Aktivieren der lokalen Schreibweiterleitung

Standardmäßig ist die lokale Schreibweiterleitung für DB-Cluster von Aurora MySQL nicht aktiviert. Sie aktivieren die lokale Schreibweiterleitung auf Cluster-Ebene, nicht auf Instance-Ebene.

> **Important**
>
> Sie können auch die lokale Schreibweiterleitung für regionsübergreifende Lesereplikate aktivieren, die Binärprotokollierung verwenden. Schreibvorgänge werden jedoch nicht an die Quelle AWS-Region weitergeleitet. Sie werden an die Writer-DB-Instance des Binärprotokoll-Lesereplikats-Clusters weitergeleitet.
> Verwenden Sie diese Methode nur, wenn Sie einen Anwendungsfall für das Schreiben in das Binärprotokoll-Lesereplikat in der sekundären AWS-Region haben. Andernfalls könnte es zu einem „Split-Brain“ -Szenario kommen, in dem replizierte Datensätze nicht miteinander konsistent sind.
> Es wird empfohlen, die globale Schreibweiterleitung für globale Datenbanken anstelle der lokalen Schreibweiterleitung für regionsübergreifende Lesereplikate zu verwenden, sofern dies nicht unbedingt erforderlich ist. Weitere Informationen finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

Konsole

Wählen Sie in der AWS-Managementkonsole das Kontrollkästchen Lokale Schreibweiterleitung aktivieren unter Read-Replica-Schreibweiterleitung aktivieren aus, wenn Sie einen DB-Cluster erstellen oder ändern.

AWS CLI

Wenn Sie die Schreibweiterleitung über die AWS CLI aktivieren möchten, verwenden Sie die Option `--enable-local-write-forwarding`. Diese Option funktioniert, wenn Sie über den Befehl `create-db-cluster` einen neuen DB-Cluster erstellen. Sie funktioniert auch, wenn Sie einen vorhandenen DB-Cluster über den Befehl `modify-db-cluster` ändern. Sie können die Schreibweiterleitung deaktivieren, indem Sie die Option `--no-enable-local-write-forwarding` mit denselben CLI-Befehlen verwenden.

Im folgenden Beispiel wird ein DB-Cluster von Aurora MySQL mit aktivierter Schreibweiterleitung erstellt.

```
aws rds create-db-cluster \
  --db-cluster-identifier write-forwarding-test-cluster \
  --enable-local-write-forwarding \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.04.0 \
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

Anschließend erstellen Sie Writer- und Reader-DB-Instances, sodass Sie die Schreibweiterleitung verwenden können. Weitere Informationen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

RDS-API

Um die Schreibweiterleitung über die Amazon-RDS-API zu aktivieren, legen Sie den Parameter `EnableLocalWriteForwarding` auf `true` fest. Dieser Parameter funktioniert, wenn Sie über die Operation `CreateDBCluster` einen neuen DB-Cluster erstellen. Sie funktioniert auch, wenn Sie einen vorhandenen DB-Cluster über die Operation `ModifyDBCluster` ändern. Sie können die Schreibweiterleitung deaktivieren, indem Sie den Parameter `EnableLocalWriteForwarding` auf `false` festlegen.

Aktivieren der Schreibweiterleitung für Datenbanksitzungen

Der Parameter `aurora_replica_read_consistency` ist ein DB-Parameter und ein DB-Cluster-Parameter, der die Schreibweiterleitung ermöglicht. Sie können `EVENTUAL`, `SESSION` oder `GLOBAL` als Lesekonsistenzstufe angeben. Weitere Informationen über Konsistenzebenen finden Sie unter Lesekonsistenz für die Schreibweiterleitung.

Für diesen Parameter gelten die folgenden Regeln:

- Der Standardwert ist (null).
- Die Schreibweiterleitung ist nur verfügbar, wenn `aurora_replica_read_consistency` auf `EVENTUAL`, `SESSION` oder `GLOBAL` festgelegt ist. Dieser Parameter ist nur in Reader-Instances von DB-Clustern relevant, für die Schreibweiterleitung aktiviert ist.
- Sie können diesen Parameter (wenn leer) nicht innerhalb einer Multistatement-Transaktion festlegen oder die Einstellung (wenn er bereits eingestellt ist) nicht aufheben. Sie können ihn während einer solchen Transaktion von einem gültigen Wert in einen anderen gültigen Wert ändern. Diese Aktion wird jedoch nicht empfohlen.

Überprüfen, ob die Schreibweiterleitung für einen DB-Cluster aktiviert ist

Wenn Sie feststellen möchten, ob Sie die Schreibweiterleitung in einem DB-Cluster verwenden können, stellen Sie sicher, dass das Attribut `LocalWriteForwardingStatus` für den Cluster auf `enabled` eingestellt ist.

In der AWS-Managementkonsole sehen Sie auf der Registerkarte Konfiguration der Detailseite des Clusters den StatusAktiviert für Lokale Lesereplikat-Schreibweiterleitung.

Um den Status der Einstellung für die Schreibweiterleitung für alle Cluster anzuzeigen, führen Sie den folgenden AWS CLI-Befehl aus.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
    {
        "LocalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-1"
    },
    {
        "LocalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "requested",
```

```
        "DBClusterIdentifier": "test-global-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "null",
        "DBClusterIdentifier": "aurora-mysql-v2-cluster"
    }
]
```

Für einen DB-Cluster sind folgende Werte `LocalWriteForwardingStatus` zulässig:

- `disabled` – Die Schreibweiterleitung ist deaktiviert.
- `disabling` – Die Schreibweiterleitung wird gerade deaktiviert.
- `enabled` – Die Schreibweiterleitung ist aktiviert.
- `enabling` – Die Schreibweiterleitung wird gerade aktiviert.
- `null` – Die Schreibweiterleitung ist für diesen DB-Cluster nicht verfügbar.
- `requested` – Die Schreibweiterleitung wurde angefordert, ist aber noch nicht aktiv.

Anwendung und SQL-Kompatibilität mit Schreibweiterleitung

Sie können die folgenden Arten von SQL-Anweisungen mit Schreibweiterleitung verwenden:

- Data Manipulation Language (DM)-Anweisungen wie `INSERT`, wie`DELETE` und `UPDATE`. Es gibt einige Einschränkungen für die Eigenschaften dieser Anweisungen, die Sie bei der Schreibweiterleitung verwenden können, wie im Folgenden beschrieben.
- `SELECT ... LOCK IN SHARE MODE`- und `SELECT FOR UPDATE`-Anweisungen.
- `PREPARE`- und `EXECUTE`-Anweisungen.

Bestimmte Anweisungen sind nicht zulässig oder können veraltete Ergebnisse erzeugen, wenn Sie sie in einem DB-Cluster mit Schreibweiterleitung verwenden. Darüber hinaus werden benutzerdefinierte Funktionen und benutzerdefinierte Prozeduren nicht unterstützt. Somit ist die die Einstellung `EnableLocalWriteForwarding` für DB-Cluster standardmäßig deaktiviert. Stellen Sie vor der Aktivierung sicher, dass der Anwendungscode von keiner dieser Einschränkungen betroffen ist.

Die folgenden Einschränkungen gelten für die SQL-Anweisungen, die Sie für die Schreibweiterleitung verwenden. In einigen Fällen können Sie die Anweisungen in DB-Clustern verwenden, bei denen eine Schreibweiterleitung auf Cluster-Ebene aktiviert ist. Dieser Ansatz funktioniert,

wenn die Schreibweiterleitung nicht innerhalb der Sitzung durch den Konfigurationsparameter `aurora_replica_read_consistency` aktiviert wird. Wenn Sie versuchen, eine Anweisung zu verwenden, obwohl diese aufgrund der Schreibweiterleitung nicht zulässig ist, wird eine Fehlermeldung ähnlich wie die folgende ausgegeben:

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

Data Definition Language (DDL)

Stellen Sie eine Verbindung mit der Writer-DB-Instance her, um DDL-Anweisungen auszuführen. Sie können sie nicht von Reader-DB-Instances aus ausführen.

Aktualisieren einer permanenten Tabelle mit Daten aus einer temporären Tabelle

Sie können in DB-Clustern mit aktivierter Schreibweiterleitung temporäre Tabellen verwenden. Sie können jedoch keine DML-Anweisung verwenden, um eine permanente Tabelle zu ändern, wenn sich die Anweisung auf eine temporäre Tabelle bezieht. Beispielsweise können Sie keine `INSERT ... SELECT`-Anweisung verwenden, die die Daten aus einer temporären Tabelle übernimmt.

XA-Transaktionen

Sie können die folgenden Anweisungen nicht in einem DB-Cluster verwenden, wenn die Schreibweiterleitung innerhalb der Sitzung aktiviert wird. Sie können diese Anweisungen in DB-Clustern verwenden, bei denen die Schreibweiterleitung nicht aktiviert ist, oder innerhalb von Sitzungen mit leerer Einstellung `aurora_replica_read_consistency`. Bevor Sie die Schreibweiterleitung innerhalb einer Sitzung aktivieren, müssen Sie überprüfen, ob Ihr Code diese Anweisungen verwendet.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

LOAD-Anweisungen für permanente Tabellen

Sie können die folgenden Anweisungen nicht in einem DB-Cluster mit aktivierter Schreibweiterleitung verwenden.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

Plugin-Anweisungen

Sie können die folgenden Anweisungen nicht in einem DB-Cluster mit aktivierter Schreibweiterleitung verwenden.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT-Anweisungen

Sie können die folgenden Anweisungen nicht in einem DB-Cluster verwenden, wenn die Schreibweiterleitung innerhalb der Sitzung aktiviert wird. Sie können diese Anweisungen in DB-Clustern ohne aktivierte Schreibweiterleitung oder in Sitzungen mit leerer Einstellung `aurora_replica_read_consistency` verwenden. Überprüfen Sie, ob Ihr Code diese Anweisungen verwendet, bevor Sie die Schreibweiterleitung innerhalb einer Sitzung aktivieren.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Isolationsstufen für die Schreibweiterleitung

In Sitzungen, die Schreibweiterleitung verwenden, können Sie nur die Isolationsstufe `REPEATABLE READ` verwenden. Obwohl Sie die Isolationsstufe `READ COMMITTED` auch mit Aurora Replicas verwenden können, funktioniert diese Isolationsstufe nicht mit Schreibweiterleitung. Weitere Informationen zu den Isolationsstufen `REPEATABLE READ` und `READ COMMITTED` finden Sie unter Aurora MySQL-Isolierungsstufen.

Lesekonsistenz für die Schreibweiterleitung

Sie können den Grad der Lesekonsistenz in einem DB-Cluster steuern. Die Lesekonsistenzstufe legt fest, wie lange der DB-Cluster vor jeder Leseoperation wartet, um sicherzustellen, dass einige oder alle Änderungen aus dem Writer repliziert werden. Sie können die Lesekonsistenzstufe anpassen, um sicherzustellen, dass alle aus Ihrer Sitzung weitergeleiteten Schreiboperationen im DB-Cluster vor nachfolgenden Abfragen angezeigt werden. Sie können diese Einstellung auch verwenden,

um sicherzustellen, dass Abfragen auf dem DB-Cluster stets die aktuellsten Updates des Writers angezeigt werden. Diese Einstellung gilt auch für Abfragen, die von anderen Sitzungen oder Clustern gesendet werden. Um diese Art von Verhalten für Ihre Anwendung anzugeben, wählen Sie einen Wert für den DB-Parameter `aurora_replica_read_consistency` oder den DB-Cluster-Parameter aus.

> **Important**
>
> Stellen Sie den DB-Parameter `aurora_replica_read_consistency` oder den DB-Cluster-Parameter immer ein, wenn Sie Schreibvorgänge weiterleiten möchten. Andernfalls leitet Aurora keine Schreibvorgänge weiter. Dieser Parameter hat standardmäßig einen leeren Wert, wählen Sie daher einen bestimmten Wert, wenn Sie diesen Parameter verwenden. Der Parameter `aurora_replica_read_consistency` wirkt sich nur auf DB-Cluster oder Instances aus, für die die Schreibweiterleitung aktiviert ist.

Wenn Sie die Konsistenzstufe erhöhen, verbringt Ihre Anwendung mehr Zeit mit dem Warten auf die Propagierung von Änderungen zwischen DB-Instances. Sie können das Verhältnis zwischen schneller Reaktionszeit und der Gewährleistung festlegen, dass vor der Ausführung Ihrer Abfragen Änderungen auf anderen DB-Instances vollständig verfügbar sind.

Sie können die folgenden Werte für den Parameter `aurora_replica_read_consistency` angeben:

- `EVENTUAL` – Ergebnisse von Schreibvorgängen in derselben Sitzung sind erst sichtbar, wenn der Schreibvorgang auf der Writer-DB-Instance ausgeführt wird. Die Abfrage wartet nicht, bis die aktualisierten Ergebnisse verfügbar sind. Daher könnte sie die älteren Daten oder die aktualisierten Daten abrufen, abhängig vom Zeitpunkt der Anweisungen und der Größe der Replikationsverzögerung. Dies entspricht der Konsistenz von DB-Clustern von Aurora MySQL, die keine Schreibweiterleitung verwenden.
- `SESSION` – Allen Abfragen, die die Schreibweiterleitung verwenden, werden die Ergebnisse sämtlicher in dieser Sitzung vorgenommenen Änderungen angezeigt. Die Änderungen werden unabhängig davon angezeigt, ob die Transaktion übergeben wurde. Wenn notwendig, wartet die Abfrage auf die Replikation der Ergebnisse weitergeleiteter Schreibvorgänge.
- `GLOBAL` – In einer Sitzung werden alle übernommenen Änderungen in allen Sitzungen und Instances im DB-Cluster angezeigt. Jede Abfrage kann für einen bestimmten Zeitraum warten, abhängig von der Sitzungsverzögerung. Die Abfrage wird fortgesetzt, wenn der DB-Cluster

mit allen übergebenen Daten aus dem Writer mit Stand zu dem Zeitpunkt, an dem die Abfrage gestartet wurde, aktualisiert ist.

Informationen zu den mit der Schreibweiterleitung verbundenen Parametern finden Sie unter Konfigurationsparameter für die Schreibweiterleitung.

> **Note**
>
> Sie können auch `aurora_replica_read_consistency` als Sitzungsvariable verwenden, zum Beispiel:
>
> ```
> mysql> set aurora_replica_read_consistency = 'session';
> ```

Beispiele für die Verwendung der Schreibweiterleitung

Die folgenden Beispiele zeigen die Auswirkungen des Parameters `aurora_replica_read_consistency` auf laufende `INSERT`-Anweisungen, gefolgt von `SELECT`-Anweisungen. Abhängig vom Wert der Einstellung `aurora_replica_read_consistency` und vom Zeitpunkt der Anweisungen können sich die Ergebnisse unterscheiden.

Um eine höhere Konsistenz zu erreichen, können Sie kurz warten, bevor Sie die Anweisung `SELECT` ausführen. Oder Aurora kann automatisch warten, bis die Ergebnisse fertig repliziert sind, bevor mit fortgefahren wir `SELECT`.

Informationen zum Einstellen von DB-Parametern finden Sie unter Parametergruppen für Amazon Aurora.

Example mit der Einstellung von **`aurora_replica_read_consistency`** auf **`EVENTUAL`**

Durch Ausführen einer `INSERT`-Anweisung, auf die unmittelbar eine `SELECT`-Anweisung folgt, wird ein Wert für `COUNT(*)` mit der Anzahl der Zeilen zurückgegeben, bevor die neue Zeile eingefügt wird. Wenn Sie kurze Zeit später `SELECT` erneut ausführen, wird die aktualisierte Zeilenzahl zurückgegeben. Die `SELECT`-Anweisungen haben keine Wartezeit.

```
mysql> select count(*) from t1;
+----------+
```

```
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Example mit der Einstellung von **`aurora_replica_read_consistency`** auf **`SESSION`**

Eine `SELECT`-Anweisung wartet unmittelbar nach einer `INSERT`-Anweisung, bis die Änderungen aus der `INSERT`-Anweisung sichtbar sind. Nachfolgende `SELECT`-Anweisungen haben keine Wartezeit.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)

mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
```

```
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

Wenn die Lesekonsistenzeinstellung weiter auf `SESSION` festgelegt ist, wird durch die Einführung einer kurzen Wartezeit nach der Ausführung einer `INSERT`-Anweisung die aktualisierte Zeilenzahl zum Zeitpunkt der nächsten Ausführung der `SELECT`-Anweisung verfügbar.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Example mit der Einstellung von **`aurora_replica_read_consistency`** auf **`GLOBAL`**

Jede `SELECT`-Anweisung wartet, um sicherzustellen, dass alle Datenänderungen mit Stand zur Startzeit der Anweisung angezeigt werden, bevor die Abfrage ausgeführt wird. Die Wartezeit für jede `SELECT`-Anweisung variiert je nach Dauer der Replikationsverzögerung.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)

mysql> select count(*) from t1;
+----------+
```

```
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

Ausführen von Multipart-Anweisungen mit Schreibweiterleitung

Eine DML-Anweisung kann aus mehreren Teilen bestehen, z. B. einer `INSERT ... SELECT`-Anweisung oder einer `DELETE ... WHERE`-Anweisung. In diesem Fall wird die gesamte Anweisung an die Writer-DB-Instance weitergeleitet und dort ausgeführt.

Transaktionen mit Schreibweiterleitung

Wenn der Transaktionszugriffsmodus schreibgeschützt ist, wird die Schreibweiterleitung nicht verwendet. Sie können den Zugriffsmodus für die Transaktion durch Verwendung der Anweisungen `SET TRANSACTION` oder `START TRANSACTION` angeben. Sie können den Transaktionszugriffsmodus auch angeben, indem Sie den Wert der Sitzungsvariablen transaction_read_only ändern. Sie können diesen Sitzungswert nur ändern, wenn Sie mit einem DB-Cluster verbunden sind, für den die Schreibweiterleitung aktiviert ist.

Wenn eine Transaktion mit langer Laufzeit für einen beträchtlichen Zeitraum keine Anweisung ausgibt, kann sie Leerlaufzeitüberschreitung überschreiten. Dieser Zeitraum hat einen Standardwert von einer Minute. Sie können den Parameter `aurora_fwd_writer_idle_timeout` festlegen, um den Wert um bis zu einem Tag zu erhöhen. Eine Transaktion, die die Leerlaufzeitüberschreitung überschreitet, wird von der Writer-Instance abgebrochen. Die nächste nachfolgende Anweisung, die Sie übermitteln, empfängt einen Zeitüberschreitungsfehler. In diesem Fall rollt Aurora die Transaktion zurück.

Diese Art von Fehler kann in anderen Fällen auftreten, wenn die Schreibweiterleitung nicht mehr verfügbar ist. Beispielsweise bricht Aurora alle Transaktionen ab, die die Schreibweiterleitung verwenden, wenn Sie den DB-Cluster neu starten oder die Schreibweiterleitung deaktivieren.

Wenn eine Writer-Instance in einem Cluster, der die lokale Schreibweiterleitung verwendet, neu gestartet wird, werden alle aktiven, weitergeleiteten Transaktionen und Abfragen auf Reader-Instances, die die lokale Schreibweiterleitung verwenden, automatisch geschlossen. Nachdem die Writer-Instance wieder verfügbar ist, können Sie diese Transaktionen erneut versuchen.

Konfigurationsparameter für die Schreibweiterleitung

Die Aurora-DB-Parametergruppen enthalten Einstellungen für die Schreibweiterleitung. Details zu diesen Parametern sind in der folgenden Tabelle zusammengefasst, mit Nutzungshinweisen unterhalb der Tabelle.

Parameter	Scope	Typ	Standardwert	Zulässige Werte
`aurora_fwd_writer_idle_timeout`	Cluster	Ganzzahl ohne Vorzeichen	60	1 – 86.400
`aurora_fwd_writer_max_connections_pct`	Cluster	Lange Ganzzahl ohne Vorzeichen	10	0 – 90
`aurora_replica_read_consistency`	Cluster oder Instances	Enum	" (null)	`EVENTUAL`, `SESSION`, `GLOBAL`

Verwenden Sie die folgenden Einstellungen, um eingehende Schreibanforderungen zu steuern:

- `aurora_fwd_writer_idle_timeout` – Die Anzahl der Sekunden, die die Writer-DB-Instance auf Aktivitäten bei einer Verbindung wartet, die von einer Reader-Instance weitergeleitet werden, bevor sie geschlossen wird. Wenn die Sitzung über diesen Zeitraum hinaus im Leerlauf ist, bricht Aurora die Sitzung ab.
- `aurora_fwd_writer_max_connections_pct` – Die obere Grenze für Datenbankverbindungen, die in einer Writer-DB-Instance für die Verarbeitung von Abfragen verwendet werden können, die von Reader-Instances weitergeleitet werden. Sie wird als Prozentsatz der Einstellung `max_connections` für den Writer ausgedrückt. Wenn

beispielsweise `max_connections` 800 ist und `aurora_fwd_master_max_connections_pct` oder `aurora_fwd_writer_max_connections_pct` 10 ist, lässt der Writer maximal 80 weitergeleitete Sitzungen gleichzeitig zu. Diese Verbindungen stammen aus demselben, von der Einstellung `max_connections` verwalteten Verbindungspool.

Diese Einstellung gilt nur für den Writer, wenn dessen Schreibweiterleitung aktiviert ist. Wenn Sie den Wert verringern, sind vorhandene Verbindungen nicht betroffen. Aurora berücksichtigt den neuen Wert der Einstellung, wenn versucht wird, eine neue Verbindung von einem DB-Cluster aus zu erstellen. Der Standardwert ist 10, was 10 % des Werts für `max_connections` entspricht.

> **Note**
>
> Da es sich bei `aurora_fwd_writer_idle_timeout` und `aurora_fwd_writer_max_connections_pct` um Cluster-Parameter handelt, haben alle DB-Instances in allen Clustern die gleichen Werte für diese Parameter.

Mehr über `aurora_replica_read_consistency` erfahren Sie unter Lesekonsistenz für die Schreibweiterleitung.

Weitere Informationen zu DB-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Amazon-CloudWatch-Metriken und Aurora-MySQL-Statusvariablen für die Schreibweiterleitung

Die folgenden Amazon-CloudWatch-Metriken und Aurora-MySQL-Statusvariablen gelten, wenn Sie die Schreibweiterleitung für Aurora MySQL verwenden. Weitere Informationen zu Metriken für Writer- und Reader-DB-Instances von Aurora MySQL finden Sie in den folgenden Themen.

Themen

- Metriken für die Schreibweiterleitung für Writer-DB-Instances von Aurora MySQL
- Metriken für die Schreibweiterleitung für Reader-DB-Instances von Aurora MySQL

Metriken für die Schreibweiterleitung für Writer-DB-Instances von Aurora MySQL

Die folgenden Amazon-CloudWatch-Metriken gelten, wenn Sie die Schreibweiterleitung in einem oder mehreren DB-Clustern verwenden. Diese Metriken werden alle auf der Writer-DB-Instance gemessen.

CloudWatch-Metrik	Einheit	Beschreibung
`ForwardingWriterDMLLatency`	Millisekunden	Durchschnittliche Zeit für die Verarbeitung jeder weitergeleiteten DML-Anweisung auf der Writer-DB-Instance. Nicht enthalten ist die Zeit, die der DB-Cluster benötigt, um die Schreibanforderung weiterzuleiten, oder die Zeit, um Änderungen zurück an den Writer zu replizieren.
`ForwardingWriterDMLThroughput`	Anzahl pro Sekunde	Anzahl der weitergeleiteten DML-Anweisungen, die pro Sekunde von dieser Writer-DB-Instance verarbeitet werden.
`ForwardingWriterOpenSessions`	Anzahl	Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance.

Die folgenden Aurora-MySQL-Statusvariablen gelten, wenn Sie die Schreibweiterleitung in einem oder mehreren DB-Clustern verwenden. Diese Statusvariablen werden alle auf der Writer-DB-Instance gemessen.

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
`Aurora_fwd_writer_dml_stmt_count`	Anzahl	Gesamtzahl der DML-Anweisungen, die an diese Writer-

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
		DB-Instance weitergeleitet werden.
`Aurora_fwd_writer_dml_stmt_duration`	Mikrosekunden	Gesamtdauer der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden.
`Aurora_fwd_writer_open_sessions`	Anzahl	Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance.
`Aurora_fwd_writer_select_stmt_count`	Anzahl	Gesamtzahl der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden.
`Aurora_fwd_writer_select_stmt_duration`	Mikrosekunden	Gesamtdauer der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden.

Metriken für die Schreibweiterleitung für Reader-DB-Instances von Aurora MySQL

Die folgenden CloudWatch-Metriken werden auf jeder Reader-DB-Instance in einem DB-Cluster mit aktivierter Schreibweiterleitung gemessen.

CloudWatch-Metrik	Einheit	Beschreibung
`ForwardingReplicaDMLLatency`	Millisekunden	Durchschnittliche Reaktionszeit weitergeleiteter DML-Anweisungen auf Replikaten.
`ForwardingReplicaDMLThroughput`	Anzahl pro Sekunde	Anzahl der weitergeleiteten DML-Anweisungen, die pro Sekunde verarbeitet werden.

CloudWatch-Metrik	Einheit	Beschreibung
`ForwardingReplicaOpenSessions`	Anzahl	Die Anzahl der Sitzungen, die die Schreibweiterleitung für eine Reader-DB-Instance verwenden.
`ForwardingReplicaReadWaitLatency`	Millisekunden	Durchschnittliche Wartezeit, die eine `SELECT`-Anweisung in einer Reader-DB-Instance darauf wartet, mit dem Writer aufzuschließen. Der Grad, in dem die Reader-DB-Instance vor der Verarbeitung einer Abfrage wartet, ist von der Einstellung `aurora_replica_read_consistency` abhängig.
`ForwardingReplicaReadWaitThroughput`	Anzahl pro Sekunde	Gesamtzahl der pro Sekunde in allen Sitzungen verarbeiteten `SELECT`-Anweisungen, die Schreibvorgänge weiterleiten.
`ForwardingReplicaSelectLatency`	Millisekunden	Weitergeleitete `SELECT`-Latenz, als Durchschnitt über alle weitergeleiteten `SELECT`-Anweisungen innerhalb des Überwachungszeitraums.
`ForwardingReplicaSelectThroughput`	Anzahl pro Sekunde	Weitergeleiteter `SELECT`-Durchsatz, pro Sekunde, als Durchschnitt innerhalb des Überwachungszeitraums.

Die folgenden Aurora MySQL-Statusvariablen werden auf jeder Reader-DB-Instance in einem DB-Cluster mit aktivierter Schreibweiterleitung gemessen.

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
`Aurora_fwd_replica_dml_stmt_count`	Anzahl	Gesamtzahl der DML-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.
`Aurora_fwd_replica_dml_stmt_duration`	Mikrosekunden	Gesamtdauer aller DML-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.
`Aurora_fwd_replica_errors_session_limit`	Anzahl	Anzahl der Sitzungen, die vom primären Cluster aufgrund einer der folgenden Fehlerbedingungen zurückgewiesen wurden: • Writer voll • Es sind zu viele weitergeleitete Anweisungen in Bearbeitung.
`Aurora_fwd_replica_open_sessions`	Anzahl	Die Anzahl der Sitzungen, die die Schreibweiterleitung für eine Reader-DB-Instance verwenden.
`Aurora_fwd_replica_read_wait_count`	Anzahl	Gesamtzahl der Lesen-Nach-Schreiben-Wartezeiten auf dieser Reader-DB-Instance.
`Aurora_fwd_replica_read_wait_duration`	Mikrosekunden	Gesamtdauer der Wartezeiten aufgrund der Lesekonsi

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
		stenzeinstellung auf dieser Reader-DB-Instance.
`Aurora_fwd_replica_select_stmt_count`	Anzahl	Gesamtzahl der `SELECT`-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.
`Aurora_fwd_replica_select_stmt_duration`	Mikrosekunden	Gesamtdauer der `SELECT`-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.

Identifizieren weitergeleiteter Transaktionen und Abfragen

Sie können die Tabelle `information_schema.aurora_forwarding_processlist` zur Identifizierung weitergeleiteter Transaktionen und Abfragen verwenden. Weitere Informationen zu dieser Tabelle finden Sie unter information_schema.aurora_forwarding_processlist.

Das folgende Beispiel zeigt alle weitergeleiteten Verbindungen auf einer Writer-DB-Instance.

```
mysql> select * from information_schema.AURORA_FORWARDING_PROCESSLIST where
 IS_FORWARDED=1 order by REPLICA_SESSION_ID;

+-----+----------+-------------------+----------+---------+------+-------------
+--------------------------------------------+--------------+-------------------
+---------------------------------+----------------------+---------------+
| ID  | USER     | HOST              | DB       | COMMAND | TIME | STATE        |
 INFO                                      | IS_FORWARDED | REPLICA_SESSION_ID |
 REPLICA_INSTANCE_IDENTIFIER     | REPLICA_CLUSTER_NAME | REPLICA_REGION |
+-----+----------+-------------------+----------+---------+------+-------------
+--------------------------------------------+--------------+-------------------
+---------------------------------+----------------------+---------------+
| 648 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest58 SET k=k+1 WHERE id=4802579 |            1 |                637 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
```

```
| 650 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest54 SET k=k+1 WHERE id=2503953 |            1 |               639 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+--------------------
+---------------------------------+---------------------+----------------+
```

Auf der weiterleitenden Reader-DB-Instance können Sie die Threads sehen, die diesen Writer-DB-Verbindungen zugeordnet sind, indem Sie `SHOW PROCESSLIST` ausführen. Die `REPLICA_SESSION_ID`-Werte auf dem Writer, 637 und 639, sind dieselben wie die `Id`-Werte auf dem Reader.

```
mysql> select @@aurora_server_id;

+---------------------------------+
| @@aurora_server_id              |
+---------------------------------+
| my-db-cluster-instance-2        |
+---------------------------------+
1 row in set (0.00 sec)

mysql> show processlist;

+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| Id  | User     | Host               | db       | Command | Time | State        | Info
                                       |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| 637 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest12 SET k=k+1 WHERE id=4802579  |
| 639 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest61 SET k=k+1 WHERE id=2503953  |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
12 rows in set (0.00 sec)
```

Integrieren von Amazon Aurora MySQL in anderen AWS-Services

Amazon Aurora MySQL ist auch in anderen AWS-Services integriert, damit Sie Ihren Aurora-MySQL-DB-Cluster erweitern können, um zusätzliche Funktionen in der AWS Cloud zu verwenden. Ihr Aurora-MySQL-DB-Cluster kann AWS-Services verwenden, um Folgendes durchzuführen:

- Rufen Sie synchron oder asynchron eine AWS Lambda-Funktion mit den nativen Funktionen `lambda_sync` oder `lambda_async` auf. Weitere Informationen finden Sie unter Aufrufen einer Lambda-Funktion aus einem Amazon Aurora MySQL-DB-Cluster.
- Laden Sie Daten aus Text- und XML-Dateien, die in einem Amazon Simple Storage Service (Amazon S3)-Bucket gespeichert sind, mithilfe des Befehls `LOAD DATA FROM S3` oder `LOAD XML FROM S3` in Ihren DB-Cluster. Weitere Informationen finden Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket.
- Speichern Sie Daten in die Textdateien, die in einem Amazon S3-Bucket aus Ihrem DB-Cluster gespeichert sind, mithilfe des Befehls `SELECT INTO OUTFILE S3`. Weitere Informationen finden Sie unter Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket.
- Automatisches Hinzufügen oder Entfernen von Aurora-Replicas mit Application Auto Scaling. Weitere Informationen finden Sie unter Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten.
- Führen Sie eine Stimmungsanalyse mit Amazon Comprehend oder eine breite Palette von Machine-Learning-Algorithmen mit SageMaker AI durch. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Machine Learning.

Mit Aurora können Sie auf andere AWS-Services mithilfe von AWS Identity and Access Management (IAM) zugreifen. Sie können die Erlaubnis erteilen, auf andere AWS-Services zuzugreifen, indem Sie eine IAM-Rolle mit den notwendigen Berechtigungen erstellen und diese dann Ihrem DB-Cluster zuordnen. Weitere Details und Anweisungen, wie Sie selbst Ihren Aurora-MySQL-DB-Cluster für den Zugriff auf andere AWS-Services berechtigen, finden Sie unter Autorisierung von Amazon Aurora MySQL zum Zugriff auf andere AWS-Services für Sie.

Autorisierung von Amazon Aurora MySQL zum Zugriff auf andere AWS-Services für Sie

Damit der Aurora-MySQL-DB-Cluster in Ihrem Auftrag auf andere Services zugreifen kann, erstellen und konfigurieren Sie eine AWS Identity and Access Management-(IAM)-Rolle. Diese Rolle

autorisiert Datenbankbenutzer in Ihrem DB-Cluster, auf andere AWS-Services zuzugreifen. Weitere Informationen finden Sie unter Einrichten von IAM-Rollen für den Zugriff auf AWS-Services.

Sie müssen auch Ihr Aurora-DB-Cluster so konfigurieren, dass ausgehende Verbindungen zum Ziel-AWS-Service erlaubt werden. Weitere Informationen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

Wenn Sie dies tun, können Ihre Datenbankbenutzer folgende Aktionen mit anderen AWS-Services durchführen:

- Rufen Sie synchron oder asynchron eine AWS Lambda-Funktion mit den nativen Funktionen `lambda_sync` oder `lambda_async` auf. Oder rufen Sie asynchron eine AWS Lambda-Funktion mit der Prozedur `mysql.lambda_async` auf. Weitere Informationen finden Sie unter Aufrufen einer Lambda-Funktion mit einer nativen Aurora MySQL-Funktion.
- Laden Sie Daten aus Text- und XML-Dateien, die in einem Amazon S3-Bucket gespeichert sind, mithilfe der Anweisungen `LOAD DATA FROM S3` oder `LOAD XML FROM S3` in Ihren DB-Cluster. Weitere Informationen finden Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket.
- Speichern Sie Daten aus Ihrem DB-Cluster in Textdateien, die in einem Amazon S3-Bucket gespeichert sind, mithilfe der Anweisung `SELECT INTO OUTFILE S3`. Weitere Informationen finden Sie unter Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket.
- Exportieren Sie Protokolldaten in Amazon CloudWatch Logs MySQL. Weitere Informationen finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.
- Automatisches Hinzufügen oder Entfernen von Aurora-Replicas mit Application Auto Scaling. Weitere Informationen finden Sie unter Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten.

Einrichten von IAM-Rollen für den Zugriff auf AWS-Services

Führen Sie die folgenden Schritte aus, um Ihrem Aurora-DB-Cluster zu erlauben, auf einen anderen AWS-Service zuzugreifen:

1. Erstellen Sie eine IAM-Zugriffsrichtlinie, die Berechtigungen für den AWS-Service erteilt. Weitere Informationen finden Sie unter den folgenden Themen.
   - Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen
   - Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS Lambda
   - Eine IAM-Richtlinie für den Zugriff auf CloudWatch Logs-Ressourcen erstellen

- Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS KMS

2. Erstellen Sie eine IAM-Rolle und fügen Sie die erstellte Zugriffsrichtlinie an. Weitere Informationen finden Sie unter Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.
3. Weisen Sie diese IAM-Rolle Ihrem Aurora-DB-Cluster zu. Weitere Informationen finden Sie unter Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster.

Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen

Aurora kann auf Amazon S3-Ressourcen zugreifen, um entweder Daten zu laden oder Daten aus einem Aurora-DB-Cluster zu speichern. Jedoch müssen Sie zuerst eine IAM-Zugriffsrichtlinie erstellen, die die Bucket- und Objektberechtigungen bereitstellt, um Aurora den Zugriff auf Amazon S3 zu erlauben.

In der folgenden Tabelle sind die Aurora-Funktionen gelistet, die in Ihrem Auftrag auf ein Amazon S3-Bucket zugreifen können, und das Minimum der erforderlichen Bucket- und Objektberechtigungen, die für jede Funktion benötigt werden.

Funktion	Bucket-Berechtigungen	Objektberechtigungen
`LOAD DATA FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`
`LOAD XML FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`

`SELECT INTO OUTFILE S3`	`ListBucket`	`AbortMultipartUpload`
		`DeleteObject`
		`GetObject`
		`ListMultipartUploadParts`
		`PutObject`

Die folgende Richtlinie fügt die Berechtigungen hinzu, die Aurora möglicherweise für den Zugriff auf einen Amazon S3-Bucket in Ihrem Namen benötigt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

> **Note**
>
> Achten Sie darauf, dass beide Einträge für den Wert `Resource` eingeschlossen sind. Aurora benötigt die Berechtigungen sowohl für den Bucket selbst als auch für alle Objekte im Bucket. Basierend auf Ihrem Anwendungsfall müssen Sie möglicherweise nicht alle Berechtigungen in der Beispielrichtlinie hinzufügen. Möglicherweise sind auch weitere Berechtigungen erforderlich. Wenn Ihr Amazon S3-Bucket beispielsweise verschlüsselt ist, müssen Sie `kms:Decrypt`-Berechtigungen hinzufügen.

Sie können die folgenden Schritte ausführen, um eine IAM-Zugriffsrichtlinie zu erstellen, die das Minimum der erforderlichen Berechtigungen für Aurora bereitstellt, um in Ihrem Auftrag auf einen Amazon S3-Bucket zuzugreifen. Um Aurora den Zugriff auf alle Ihre Amazon S3-Buckets zu erlauben, können Sie diese Schritte überspringen und die vordefinierte IAM-Richtlinie `AmazonS3ReadOnlyAccess` oder `AmazonS3FullAccess` verwenden, anstatt eigene zu erstellen.

So können Sie eine IAM-Zugriffsrichtlinie erstellen, um Zugriff auf Ihre Amazon S3-Ressourcen zu erlauben

1. Öffnen Sie die IAM-Managementkonsole.
2. Wählen Sie im Navigationsbereich Richtlinien.
3. Wählen Sie Richtlinie erstellen aus.
4. Wählen Sie auf der Registerkarte Visueller Editor die Option Service auswählen und dann S3 aus.
5. Wählen Sie unter Actions (Aktionen) die Option Expand all (Alle expandieren). Wählen Sie dann die Bucket-Berechtigungen und Objektberechtigungen aus, die für die IAM-Richtlinie benötigt werden.

   Objektberechtigungen sind Berechtigungen für Objektoperationen in Amazon S3 und müssen für Objekte in einem Bucket und nicht für das Bucket selbst erteilt werden. Weitere Informationen über Berechtigungen für Objektoperationen in Amazon S3 finden Sie unter Berechtigungen für Objektoperationen.
6. Wählen Sie Resources (Ressourcen) und dann Add ARN (ARN hinzufügen) für Bucket (Bucket) aus.
7. Geben Sie im Dialogfeld ARN(s) hinzufügen die Details zu Ihrer Ressource an, und wählen Sie Hinzufügen.

   Geben Sie das Amazon S3-Bucket an, wofür der Zugriff erlaubt werden soll. Wenn Sie beispielsweise Aurora den Zugriff auf den Amazon S3-Bucket mit dem Namen *`amzn-s3-demo-bucket`* gewähren möchten, dann stellen Sie den Amazon-Ressourcennamen (ARN)-Wert auf `arn:aws:s3:::`*`amzn-s3-demo-bucket`* ein.
8. Wenn die Ressource Objekt aufgelistet ist, wählen Sie ARN hinzufügen für Objekt.
9. Geben Sie im Dialogfeld ARN(s) hinzufügen die Details zu Ihrer Ressource an.

   Geben Sie für den Amazon S3-Bucket den Amazon S3-Bucket an, für den der Zugriff erlaubt werden soll. Sie können für das Objekt Beliebig auswählen, um Berechtigungen für alle Objekte im Bucket bereitzustellen.

> **Note**
>
> Sie können Amazon-Ressourcenname (ARN) auf einen spezifischen ARN-Wert einstellen, um Aurora nur den Zugriff auf spezifische Dateien oder Ordern in einem Amazon S3-Bucket zu erlauben. Weitere Informationen über das Definieren von Zugriffsrichtlinien für Amazon S3 finden Sie unter Verwaltung der Zugriffsberechtigungen zu Ihren Amazon S3-Ressourcen.

10. (Optional) Wählen Sie Add ARN (ARN hinzufügen) für Bucket, um der Richtlinie einen weiteren Amazon S3-Bucket hinzuzufügen, und wiederholen Sie die vorherigen Schritte für den Bucket.

> **Note**
>
> Sie können diesen Schritt wiederholen, um Ihrer Richtlinie die entsprechenden Bucket-Berechtigungsanweisungen für jeden Amazon S3-Bucket hinzuzufügen, auf den Aurora zugreifen soll. Alternativ können Sie auch Zugriff auf alle Buckets und Objekte in Amazon S3 erlauben.

11. Wählen Sie Richtlinie prüfen.
12. Geben Sie unter Name einen Namen für Ihre IAM-Richtlinie ein, z. B. `AllowAuroraToExampleBucket`. Sie verwenden diesen Namen, wenn Sie eine IAM-Rolle erstellen, um sie Ihrem Aurora-DB-Cluster zuzuweisen. Sie können auch einen optionalen Wert für Description (Beschreibung) hinzufügen.
13. Wählen Sie Richtlinie erstellen aus.
14. Führen Sie die Schritte unter au Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.

Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS Lambda

Sie können eine IAM-Richtlinie erstellen, die die Mindestberechtigungen vorsieht, die Aurora benötigt, um eine AWS Lambda Funktion in Ihrem Namen aufzurufen.

Die folgende Richtlinie fügt die Berechtigungen hinzu, die Aurora benötigt, um in Ihrem Namen eine AWS Lambda -Funktion aufzurufen.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAuroraToExampleFunction",
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-
east-1:123456789012:function:example_function"
    }
  ]
}
```

Sie können die folgenden Schritte verwenden, um eine IAM-Richtlinie zu erstellen, die die Mindestberechtigungen für Aurora bereitstellt, um eine AWS Lambda Funktion in Ihrem Namen aufzurufen. Damit Aurora all Ihre AWS Lambda Funktionen aufrufen kann, können Sie diese Schritte überspringen und die vordefinierte `AWSLambdaRole` Richtlinie verwenden, anstatt Ihre eigene zu erstellen.

So können Sie eine IAM-Zugriffrichtlinie erstellen, um den Aufruf Ihrer AWS Lambda -Funktionen zu genehmigen

1. Öffnen Sie die IAM-Konsole.
2. Wählen Sie im Navigationsbereich Richtlinien.
3. Wählen Sie Richtlinie erstellen aus.
4. Wählen Sie auf der Registerkarte Visueller Editor die Option Service auswählen und dann Lambda aus.
5. Wählen Sie für Actions (Aktionen) die Option Expand all (Alle erweitern) und danach die für die IAM-Richtlinie erforderlichen AWS Lambda -Berechtigungen aus.

   Stellen Sie sicher, dass `InvokeFunction` ausgewählt ist. Dies ist die Mindestberechtigung, die erforderlich ist, um Amazon Aurora das Aufrufen einer AWS Lambda Funktion zu ermöglichen.
6. Wählen Sie Ressourcen und dann ARN hinzufügen für Funktion.

7. Geben Sie im Dialogfeld ARN(s) hinzufügen die Details zu Ihrer Ressource an.

   Geben Sie die Lambda-Funktion an, wofür der Zugriff erlaubt werden soll. Wenn Sie beispielsweise Aurora den Zugriff auf eine Lambda-Funktion mit dem Namen `example_function` erlauben möchten, stellen Sie den ARN-Wert auf `arn:aws:lambda:::function:example_function` ein.

   Weitere Informationen zur Definition einer Zugriffsrichtlinie für AWS Lambda finden Sie unter Authentifizierung und Zugriffskontrolle für AWS Lambda.

8. Wählen Sie optional Zusätzliche Berechtigungen hinzufügen aus, um der Richtlinie eine weitere AWS Lambda Funktion hinzuzufügen, und wiederholen Sie die vorherigen Schritte für die Funktion.

   > **Note**
   >
   > Sie können dies wiederholen, um Ihrer Richtlinie für jede Funktion, auf die Aurora zugreifen soll, entsprechende AWS Lambda Funktionsberechtigungserklärungen hinzuzufügen.

9. Wählen Sie Richtlinie prüfen.
10. Stellen Sie Name auf einen Namen für Ihre IAM-Zugriffsrichtlinie ein, zum Beispiel `AllowAuroraToExampleFunction`. Sie verwenden diesen Namen, wenn Sie eine IAM-Rolle erstellen, um sie Ihrem Aurora-DB-Cluster zuzuweisen. Sie können auch einen optionalen Wert für Description (Beschreibung) hinzufügen.
11. Wählen Sie Richtlinie erstellen aus.
12. Führen Sie die Schritte unter au Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.

Eine IAM-Richtlinie für den Zugriff auf CloudWatch Logs-Ressourcen erstellen

Aurora kann auf CloudWatch Logs zugreifen, um Audit-Log-Daten aus einem Aurora-DB-Cluster zu exportieren. Sie müssen jedoch zunächst eine IAM-Richtlinie erstellen, die die Protokollgruppen- und Protokollstream-Berechtigungen bereitstellt, die Aurora den Zugriff auf CloudWatch Logs ermöglichen.

Die folgende Richtlinie fügt die Berechtigungen hinzu, die Aurora benötigt, um in Ihrem Namen auf Amazon CloudWatch Logs zuzugreifen, und die erforderlichen Mindestberechtigungen, um Protokollgruppen zu erstellen und Daten zu exportieren.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*"
        },
        {
            "Sid":
 "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutRetentionPolicy",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*"
        }
    ]
}
```

Sie können die ARNs in der Richtlinie enthaltenen Änderungen ändern, um den Zugriff auf eine bestimmte AWS Region und ein bestimmtes Konto zu beschränken.

Sie können die folgenden Schritte verwenden, um eine IAM-Richtlinie zu erstellen, die die Mindestberechtigungen für Aurora bereitstellt, um in Ihrem Namen auf CloudWatch Logs zuzugreifen. Um Aurora vollen Zugriff auf CloudWatch Logs zu gewähren, können Sie diese Schritte überspringen und die `CloudWatchLogsFullAccess` vordefinierte IAM-Richtlinie verwenden, anstatt Ihre eigene

zu erstellen. Weitere Informationen finden Sie unter Verwenden identitätsbasierter Richtlinien (IAM-Richtlinien) für CloudWatch Protokolle im CloudWatch Amazon-Benutzerhandbuch.

So erstellen Sie eine IAM-Richtlinie, um Zugriff auf Ihre Logs-Ressourcen zu gewähren CloudWatch

1. Öffnen Sie die IAM-Konsole.
2. Wählen Sie im Navigationsbereich Richtlinien.
3. Wählen Sie Richtlinie erstellen aus.
4. Wählen Sie auf der Registerkarte Visual Editor die Option Service auswählen und dann CloudWatchLogs aus.
5. Wählen Sie für Aktionen die Option Alle erweitern (rechts) und wählen Sie dann die Amazon CloudWatch Logs-Berechtigungen aus, die für die IAM-Richtlinie erforderlich sind.

   Vergewissern Sie sich, dass die folgenden Berechtigungen ausgewählt sind:

   - `CreateLogGroup`
   - `CreateLogStream`
   - `DescribeLogStreams`
   - `GetLogEvents`
   - `PutLogEvents`
   - `PutRetentionPolicy`
6. Wählen Sie Ressourcen und dann ARN hinzufügen für log-group (Protokollgruppe).
7. Geben Sie im Dialogfeld ARN(s) hinzufügen die folgenden Werte ein:

   - Region — Eine AWS Region oder *
   - Konto – Eine Kontonummer oder *
   - Protokollgruppenname – `/aws/rds/*`
8. Wählen Sie im Dialogfeld Add ARN(s) (ARN(s) hinzufügen) Add (Hinzufügen).
9. Wählen Sie ARN hinzufügen für log-stream (Protokollstream).
10. Geben Sie im Dialogfeld ARN(s) hinzufügen die folgenden Werte ein:

    - Region — Eine AWS Region oder *
    - Konto – Eine Kontonummer oder *
    - Protokollgruppenname – `/aws/rds/*`
    - Protokollstreamname – *

11. Wählen Sie im Dialogfeld Add ARN(s) (ARN(s) hinzufügen) Add (Hinzufügen).
12. Wählen Sie Richtlinie prüfen.
13. Stellen Sie Name auf einen Namen für Ihre IAM-Zugriffsrichtlinie ein, zum Beispiel `AmazonRDSCloudWatchLogs`. Sie verwenden diesen Namen, wenn Sie eine IAM-Rolle erstellen, um sie Ihrem Aurora-DB-Cluster zuzuweisen. Sie können auch einen optionalen Wert für Description (Beschreibung) hinzufügen.
14. Wählen Sie Richtlinie erstellen aus.
15. Führen Sie die Schritte unter au Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.

Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS KMS

Aurora kann auf die AWS KMS keys zugreifen, die für die Verschlüsselung ihrer Datenbanksicherungen verwendet wird. Sie müssen jedoch zunächst eine IAM-Richtlinie erstellen, die Aurora den Zugriff auf KMS-Schlüssel erlaubt.

Die folgende Richtlinie fügt die Berechtigungen hinzu, die Aurora für den Zugriff auf KMS-Schlüssel in Ihrem Namen benötigt.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAuroraToAccessKey",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/key-ID"
    }
  ]
}
```

Anhand der folgenden Schritte können Sie eine IAM-Richtlinie erstellen, die Aurora die erforderlichen Mindestberechtigungen für den Zugriff auf KMS-Schlüssel in Ihrem Namen erteilt.

So erstellen Sie eine IAM-Richtlinie zum Gewähren des Zugriffs auf Ihre KMS-Schlüssel

1. Öffnen Sie die IAM-Konsole.
2. Wählen Sie im Navigationsbereich Richtlinien.
3. Wählen Sie Richtlinie erstellen aus.
4. Wählen Sie auf der Registerkarte Visueller Editor die Option Service auswählen und dann KMS aus.
5. Wählen Sie unter Actions (Aktionen) die Option Write (Schreiben) und danach Decrypt (Entschlüsseln) aus.
6. Wählen Sie Ressourcen und danach ARN hinzufügen.
7. Geben Sie im Dialogfeld ARN(s) hinzufügen die folgenden Werte ein:
   - Region — Geben Sie die AWS Region ein, z. B. `us-west-2`
   - Konto – Geben Sie die Nummer des Benutzerkontos ein.
   - Log Stream Name - Geben Sie den KMS-Schlüsselbezeichner ein.
8. Wählen Sie im Dialogfeld Add ARN(s) (ARN(s) hinzufügen) Add (Hinzufügen).
9. Wählen Sie Richtlinie prüfen.
10. Stellen Sie Name auf einen Namen für Ihre IAM-Zugriffsrichtlinie ein, zum Beispiel `AmazonRDSKMSKey`. Sie verwenden diesen Namen, wenn Sie eine IAM-Rolle erstellen, um sie Ihrem Aurora-DB-Cluster zuzuweisen. Sie können auch einen optionalen Wert für Description (Beschreibung) hinzufügen.
11. Wählen Sie Richtlinie erstellen aus.
12. Führen Sie die Schritte unter au Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.

Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben

Nachdem Sie eine IAM-Zugriffsrichtlinie erstellt haben, um Aurora Zugriff auf AWS-Ressourcen zu erlauben, müssen Sie eine IAM-Rolle erstellen und der IAM-Zugriffsrichtlinie die neue IAM-Rolle anfügen.

Gehen Sie wie folgt vor, um eine IAM-Rolle zu erstellen, die Ihrem Amazon-RDS-Cluster die Kommunikation mit anderen AWS-Services für Sie ermöglicht.

So erstellen Sie eine IAM-Rolle, um Amazon RDS; den Zugriff auf AWS-Services zu ermöglichen:

1. Öffnen Sie die IAM-Konsole.
2. Wählen Sie im Navigationsbereich Roles aus.
3. Wählen Sie Create role (Rolle erstellen) aus.
4. Wählen Sie unter AWS-ServiceRDS.
5. Wählen Sie unter Select your use case (Anwendungsfall auswählen) die Option RDS – Add Role to Database (RDS Rolle zur Datenbank hinzufügen) aus.
6. Wählen Sie Weiter aus.
7. Geben Sie auf der Seite Attach permissions policies (Berechtigungsrichtlinien anfügen) den Namen Ihrer Richtlinie im Feld Search (Suchen) ein.
8. Wenn die zuvor definierte Richtlinie in der Liste angezeigt wird, wählen Sie diese entsprechend den Anweisungen in einem der folgenden Abschnitte aus:

   - Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen
   - Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS Lambda
   - Eine IAM-Richtlinie für den Zugriff auf CloudWatch Logs-Ressourcen erstellen
   - Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS KMS
9. Wählen Sie Weiter aus.
10. Geben Sie unter Role name (Rollenname) einen Namen für Ihre IAM-Rolle ein, z. B. `RDSLoadFromS3`. Sie können auch einen optionalen Wert für Description (Beschreibung) hinzufügen.
11. Wählen Sie Rolle erstellen aus.
12. Führen Sie die Schritte unter au Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster.

Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster

Weisen Sie die IAM-Rolle, die Sie in Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben erstellt haben, einem DB-Cluster zu, um Datenbankbenutzern in diesem DB-Cluster von Amazon Aurora den Zugriff auf andere AWS-Services zu erlauben. Sie können auch eine neue IAM-Rolle durch AWS erstellen lassen, indem Sie den Service direkt zuordnen.

> **Note**
>
> Sie können keine IAM-Rolle mit einem Aurora Serverless v1-DB-Cluster verknüpfen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Serverless v1.
> Sie können eine IAM-Rolle mit einem DB-Cluster von Aurora Serverless v2verknüpfen.

Sie müssen zwei Dinge tun, um eine IAM-Rolle mit einem DB-Cluster zu verbinden:

1. Fügen Sie die Rolle zu der Liste mit den zugewiesenen Rollen für ein DB-Cluster mithilfe der RDS-Konsole und dem AWS CLI-Befehl add-role-to-db-cluster oder der RDS-API-Operation AddRoleToDBCluster hinzu.

   Sie können maximal fünf IAM-Rollen für jeden Aurora-DB-Cluster hinzufügen.

2. Stellen Sie den Cluster-Level-Parameter für den zugehörigen AWS-Service auf den ARN für die zugeordnete IAM-Rolle ein.

   Die folgende Tabelle beschreibt die Cluster-Level-Parameter-Namen für die IAM-Rollen, die für den Zugriff auf andere AWS-Services verwendet werden.

Parameter auf Clusterebene	Beschreibung	
`aws_default_lambda_role`	Wird beim Aufrufen einer Lambda-Funktion aus dem DB-Cluster verwendet.	
`aws_default_logs_role`	Dieser Parameter ist nicht mehr erforderlich, um Protokolldaten von Ihrem DB-Cluster in Amazon CloudWatch Logs zu exportieren. Aurora MySQL verwendet jetzt eine serviceverknüpfte Rolle für die erforderlichen Berechtigungen. Weitere Informationen zu Service-verknüpften Rollen finden Sie unter Verwenden von serviceverknüpften Rollen für Amazon Aurora.	
`aws_default_s3_role`	Wird beim Aufrufen der Anweisung `LOAD DATA FROM S3`, `LOAD XML FROM S3` oder	

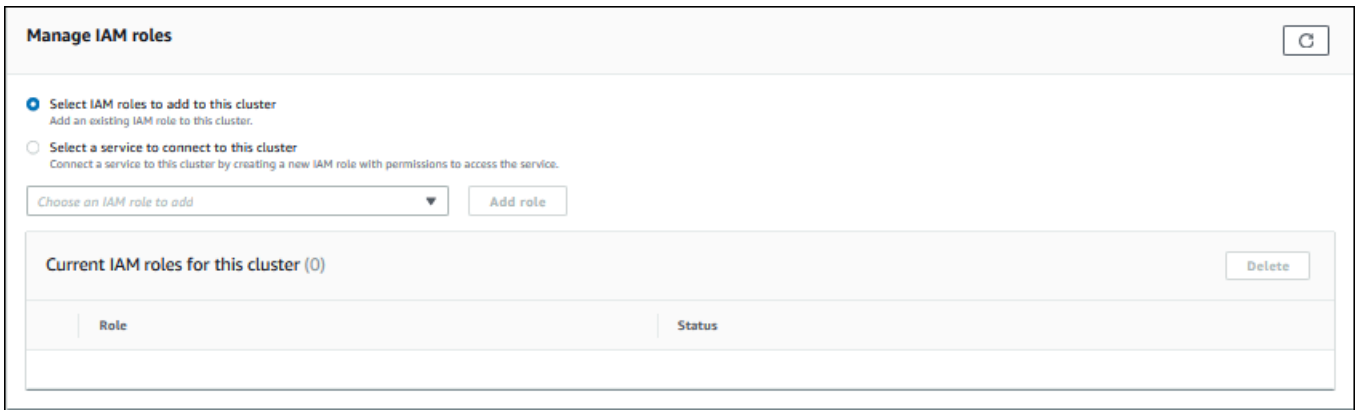
Parameter auf Clusterebene	Beschreibung	
	`SELECT INTO OUTFILE S3` aus Ihrem DB-Cluster verwendet. Bei Aurora MySQL Version 2 wird die in diesem Parameter festgelegte IAM-Rolle verwendet, wenn keine IAM-Rolle für `aurora_load_from_s3_role` oder `aurora_select_into_s3_role` für die entsprechende Anweisung festgelegt ist. Bei Aurora-MySQL-Version 3 wird die für diesen Parameter festgelegte IAM-Rolle immer verwendet.	
`aurora_load_from_s3_role`	Wird beim Aufrufen der Anweisung `LOAD DATA FROM S3` oder `LOAD XML FROM S3` aus Ihrem DB-Cluster verwendet. Wenn keine IAM-Rolle für diesen Parameter festgelegt ist, wird die in `aws_default_s3_role` festgelegte IAM-Rolle verwendet. Bei Aurora MySQL Version 3 ist dieser Parameter nicht verfügbar.	
`aurora_select_into_s3_role`	Wird beim Aufrufen der Anweisung `SELECT INTO OUTFILE S3` aus Ihrem DB-Cluster verwendet. Wenn keine IAM-Rolle für diesen Parameter festgelegt ist, wird die in `aws_default_s3_role` festgelegte IAM-Rolle verwendet. Bei Aurora MySQL Version 3 ist dieser Parameter nicht verfügbar.	

Gehen Sie wie folgt vor, um einer IAM-Rolle die Erlaubnis, Ihrem Amazon-RDS-Cluster die Kommunikation mit anderen AWS-Services für Sie zu ermöglichen, zuzuweisen.

Konsole

Sie müssen zwei Dinge tun, um eine IAM-Rolle einem Aurora-DB-Cluster mithilfe der Konsole zuzuweisen:

1. Öffnen Sie die RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Datenbanken aus.
3. Wählen Sie den Namen des Aurora-DB-Clusters, den Sie einer IAM-Rolle zuweisen möchten, aus, um die entsprechenden Details anzuzeigen.
4. Führen Sie auf der Registerkarte Konnektivität und Sicherheit im Abschnitt IAM-Rollen verwalten eine der folgenden Aktionen aus:
   - IAM-Rollen auswählen, die diesem Cluster hinzugefügt werden sollen (Standard)
   - Einen Service zum Herstellen einer Verbindung mit diesem Cluster auswählen



5. Wenn Sie eine vorhandene IAM-Rolle verwenden möchten, wählen Sie diese aus dem Menü aus und klicken Sie dann auf Add role (Rolle hinzufügen).

   Wenn die Rolle erfolgreich hinzugefügt wurde, wird ihr Status als `Pending` und dann als `Available` angezeigt.
6. So verbinden Sie einen Service direkt:
   a. Wählen Sie Select a service to connect to this cluster (Einen Service zum Herstellen einer Verbindung mit diesem Cluster auswählen) aus.

b. Wählen Sie den Service aus dem Menü aus und klicken Sie dann auf Connect service (Service verbinden).

c. Geben Sie unter Connect cluster to ***`Service Name`*** (Cluster mit „Servicename“ verbinden) den Amazon-Ressourcennamen (ARN) ein, der für die Verbindung mit dem Service verwendet werden soll, und wählen Sie dann Connect service (Service verbinden) aus.

AWS erstellt eine neue IAM-Rolle für die Verbindung mit dem Service. Der Status wird als `Pending` und dann als `Available` angezeigt.

7. (Optional) Wenn Sie die Zuordnung einer IAM-Rolle mit einem DB-Cluster aufheben und die zugehörige Berechtigung entziehen möchten, wählen Sie die Rolle und dann Delete (Löschen) aus.

So legen Sie den Cluster-Level-Parameter für die zugeordnete Rolle fest

1. Wählen Sie in der RDS-Konsole Parametergruppen im Navigationsbereich aus.
2. Wenn Sie bereits eine benutzerdefinierte DB-Parametergruppe verwenden, können Sie diese Gruppe auswählen, statt eine neue DB-Cluster-Parametergruppe zu erstellen. Wenn Sie die Standard-DB-Cluster-Parametergruppe verwenden, erstellen Sie eine neue DB-Cluster-Parametergruppe, wie in den folgenden Schritten beschrieben ist:

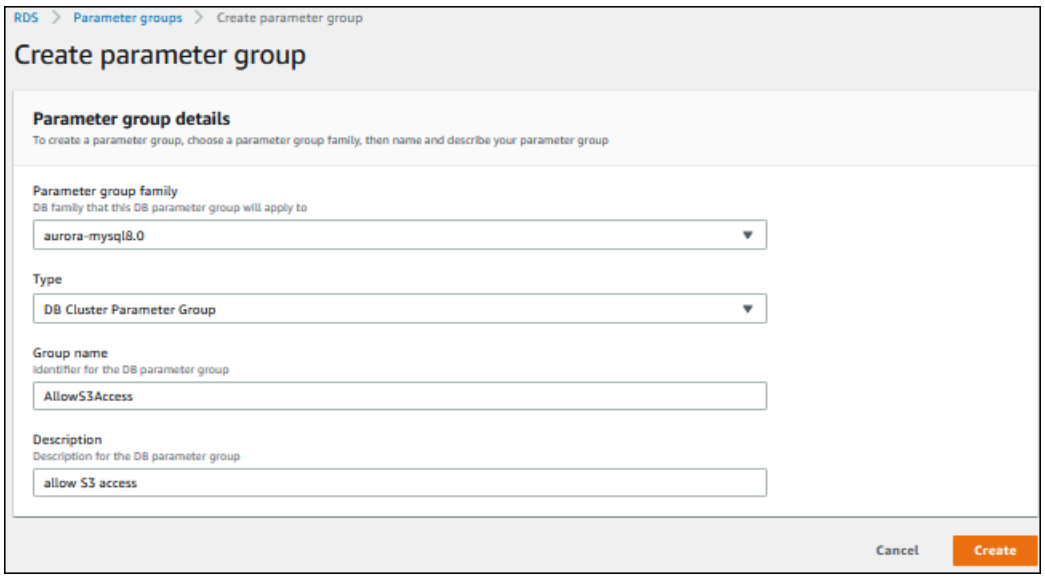
   a. Wählen Sie Parametergruppe erstellen.

   b. Wählen Sie für Parametergruppenfamilie für einen mit Aurora MySQL 5.6 kompatiblen DB-Cluster `aurora-mysql8.0`, oder `aurora-mysql5.7` für einen mit Aurora MySQL 5.7 kompatiblen DB-Cluster.

   c. Wählen Sie für Typ die Option DB-Cluster-Parametergruppe.

   d. Geben Sie für Gruppenname den Namen Ihrer neuen DB-Cluster-Parametergruppe ein.

   e. Geben Sie unter Beschreibung die Beschreibung für Ihre neue DB-Cluster-Parametergruppe ein.



f. Wählen Sie Create aus.

3. Wählen Sie auf der Seite Parameter Groups (Parametergruppen) Ihre DB-Cluster-Parametergruppe und danach die Option Edit (Bearbeiten) unter Parameter group actions (Parametergruppenaktionen) aus.
4. Legen Sie die entsprechenden Cluster-Level-Parameter für die zugehörigen ARN-Werte der IAM-Rolle fest.

   Sie können beispielsweise den Parameter `aws_default_s3_role` auf `arn:aws:iam::123456789012:role/AllowS3Access` einstellen.
5. Wählen Sie Änderungen speichern.
6. Um die DB-Cluster-Parametergruppe für Ihren DB-Cluster zu ändern, führen Sie die folgenden Schritte aus:

   a. Wählen Sie Datenbanken aus und wählen Sie dann Ihren Aurora DB-Cluster aus.

   b. Wählen Sie Ändern aus.

   c. Blättern Sie zu Datenbankoptionen und legen Sie DB-Cluster-Parametergruppe auf die DB-Cluster-Parametergruppe fest.

   d. Klicken Sie auf Weiter.

   e. Überprüfen Sie Ihre Änderungen, und klicken Sie anschließend auf Sofort anwenden.

   f. Wählen Sie Modify Cluster (Cluster ändern) aus.

   g. Wählen Sie Databases (Datenbanken) und anschließend die primäre Instance für Ihren DB-Cluster aus.

h. Wählen Sie unter Aktionen die Option Neustart aus.

Wenn die Instance neu gestartet wurde, wird Ihre IAM-Rolle Ihrem DB-Cluster zugewiesen.

Weitere Informationen zu Cluster-Parametergruppen finden Sie unter Aurora MySQL Konfigurationsparameter.

CLI

So weisen Sie eine IAM-Rolle einem DB-Cluster mithilfe der AWS CLI zu:

1. Rufen Sie den Befehl `add-role-to-db-cluster` aus der AWS CLI auf, um die ARNs für Ihre IAM-Rollen zum DB-Cluster hinzuzufügen, wie im Folgenden gezeigt.

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. Wenn Sie die Standard-DB-Cluster-Parametergruppe verwenden, erstellen Sie eine neue DB-Cluster-Parametergruppe. Wenn Sie bereits eine benutzerdefinierte DB-Parametergruppe verwenden, können Sie diese Gruppe verwenden, anstatt eine neue DB-Cluster-Parametergruppe zu erstellen.

   Rufen Sie den Befehl `create-db-cluster-parameter-group` aus der AWS CLI auf, um eine neue DB-Cluster-Parametergruppe zu erstellen, wie im Folgenden gezeigt.

```
PROMPT> aws rds create-db-cluster-parameter-group  --db-cluster-parameter-group-
name AllowAWSAccess \
     --db-parameter-group-family aurora5.7 --description "Allow access to Amazon S3
 and AWS Lambda"
```

   Für einen mit Aurora MySQL 5.7 kompatiblen DB-Cluster geben Sie `aurora-mysql5.7` für `--db-parameter-group-family` an. Für einen mit Aurora MySQL 8.0 kompatiblen DB-Cluster geben Sie `aurora-mysql8.0` für `--db-parameter-group-family` an.

3. Stellen Sie den/die entsprechenden Cluster-Level-Parameter und die zugehörigen ARN-Werte der IAM-Rolle in Ihrer DB-Cluster-Parametergruppe ein, wie im Folgenden gezeigt.

```
PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
 AllowAWSAccess \
    --parameters
 "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraS3Role,method=pending-reboot" \
    --parameters
 "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraLambdaRole,method=pending-reboot"
```

4. Ändern Sie das DB-Cluster, um die neue DB-Clusterparametergruppe zu verwenden und starten Sie das Cluster anschließend neu, wie im Folgenden gezeigt.

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
parameter-group-name AllowAWSAccess
PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
```

   Wenn die Instance neu gestartet ist, sind Ihre IAM-Rollen Ihrem DB-Cluster zugewiesen.

   Weitere Informationen zu Cluster-Parametergruppen finden Sie unter Aurora MySQL Konfigurationsparameter.

Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services

Um bestimmte andere AWS-Services mit Amazon Aurora nutzen zu können, muss die Netzwerkkonfiguration Ihres Aurora-DB-Clusters ausgehende Verbindungen zu Endpunkten für diese Services zulassen. Für die folgenden Operationen ist diese Netzwerkkonfiguration erforderlich.

- Aufrufen von AWS Lambda-Funktionen Weitere Informationen zu dieser Funktion finden Sie unter Aufrufen einer Lambda-Funktion mit einer nativen Aurora MySQL-Funktion.
- Zugriff auf Dateien von Amazon S3. Weitere Informationen zu dieser Funktion finden Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket und Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket.
- Zugriff auf AWS KMS-Endpunkte. AWS KMS-Zugriff ist erforderlich, um Datenbankaktivitäts-Streams mit Aurora MySQL verwenden zu können. Weitere Informationen zu dieser Funktion finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.

- Zugriff auf SageMaker-AI-Endpunkte SageMaker-AI-Zugriff ist erforderlich, um SageMaker-AI-Machine-Learning mit Aurora MySQL verwenden zu können. Weitere Informationen zu dieser Funktion finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.

Aurora gibt die folgenden Fehlermeldungen zurück, wenn keine Verbindung mit einem Service-Endpunkt hergestellt werden kann.

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to
 endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

Bei Datenbankaktivitäts-Streams, die Aurora MySQL verwenden, funktioniert der Aktivitäts-Stream nicht mehr, wenn der DB-Cluster nicht auf den AWS KMS-Endpunkt zugreifen kann. Sie werden von Aurora über dieses Problem mithilfe von RDS-Ereignissen benachrichtigt.

Wenn Sie bei der Nutzung der entsprechenden AWS-Services auf diese Meldungen stoßen, prüfen Sie, ob Ihr Aurora-DB-Cluster öffentlich oder privat ist. Wenn Ihr Aurora-DB-Cluster privat ist, müssen Sie es konfigurieren, um die Verbindungen zu aktivieren.

Damit ein Aurora-DB-Cluster öffentlich ist, muss es als öffentlich zugänglich markiert sein. Wenn Sie in den Details für das DB-Cluster in der AWS-Managementkonsole nachsehen, sollte für Publicly Accessible (Öffentlich zugänglich) Ja ausgewählt sein, wenn dies der Fall ist. Das DB-Cluster muss sich auch in einem Amazon VPC-öffentlichen Subnetz befinden. Weitere Informationen über öffentlich zugängliche DB-Instances finden Sie unter Arbeiten mit einer DB-Cluster in einer VPC. Weitere Informationen zu öffentlichen Amazon VPC-Subnetzen finden Sie unter Ihre VPC und Subnetze.

Wenn Ihr Aurora-DB-Cluster nicht öffentlich zugänglich ist und sich nicht in einem öffentlichen VPC-Subnetz befindet, ist er privat. Möglicherweise verfügen Sie über einen DB-Cluster, der privat ist, und möchten eine der Funktionen nutzen, für die diese Netzwerkkonfiguration erforderlich ist. Konfigurieren Sie dazu den Cluster so, dass er sich mit Internetadressen über Network Address Translation (NAT) verbinden kann. Als Alternative zu Amazon S3, Amazon SageMaker AI und AWS Lambda können Sie stattdessen die VPC so konfigurieren, dass sie einen VPC-Endpunkt für den anderen Service besitzt, der der Routing-Tabelle des DB-Clusters zugeordnet ist, vgl. Arbeiten mit einer DB-Cluster in einer VPC. Weitere Informationen über das Konfigurieren von NAT in Ihrer VPC finden Sie unter NAT Gateways. Weitere Informationen über das Konfigurieren von VPC-Endpunkten

finden Sie unter VPC-Endpunkte. Sie können auch einen S3-Gateway-Endpunkt erstellen, um auf Ihren S3-Bucket zuzugreifen. Weitere Informationen finden Sie unter Gateway-Endpunkte für Amazon S3.

Möglicherweise müssen Sie auch die kurzlebigen Ports für Ihre Netzwerkzugriffssteuerungslisten (ACLs) in den ausgehenden Regeln für Ihre VPC-Sicherheitsgruppe öffnen. Weitere Informationen zu kurzlebigen Ports für Netzwerk-ACLs finden Sie unter Kurzlebige Ports im Benutzerhandbuch für Amazon Virtual Private Cloud.

Verwandte Themen

- Integration von Aurora mit anderen AWS Diensten
- Verwalten eines Amazon Aurora-DB-Clusters

Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket

Sie können den Ausdruck `LOAD DATA FROM S3` oder `LOAD XML FROM S3` verwenden, um Daten aus Dateien zu laden, die in einem Amazon S3-Bucket gespeichert sind. In Aurora MySQL werden die Dateien zuerst auf der lokalen Festplatte gespeichert und dann in die Datenbank importiert. Nachdem die Importe in die Datenbank abgeschlossen sind, werden die lokalen Dateien gelöscht.

> **Note**
>
> Für Aurora Serverless v1 wird das Laden von Daten aus Textdateien in eine Tabelle nicht unterstützt. Für Aurora Serverless v2 wird sie unterstützt.

Inhalt

- Gewähren von Zugriff auf Amazon S3 für Aurora
- Erteilen von Berechtigungen für das Laden von Daten in Amazon Aurora MySQL
- Angeben des Pfads (URI) zu einem Amazon-S3-Bucket
- LOAD DATA FROM S3
  - Syntax
  - Parameter
  - Verwenden eines Manifests für zu ladende Dateien

    - Überprüfen der geladenen Dateien mit der Tabelle „aurora_s3_load_history“
  - Beispiele
- LOAD XML FROM S3
  - Syntax
  - Parameter

Gewähren von Zugriff auf Amazon S3 für Aurora

Bevor Sie Daten aus einem Amazon S3-Bucket laden können, müssen Sie Ihrem Aurora MySQL-DB-Cluster die Berechtigung für den Zugriff auf Amazon S3 erteilen.

So gewähren Sie Aurora MySQL Zugriff auf Amazon S3:

1. Erstellen Sie eine AWS Identity and Access Management-(IAM)-Richtlinie, die die Bucket- und Objektberechtigungen bereitstellt, die Ihrem Aurora-MySQL-DB-Cluster gestatten, auf Amazon S3 zuzugreifen. Detaillierte Anleitungen finden Sie unter Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen.

   > **Note**
   >
   > In Aurora MySQL Version 3.05 und höher können Sie Objekte laden, die mit vom Kunden verwalteten AWS KMS keys verschlüsselt sind. Nehmen Sie dazu die Berechtigung `kms:Decrypt` in Ihre IAM-Richtlinie auf. Weitere Informationen finden Sie unter Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS KMS.
   > Zum Laden von Objekten, die mit Von AWS verwaltete Schlüssel oder von Amazon S3 verwalteten Schlüsseln (SSE-S3) verschlüsselt sind, benötigen Sie diese Berechtigung nicht.

2. Erstellen Sie eine IAM-Rolle und fügen Sie der neuen IAM-Rolle die IAM-Richtlinie hinzu, die Sie in Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen erstellt haben. Detaillierte Anweisungen finden Sie unter Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.
3. Stellen Sie sicher, dass der DB-Cluster eine benutzerdefinierte DB-Cluster-Parametergruppe verwendet.

   Weitere Informationen über das Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe finden Sie unter Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora.

4. Legen Sie in Aurora MySQL Version 2 einen der DB-Cluster-Parameter `aurora_load_from_s3_role` oder `aws_default_s3_role` auf den Amazon-Ressourcennamen (ARN) der neuen IAM-Rolle fest. Wenn für `aurora_load_from_s3_role` keine IAM-Rolle angegeben wird, verwendet Aurora die unter `aws_default_s3_role` angegebene IAM-Rolle.

   Verwenden Sie `aws_default_s3_role` in Aurora MySQL Version 3.

   Wenn das Cluster Teil einer globalen Aurora-Datenbank ist, legen Sie diesen Parameter für jedes Aurora-Cluster in der globalen Datenbank fest. Auch wenn nur das primäre Cluster in einer globalen Aurora-Datenbank Daten laden kann, kann ein anderes Cluster durch den Failover-Mechanismus zum primären Cluster hochgestuft werden.

   Weitere Informationen zu DB-Cluster-Parametern finden Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter.
5. Weisen Sie die Rolle, die Sie in Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben erstellt haben, diesem DB-Cluster zu, um Datenbankbenutzern in einem Aurora MySQL DB-Cluster den Zugriff auf Amazon S3 zu erlauben. Bei einer globalen Aurora-Datenbank verknüpfen Sie die Rolle mit jedem Aurora-Cluster in der globalen Datenbank. Weitere Informationen zum Zuordnen einer IAM-Rolle zu einem DB-Cluster finden Sie unter Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster.
6. Konfigurieren Sie Ihren Aurora MySQL-DB-Cluster so, dass er ausgehende Verbindungen zu Amazon S3 zulässt. Detaillierte Anweisungen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

   Wenn Ihr -DB-Cluster nicht öffentlich zugänglich ist und sich nicht in einem öffentlichen VPC-Subnetz befindet, ist er privat. Sie können einen S3-Gateway-Endpunkt für den Zugriff auf Ihren S3-Bucket erstellen. Weitere Informationen finden Sie unter Gateway-Endpunkte für Amazon S3.

   Bei einer globalen Aurora-Datenbank aktivieren Sie ausgehende Verbindungen für jeden Aurora-Cluster in der globalen Datenbank.

Erteilen von Berechtigungen für das Laden von Daten in Amazon Aurora MySQL

Der Datenbankbenutzer, der die `LOAD DATA FROM S3`- oder `LOAD XML FROM S3`-Anweisung ausgibt, muss über eine bestimmte Rolle oder Berechtigung verfügen, um eine der Anweisungen auszugeben. In Aurora MySQL Version 3 gewähren Sie die Rolle `AWS_LOAD_S3_ACCESS`. In Aurora MySQL Version 2 gewähren Sie die Berechtigung `LOAD FROM S3`. Dem Administratorbenutzer für

ein DB-Cluster wird die entsprechende Rolle oder -Berechtigung standardmäßig gewährt. Sie können die Berechtigung einem anderen Benutzer erteilen, indem Sie eine der folgenden Anweisungen verwenden.

Verwenden Sie die folgende Anweisung für Aurora MySQL Version 3:

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Wenn Sie die Rollentechnik in Aurora MySQL Version 3 verwenden, können Sie die Rolle auch mit der Anweisung `SET ROLE role_name` oder `SET ROLE ALL` aktivieren. Wenn Sie mit dem MySQL 8.0-Rollensystem nicht vertraut sind, finden Sie in Rollenbasiertes Berechtigungsmodell weitere Informationen. Weitere Informationen finden Sie unter Verwenden von Rollen im MySQL-Referenzhandbuch.
> Dies gilt nur für die aktuelle aktive Sitzung. Wenn Sie sich wieder verbinden, müssen Sie die `SET ROLE`-Anweisung noch einmal ausführen, um Berechtigungen zu gewähren. Weitere Informationen finden Sie unter SET ROLE-Anweisung im MySQL-Referenzhandbuch.
> Sie können den DB-Cluster-Parameter `activate_all_roles_on_login` zum automatischen Aktivieren aller Rollen verwenden, wenn ein Benutzer eine Verbindung mit einer DB-Instance herstellt. Wenn dieser Parameter festgelegt ist, müssen Sie die `SET ROLE`-Anweisung in der Regel nicht explizit aufrufen, um eine Rolle zu aktivieren. Weitere Informationen finden Sie unter activate_all_roles_on_login im MySQL-Referenzhandbuch.
> Sie müssen jedoch `SET ROLE ALL` zu Beginn einer gespeicherten Prozedur explizit aufrufen, um die Rolle zu aktivieren, wenn die gespeicherte Prozedur von einem anderen Benutzer aufgerufen wird.

Verwenden Sie die folgende Anweisung für Aurora MySQL Version 2:

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

Die Rolle `AWS_LOAD_S3_ACCESS` und die Berechtigung `LOAD FROM S3` sind spezifisch für Amazon Aurora und nicht für externe MySQL-Datenbanken oder DB-Instances von RDS für MySQL verfügbar. Wenn Sie die Replikation zwischen einem Aurora-DB-Cluster als Replikationsquelle und einer MySQL-Datenbank als Replikations-Client eingerichtet haben, führt die `GRANT`-Anweisung für die Rolle oder Berechtigung dazu, dass die Replikation mit einem Fehler beendet wird. Sie können diese Fehlermeldung ignorieren und mit der Replikation fortfahren. Verwenden Sie das

Verfahren mysql_rds_skip_repl_error, um die Fehlermeldung für eine Instance von RDS für MySQL zu überspringen. Um den Fehler in einer externen MySQL-Datenbank zu überspringen, verwenden Sie die Systemvariable slave_skip_errors (Aurora MySQL Version 2) oder die Systemvariable replica_skip_errors (Aurora MySQL Version 3).

> **Note**
>
> Der Datenbankbenutzer muss über `INSERT`-Berechtigungen für die Datenbank verfügen, in die er Daten lädt.

Angeben des Pfads (URI) zu einem Amazon-S3-Bucket

Mit folgender Syntax wird der Pfad (URI) zu Dateien angegeben, die in einem Amazon-S3-Bucket gespeichert sind.

```
s3-region://amzn-s3-demo-bucket/file-name-or-prefix
```

Der Pfad enthält die folgenden Werte:

- `region` (optional) – Die AWS-Region mit dem Amazon-S3-Bucket, aus dem Daten geladen werden sollen. Dieser Wert ist optional. Falls Sie keinen Wert für `region` angeben, lädt Aurora die Datei aus Amazon S3 in derselben Region wie Ihr DB-Cluster.
- `bucket-name` – Der Name des Amazon S3-Buckets mit den zu ladenden Daten. Objekt-Präfixe, die einen virtuellen Ordnerpfad identifizieren, werden unterstützt.
- `file-name-or-prefix` – Der Name der Amazon S3-Textdatei oder XML-Datei oder ein Präfix, das eine oder mehrere Text- oder XML-Dateien zum Laden bestimmt. Sie können auch eine Manifestdatei festlegen, die eine oder mehrere Textdateien zum Laden angibt. Weitere Informationen darüber, wie Sie mit einer Manifestdatei Textdateien aus Amazon S3 laden, finden Sie unter Verwenden eines Manifests für zu ladende Dateien.

So kopieren Sie den URI für Dateien in einem S3-Bucket

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-S3-Konsole unter https://console.aws.amazon.com/s3/.
2. Wählen Sie im Navigationsbereich Buckets und dann den Bucket aus, dessen URI Sie kopieren möchten.

3. Wählen Sie das Präfix oder die Datei aus, die Sie aus S3 laden möchten.
4. Wählen Sie S3-URI kopieren aus.

LOAD DATA FROM S3

Mit der Anweisung `LOAD DATA FROM S3` können Sie Daten in jedem Textdateiformat laden, das von der MySQL-Anweisung LOAD DATA INFILE unterstützt wird (z. B. durch Komma-getrennte Textdaten). Komprimierte Dateien werden nicht unterstützt.

> **Note**
>
> Stellen Sie sicher, dass Ihr Aurora-MySQL-DB-Cluster ausgehende Verbindungen zu S3 zulässt. Weitere Informationen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

Syntax

```
LOAD DATA [FROM] S3 [FILE | PREFIX | MANIFEST] 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

> **Note**
>
> In Aurora MySQL Version 3.05 und höher ist das Schlüsselwort `FROM` optional.

Parameter

Die Anweisung `LOAD DATA FROM S3` verwendet die folgenden erforderlichen und optionalen Parameter. Weitere Informationen zu einigen der Parameter finden Sie unter LOAD DATA-Anweisung in der MySQL-Dokumentation.

FILE | PREFIX | MANIFEST

Gibt an, ob die Daten aus einer einzelnen Datei, aus allen Dateien mit einem bestimmten Präfix oder aus allen Dateien in einem angegebenen Manifest geladen werden sollen. `FILE` ist der Standardwert.

S3-URI

Gibt den URI der zu ladenden Text- oder Manifestdatei oder das zu verwendende Amazon-S3-Präfix an. Legen Sie den URI mithilfe der unter beschriebenen Syntax fes Angeben des Pfads (URI) zu einem Amazon-S3-Bucket.

REPLACE | IGNORE

Gibt an, welche Aktion ausgeführt werden soll, falls eine Eingabezeile die gleichen eindeutigen Schlüsselwerte aufweist wie eine vorhandene Zeile in der Datenbanktabelle.

- Geben Sie `REPLACE` an, wenn die Eingabezeile die bestehende Zeile überschreiben soll.
- Geben Sie `IGNORE` an, wenn die Eingabezeile verworfen werden soll.

INTO TABLE

Gibt den Namen der Datenbanktabelle an, in welche die Eingabezeilen geladen werden sollen.

PARTITION

Gibt vor, dass alle Eingabezeilen in die Partitionen, die in der Liste mit durch Komma getrennten Partitionsnamen angegeben sind, eingefügt werden müssen. Lässt sich eine Eingabezeile nicht in eine der angegebenen Partitionen einfügen, schlägt die Anweisung fehl und eine Fehlermeldung wird zurückgegeben.

CHARACTER SET

Gibt den Zeichensatz der Daten in der Eingabedatei an.

FIELDS | COLUMNS

Gibt an, wie die Felder oder Spalten in der Eingabedatei getrennt werden. Für Felder wird standardmäßig ein Tabulator als Trennzeichen verwendet.

LINES

Gibt an, wie die Zeilen in der Eingabedatei getrennt werden. Zeilen werden standardmäßig durch ein Zeilenumbruchzeichen (`'\n'`) getrennt.

IGNORE ***number*** LINES | ROWS

Gibt an, dass eine bestimmte Anzahl an Zeilen am Anfang der Eingabedatei ignoriert werden sollen. Sie können beispielsweise mit `IGNORE 1 LINES` die erste Headerzeile mit den Spaltennamen überspringen oder mit `IGNORE 2 ROWS` die ersten beiden Datenzeilen der Eingabedatei auslassen. Wenn Sie auch `PREFIX` verwenden, überspringt `IGNORE` eine bestimmte Anzahl von Zeilen am Anfang der ersten Eingabedatei.

col_name_or_user_var, ...

Gibt eine durch Komma getrennte Liste mit einem oder mehreren Spaltennamen bzw. einer oder mehreren Benutzervariablen an, die die Namen der zu ladenden Elemente identifizieren. Der Name einer Benutzervariable, der für diesen Zweck verwendet wird, muss mit dem Namen eines Elements aus der Textdatei übereinstimmen, welcher ein @ als Präfix hat. Sie können Benutzervariablen einsetzen, um die entsprechenden Feldwerte für die spätere Wiederverwendung zu speichern.

Beispielsweise wird mit folgender Anweisung die erste Spalte aus der Eingabedatei in die erste Spalte von `table1` geladen, zudem wird der Wert der Spalte `table_column2` in `table1` auf den Eingabewert der zweiten Spalte geteilt durch 100 gesetzt.

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

SET

Gibt eine durch Komma getrennte Liste mit Zuweisungsvorgängen an, die die Werte von Spalten in der Tabelle auf Werte setzen, die nicht in der Eingabedatei enthalten sind.

Beispielsweise setzt die folgende Anweisung die ersten beiden Spalten von `table1` auf die Werte in den ersten beiden Spalten aus der Eingabedatei und anschließend den Wert von `column3` in `table1` auf den aktuellen Zeitstempel.

```
LOAD DATA FROM S3  's3://amzn-s3-demo-bucket/data.txt'
```

```
    INTO TABLE table1
    (column1, column2)
    SET column3 = CURRENT_TIMESTAMP;
```

Auf der rechten Seite von SET-Zuweisungen können Sie Unterabfragen formulieren. Für eine Unterabfrage, die einen Wert zurückgibt, der einer Spalte zugewiesen werden soll, kommt ausschließlich eine skalare Unterabfrage infrage. Des Weiteren können Sie mit einer Unterabfrage keine Daten aus der zu ladenden Tabelle auswählen.

Sie können nicht das Schlüsselwort LOCAL der Anweisung LOAD DATA FROM S3 verwenden, wenn Sie Daten aus einem Amazon-S3-Bucket laden.

Verwenden eines Manifests für zu ladende Dateien

Sie können mit der Anweisung LOAD DATA FROM S3 und dem Schlüsselwort MANIFEST eine Manifestdatei im JSON-Format angeben, in der die Textdateien aufgeführt sind, die in eine Tabelle im DB-Cluster geladen werden sollen.

Das folgende JSON-Schema beschreibt das Format und den Inhalt einer Manifestdatei.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false",
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
                        "maxLength": 1024,
                        "minLength": 1,
```

```
                        "type": "string"
                    }
                },
                "required": [
                    "url"
                ],
                "type": "object"
            },
            "type": "array",
            "uniqueItems": true
        }
    },
    "required": [
        "entries"
    ],
    "type": "object"
}
```

Jeder `url`-Wert im Manifest muss eine URL mit dem Bucket-Namen und dem vollständigen Objektpfad zur Datei angeben (nicht nur ein Präfix). Sie können ein Manifest verwenden, um Dateien aus verschiedenen Buckets, verschiedenen Regionen oder mit unterschiedlichen Präfixen zu laden. Falls in der URL keine Region angegeben ist, wird die Region des Ziel-DB-Clusters in Aurora verwendet. Das folgende Beispiel zeigt eine Manifestdatei, mit der vier Dateien aus unterschiedlichen Buckets geladen werden.

```
{
  "entries": [
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":true
    },
    {
      "url":"s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory":true
    },
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":false
    },
    {
      "url":"s3://aurora-bucket/2013-10-05-customerdata"
    }
```

```
  ]
}
```

Das optionale Flag `mandatory` gibt an, ob `LOAD DATA FROM S3` eine Fehlermeldung zurückgeben soll, wenn die Datei nicht gefunden wird. Das Flag `mandatory` ist standardmäßig auf `false` gesetzt. Wenn keine Dateien gefunden werden, wird `mandatory` beendet, und zwar unabhängig vom `LOAD DATA FROM S3`-Wert.

Manifestdateien können eine beliebige Erweiterung haben. Im folgenden Beispiel wird die Anweisung `LOAD DATA FROM S3` mit dem Manifest aus dem vorherigen Beispiel ausgeführt, das den Namen **`customer.manifest`** trägt.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
    INTO TABLE CUSTOMER
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL);
```

Nachdem die Anweisung beendet ist, wird ein Eintrag für jede erfolgreich geladene Datei in die Tabelle `aurora_s3_load_history` geschrieben.

Überprüfen der geladenen Dateien mit der Tabelle „aurora_s3_load_history“

Mit jeder erfolgreich ausgeführten Anweisung `LOAD DATA FROM S3` wird die Tabelle `aurora_s3_load_history` im Schema `mysql` mit einem Eintrag für jede geladene Datei aktualisiert.

Nach der Ausführung der Anweisung `LOAD DATA FROM S3` können Sie überprüfen, welche Dateien geladen wurden. Dazu führen Sie eine Abfrage für die Tabelle `aurora_s3_load_history` aus. Um die Dateien anzuzeigen, die aus einer Iteration der Anweisung geladen wurden, verwenden Sie die Klausel `WHERE`, um die Datensätze auf dem Amazon S3-URI für die in der Anweisung verwendete Manifestdatei zu filtern. Wenn Sie zuvor dieselbe Manifestdatei verwendet haben, filtern Sie die Ereignisse mithilfe des Felds `timestamp`.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

In der folgenden Tabelle werden die Felder der Tabelle `aurora_s3_load_history` beschrieben.

Feld	Beschreibung
`load_prefix`	Der URI, der in der Load-Anweisung angegeben wurde. Dieser URI kann alle der folgenden Elemente abbilden: • Eine einzelne Datei für ein Statement `LOAD DATA FROM S3 FILE` • Ein Amazon S3-Präfix, das auf mehrere Dateien für eine `LOAD DATA FROM S3 PREFIX`-Anweisung verweist • Eine einzelne Manifestdatei, die die zu ladenden Namen der Dateien für ein Statement `LOAD DATA FROM S3 MANIFEST` enthält
`file_name`	Der Name der Datei, die – mithilfe des im Feld `load_prefix` angegebenen URI – aus Amazon S3 in Aurora geladen wurde.
`version_number`	Die Versionsnummer der im Feld `file_name` bezeichneten Datei, die geladen wurde, wenn der Amazon S3-Bucket eine Versionsnummer hat.
`bytes_loaded`	Die Größe der geladenen Datei in Bytes.
`load_timestamp`	Der Zeitstempel, zu dem die Anweisung `LOAD DATA FROM S3` abgeschlossen ist.

Beispiele

Mit folgender Anweisung werden Daten aus einem Amazon S3-Bucket geladen, der sich in derselben Region befindet wie der Aurora-DB-Cluster. Die Anweisung liest die durch Kommata getrennten Daten der Datei `customerdata.txt` im Amazon-S3-Bucket *`amzn-s3-demo-bucket`* und lädt dann diese Daten in die Tabelle `store-schema.customer-table`.

```
LOAD DATA FROM S3 's3://amzn-s3-demo-bucket/customerdata.csv'
    INTO TABLE store-schema.customer-table
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

Mit folgender Anweisung werden Daten aus einem Amazon S3-Bucket geladen, der sich in einer anderen Region befindet als der Aurora-DB-Cluster. Die Anweisung liest die durch Kommata getrennten Daten aller Dateien, die mit dem Objektpräfix `employee-data` im Amazon-S3-Bucket *`amzn-s3-demo-bucket`* in der Region `us-west-2` übereinstimmen, und lädt dann diese Daten in die Tabelle `employees`.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://amzn-s3-demo-bucket/employee_data'
    INTO TABLE employees
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

Mit folgender Anweisung werden Daten aus Dateien, die in einer JSON-Manifestdatei mit dem Namen "q1_sales.json" angegeben sind, in die Tabelle `sales` geladen.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://amzn-s3-demo-bucket1/q1_sales.json'
    INTO TABLE sales
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

Mit der Anweisung `LOAD XML FROM S3` können Sie Daten aus XML-Dateien, die in einem Amazon S3-Bucket gespeichert sind, in drei verschiedenen XML-Formaten laden:

- Spaltennamen als Attribute des Elements `<row>`. Der Attributwert bestimmt die Inhalte des Tabellenfeldes.

  ```
  <row column1="value1" column2="value2" .../>
  ```

- Spaltennamen als untergeordnete Elemente des Elements `<row>`. Der Wert des untergeordneten Elements bestimmt die Inhalte des Tabellenfeldes.

  ```
  <row>
    <column1>value1</column1>
    <column2>value2</column2>
  </row>
  ```

- Spaltennamen im Attribut `name` des Elements `<field>` im Element `<row>`. Der Wert des Elements `<field>` bestimmt die Inhalte des Tabellenfeldes.

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

Syntax

```
LOAD XML FROM S3 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<element-name>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

Parameter

Die Anweisung `LOAD XML FROM S3` verwendet die folgenden erforderlichen und optionalen Parameter. Weitere Informationen zu einigen der Parameter finden Sie unter LOAD XML-Anweisung in der MySQL-Dokumentation.

FILE | PREFIX

Gibt an, ob die Daten aus einer einzelnen Datei oder aus allen Dateien mit einem bestimmten Präfix geladen werden sollen. `FILE` ist der Standardwert.

REPLACE | IGNORE

Gibt an, welche Aktion ausgeführt werden soll, falls eine Eingabezeile die gleichen eindeutigen Schlüsselwerte aufweist wie eine vorhandene Zeile in der Datenbanktabelle.

- Geben Sie `REPLACE` an, wenn die Eingabezeile die bestehende Zeile überschreiben soll.
- Geben Sie `IGNORE` an, wenn Sie die Eingabezeile verwerfen möchten. `IGNORE` ist der Standardwert.

INTO TABLE

Gibt den Namen der Datenbanktabelle an, in welche die Eingabezeilen geladen werden sollen.

PARTITION

Gibt vor, dass alle Eingabezeilen in die Partitionen, die in der Liste mit durch Komma getrennten Partitionsnamen angegeben sind, eingefügt werden müssen. Lässt sich eine Eingabezeile nicht in eine der angegebenen Partitionen einfügen, schlägt die Anweisung fehl und eine Fehlermeldung wird zurückgegeben.

CHARACTER SET

Gibt den Zeichensatz der Daten in der Eingabedatei an.

ROWS IDENTIFIED BY

Gibt den Elementnamen an, der eine Zeile in der Eingabedatei bestimmt. Der Standardwert ist `<row>`.

IGNORE ***number*** LINES | ROWS

Gibt an, dass eine bestimmte Anzahl an Zeilen am Anfang der Eingabedatei ignoriert werden sollen. Sie können beispielsweise mit `IGNORE 1 LINES` die erste Zeile in der Textdatei überspringen oder mit `IGNORE 2 ROWS` die ersten beiden Datenzeilen der XML-Eingabedatei auslassen.

field_name_or_user_var, ...

Gibt eine durch Komma getrennte Liste mit einem oder mehreren XML-Elementnamen bzw. einer oder mehreren Benutzervariablen an, die die Namen der zu ladenden Elemente identifizieren. Der Name einer Benutzervariable, die für diesen Zweck verwendet wird, muss mit dem Namen eines Elements aus der XML-Datei übereinstimmen, welcher ein @ als Präfix hat. Sie können Benutzervariablen einsetzen, um die entsprechenden Feldwerte für die spätere Wiederverwendung zu speichern.

Beispielsweise wird mit folgender Anweisung die erste Spalte aus der Eingabedatei in die erste Spalte von `table1` geladen, zudem wird der Wert der Spalte `table_column2` in `table1` auf den Eingabewert der zweiten Spalte geteilt durch 100 gesetzt.

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, @var1)
```

```
   SET table_column2 = @var1/100;
```

SET

Gibt eine durch Komma getrennte Liste mit Zuweisungsvorgängen an, die die Werte von Spalten in der Tabelle auf Werte setzen, die nicht in der Eingabedatei enthalten sind.

Beispielsweise setzt die folgende Anweisung die ersten beiden Spalten von `table1` auf die Werte in den ersten beiden Spalten aus der Eingabedatei und anschließend den Wert von `column3` in `table1` auf den aktuellen Zeitstempel.

```
LOAD XML FROM S3 's3://amzn-s3-demo-bucket/data.xml'
   INTO TABLE table1
   (column1, column2)
   SET column3 = CURRENT_TIMESTAMP;
```

Auf der rechten Seite von SET-Zuweisungen können Sie Unterabfragen formulieren. Für eine Unterabfrage, die einen Wert zurückgibt, der einer Spalte zugewiesen werden soll, kommt ausschließlich eine skalare Unterabfrage infrage. Darüber hinaus können Sie mit einer Unterabfrage keine Daten aus der zu ladenden Tabelle auswählen.

Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket

Sie können die Anweisung `SELECT INTO OUTFILE S3` verwenden, um Daten aus einem DB-Cluster von Amazon Aurora MySQL abzufragen und sie in Textdateien in einem Amazon-S3-Bucket zu speichern. In Aurora MySQL werden die Dateien zuerst auf der lokalen Festplatte gespeichert und dann in S3 exportiert. Nachdem die Exporte abgeschlossen sind, werden die lokalen Dateien gelöscht.

Sie können den Amazon-S3-Bucket mit einem von Amazon S3 verwalteten Schlüssel (SSE-S3) oder einem AWS KMS key (SSE-KMS: Von AWS verwalteter Schlüssel oder vom Kunden verwalteter Schlüssel) verschlüsseln.

Die Anweisung `LOAD DATA FROM S3` kann die von der Anweisung `SELECT INTO OUTFILE S3` erstellten Dateien verwenden, um Daten in einen Aurora-DB-Cluster zu laden. Weitere Informationen finden Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket.

> **Note**
>
> Diese Funktion wird für DB-Cluster von Aurora Serverless v1 nicht unterstützt. Sie wird für DB-Cluster von Aurora Serverless v2 unterstützt.
> Sie können DB-Cluster-Daten und DB-Cluster-Snapshot-Daten auch in Amazon S3 speichern, indem Sie die AWS-Managementkonsole, AWS CLI oder Amazon-RDS-API verwenden. Weitere Informationen erhalten Sie unter Exportieren von DB-Cluster-Daten nach Amazon S3 und Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3.

Inhalt

- Gewähren von Zugriff auf Amazon S3 für Aurora MySQL
- Gewähren von Rechten für das Speichern von Daten in Aurora MySQL
- Angeben eines Pfades zu einem Amazon S3-Bucket
- Erstellen eines Manifests für das Auflisten von Dateien
- SELECT INTO OUTFILE S3
  - Syntax
  - Parameter
  - Überlegungen
  - Beispiele

Gewähren von Zugriff auf Amazon S3 für Aurora MySQL

Bevor Sie Daten in einem Amazon S3-Bucket speichern können, müssen Sie Ihrem Aurora MySQL-DB-Cluster die Berechtigung für den Zugriff auf Amazon S3 erteilen.

So gewähren Sie Aurora MySQL Zugriff auf Amazon S3:

1. Erstellen Sie eine AWS Identity and Access Management-(IAM)-Richtlinie, die die Bucket- und Objektberechtigungen bereitstellt, die Ihrem Aurora-MySQL-DB-Cluster gestatten, auf Amazon S3 zuzugreifen. Detaillierte Anweisungen finden Sie unter Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen.

> **Note**
>
> In Aurora MySQL Version 3.05 und höher können Sie Objekte mit vom Kunden verwalteten AWS KMS-Schlüsseln verschlüsseln. Nehmen Sie dazu die Berechtigung `kms:GenerateDataKey` in Ihre IAM-Richtlinie auf. Weitere Informationen finden Sie unter Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS KMS.
> Zum Verschlüsseln von Objekten mit Von AWS verwaltete Schlüssel oder von Amazon S3 verwalteten Schlüsseln (SSE-S3) benötigen Sie diese Berechtigung nicht.

2. Erstellen Sie eine IAM-Rolle und fügen Sie der neuen IAM-Rolle die IAM-Richtlinie hinzu, die Sie in Erstellen einer IAM-Zugriffsrichtlinie für Amazon S3-Ressourcen erstellt haben. Detaillierte Anweisungen finden Sie unter Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.
3. Legen Sie in Aurora MySQL Version 2 einen der DB-Cluster-Parameter `aurora_select_into_s3_role` oder `aws_default_s3_role` auf den Amazon-Ressourcennamen (ARN) der neuen IAM-Rolle fest. Wenn für `aurora_select_into_s3_role` keine IAM-Rolle angegeben wird, verwendet Aurora die unter `aws_default_s3_role` angegebene IAM-Rolle.

   Verwenden Sie `aws_default_s3_role` in Aurora MySQL Version 3.

   Wenn das Cluster Teil einer globalen Aurora-Datenbank ist, legen Sie diesen Parameter für jedes Aurora-Cluster in der globalen Datenbank fest.

   Weitere Informationen zu DB-Cluster-Parametern finden Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter.
4. Weisen Sie die Rolle, die Sie in Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben erstellt haben, diesem DB-Cluster zu, um Datenbankbenutzern in einem Aurora MySQL DB-Cluster den Zugriff auf Amazon S3 zu erlauben.

   Bei einer globalen Aurora-Datenbank verknüpfen Sie die Rolle mit jedem Aurora-Cluster in der globalen Datenbank.

   Weitere Informationen zum Zuordnen einer IAM-Rolle zu einem DB-Cluster finden Sie unter Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster.

5. Konfigurieren Sie Ihren Aurora MySQL-DB-Cluster so, dass er ausgehende Verbindungen zu Amazon S3 zulässt. Detaillierte Anweisungen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

   Bei einer globalen Aurora-Datenbank aktivieren Sie ausgehende Verbindungen für jeden Aurora-Cluster in der globalen Datenbank.

Gewähren von Rechten für das Speichern von Daten in Aurora MySQL

Der Datenbankbenutzer, der die `SELECT INTO OUTFILE S3`-Anweisung ausgibt, muss über eine bestimmte Rolle oder Berechtigung verfügen. In Aurora MySQL Version 3 gewähren Sie die Rolle `AWS_SELECT_S3_ACCESS`. In Aurora MySQL Version 2 gewähren Sie die Berechtigung `SELECT INTO S3`. Dem Administratorbenutzer für ein DB-Cluster wird die entsprechende Rolle oder -Berechtigung standardmäßig gewährt. Sie können die Berechtigung einem anderen Benutzer erteilen, indem Sie eine der folgenden Anweisungen verwenden.

Verwenden Sie die folgende Anweisung für Aurora MySQL Version 3:

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Wenn Sie die Rollentechnik in Aurora MySQL Version 3 verwenden, können Sie die Rolle auch mit der Anweisung `SET ROLE role_name` oder `SET ROLE ALL` aktivieren. Wenn Sie mit dem MySQL 8.0-Rollensystem nicht vertraut sind, finden Sie in Rollenbasiertes Berechtigungsmodell weitere Informationen. Weitere Informationen finden Sie unter Verwenden von Rollen im MySQL-Referenzhandbuch.
> Dies gilt nur für die aktuelle aktive Sitzung. Wenn Sie sich wieder verbinden, müssen Sie die `SET ROLE`-Anweisung noch einmal ausführen, um Berechtigungen zu gewähren. Weitere Informationen finden Sie unter SET ROLE-Anweisung im MySQL-Referenzhandbuch.
> Sie können den DB-Cluster-Parameter `activate_all_roles_on_login` zum automatischen Aktivieren aller Rollen verwenden, wenn ein Benutzer eine Verbindung mit einer DB-Instance herstellt. Wenn dieser Parameter festgelegt ist, müssen Sie die `SET ROLE`-Anweisung in der Regel nicht explizit aufrufen, um eine Rolle zu aktivieren. Weitere Informationen finden Sie unter activate_all_roles_on_login im MySQL-Referenzhandbuch.

Sie müssen jedoch `SET ROLE ALL` zu Beginn einer gespeicherten Prozedur explizit aufrufen, um die Rolle zu aktivieren, wenn die gespeicherte Prozedur von einem anderen Benutzer aufgerufen wird.

Verwenden Sie die folgende Anweisung für Aurora MySQL Version 2:

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

Die Rolle `AWS_SELECT_S3_ACCESS` und Berechtigung `SELECT INTO S3` gelten speziell für Amazon Aurora MySQL und sind nicht für MySQL-Datenbanken oder RDS für MySQL-DB-Instances verfügbar. Wenn Sie die Replikation zwischen einem DB-Cluster von Aurora MySQL als Replikationsquelle und einer MySQL-Datenbank als Replikations-Client eingerichtet haben, führt die Anweisung `GRANT` für die Rolle oder Berechtigung dazu, dass die Replikation mit einem Fehler beendet wird. Sie können diese Fehlermeldung ignorieren und mit der Replikation fortfahren. Verwenden Sie das Verfahren mysql_rds_skip_repl_error, um die Fehlermeldung für eine RDS für MySQL-DB-Instance zu überspringen. Um den Fehler in einer externen MySQL-Datenbank zu überspringen, verwenden Sie die Systemvariable slave_skip_errors (Aurora MySQL Version 2) oder die Systemvariable replica_skip_errors (Aurora MySQL Version 3).

Angeben eines Pfades zu einem Amazon S3-Bucket

Die Syntax für das Bestimmen eines Pfades zu einem Speicherort für Daten und Manifestdateien in einem Amazon S3-Bucket ist ähnlich der in der Anweisung `LOAD DATA FROM S3 PREFIX`, wie im Folgenden zu sehen ist.

```
s3-region://bucket-name/file-prefix
```

Der Pfad enthält die folgenden Werte:

- `region` (optional) – Die AWS-Region mit dem Amazon-S3-Bucket, in dem die Daten gespeichert werden sollen. Dieser Wert ist optional. Wenn Sie keinen `region`-Wert festlegen, speichert Aurora Ihre Datei in Amazon S3, das sich in derselben Region befindet wie Ihr DB-Cluster.
- `bucket-name` – Der Name des Amazon S3-Buckets, in dem die Daten gespeichert werden sollen. Objekt-Präfixe, die einen virtuellen Ordnerpfad identifizieren, werden unterstützt.
- `file-prefix` – Das Amazon S3-Objektpräfix, das die Dateien identifiziert, die in Amazon S3 gespeichert werden sollen.

Die Dateien, die durch die Anweisung `SELECT INTO OUTFILE S3` erstellt wurden, verwenden den folgenden Pfad, in welchem *`00000`* eine fünfstellige, nullbasierte Ganzzahl ist.

```
s3-region://bucket-name/file-prefix.part_00000
```

Nehmen wir beispielsweise an, dass ein Statement `SELECT INTO OUTFILE S3` `s3-us-west-2://bucket/prefix` als den Pfad festlegt, in dem Dateien gespeichert und drei Datendateien erstellt werden sollen. Das festgelegte Amazon S3-Bucket beinhaltet die folgenden Dateien.

- s3-us-west-2://bucket/prefix.part 00000
- s3-us-west-2://bucket/prefix.part 00001
- s3-us-west-2://bucket/prefix.part 00002

Erstellen eines Manifests für das Auflisten von Dateien

Sie können das Statement `SELECT INTO OUTFILE S3` mit der Option `MANIFEST ON` verwenden, um eine Manifestdatei im JSON-Format zu erstellen, die die mithilfe des Statements erstellten Textdateien auflistet. Die Anweisung `LOAD DATA FROM S3` kann die Manifestdatei verwenden, um Dateien zurück in ein Aurora MySQL-DB-Cluster zu laden. Weitere Informationen über die Verwendung eines Manifests für das Laden von Dateien aus Amazon S3 in einen Aurora MySQL-DB-Cluster finden Sie unter Verwenden eines Manifests für zu ladende Dateien.

Die im Manifest beinhalteten Dateien, die mithilfe des Statements `SELECT INTO OUTFILE S3` erstellt wurden, werden in der Reihenfolge gelistet, in der sie mit dem Statement erstellt wurden. Nehmen wir beispielsweise an, dass ein Statement `SELECT INTO OUTFILE S3` `s3-us-west-2://bucket/prefix` als den Pfad angegeben hat, in dem Dateien gespeichert und drei Dateien und eine Manifestdatei erstellt werden sollen. Der angegebene Amazon S3-Bucket beinhaltet eine Manifestdatei mit dem Namen `s3-us-west-2://bucket/prefix.manifest`, die die folgenden Informationen beinhaltet.

```
{
  "entries": [
    {
      "url":"s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00001"
```

```
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

Sie können die Anweisung `SELECT INTO OUTFILE S3` verwenden, um Daten aus einem DB-Cluster abzufragen und diese direkt in separierten Textdateien in einem Amazon S3-Bucket zu speichern.

Komprimierte Dateien werden nicht unterstützt. Verschlüsselte Dateien werden ab Aurora MySQL Version 2.09.0 unterstützt.

Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
        [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
    [export_options]
    [MANIFEST {ON | OFF}]
    [OVERWRITE {ON | OFF}]
    [ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['cmk_id']}]

export_options:
    [FORMAT {CSV|TEXT} [HEADER]]
```

```
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
]
```

Parameter

Die Anweisung `SELECT INTO OUTFILE S3` verwendet die folgenden erforderlichen und optionalen Parameter, die für Aurora spezifisch sind.

s3-uri

Gibt den URI für ein zu verwendendes Amazon-S3-Präfix an. Verwenden Sie die unter Angeben eines Pfades zu einem Amazon S3-Bucket beschriebene Syntax.

FORMAT {CSV|TEXT} [HEADER]

Speichert die Daten optional im CSV-Format.

Die Option `TEXT` ist die Standardeinstellung und erzeugt das vorhandene MySQL-Exportformat.

Die Option `CSV` erzeugt kommagetrennte Datenwerte. Das CSV-Format entspricht der Spezifikation in RFC-4180. Wenn Sie das optionale Schlüsselwort `HEADER` angeben, enthält die Ausgabedatei eine Kopfzeile. Die Beschriftungen in der Kopfzeile entsprechen den Spaltennamen aus der `SELECT`-Anweisung. Sie können die CSV-Dateien für Trainingsdatenmodelle zur Verwendung mit AWS ML-Services verwenden. Weitere Hinweise zur Verwendung von exportierten Aurora-Daten mit AWS-ML-Services finden Sie unter Export von Daten nach Amazon S3 für das SageMaker AI-Modelltraining (erweitert).

MANIFEST {ON | OFF}

Gibt an, ob eine Manifestdatei in Amazon S3 erstellt wird. Die Manifestdatei ist eine JavaScript-Objekt-Notation (JSON)-Datei, die verwendet werden kann, um Daten in einen Aurora-DB-Cluster mithilfe der Anweisung `LOAD DATA FROM S3 MANIFEST` zu laden. Mehr über `LOAD DATA FROM S3 MANIFEST` erfahren Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket.

Wenn `MANIFEST ON` in der Abfrage angegeben ist, wird die Manifestdatei in Amazon S3 erstellt, nachdem alle Dateien erstellt und hochgeladen wurden. Die Manifestdatei wird unter Verwendung des folgenden Pfads erstellt:

```
s3-region://bucket-name/file-prefix.manifest
```

Weitere Informationen über das Format der Inhalte der Manifestdatei finden Sie unter Erstellen eines Manifests für das Auflisten von Dateien.

OVERWRITE {ON | OFF}

Gibt an, ob bestehende Dateien im angegebenen Amazon-S3-Bucket überschrieben werden. Wenn `OVERWRITE ON` festgelegt ist, werden bestehende Dateien, die mit dem Präfix des im `s3-uri`-URI angegebenen Präfixes übereinstimmen, überschrieben. Andernfalls tritt ein Fehler auf.

ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['*cmk_id*']}

Gibt an, ob eine serverseitige Verschlüsselung mit von Amazon S3 verwalteten Schlüsseln (SSE-S3) oder AWS KMS keys (SSE-KMS, darunter auch Von AWS verwaltete Schlüssel und vom Kunden verwaltete Schlüssel) verwendet werden soll. Die Einstellungen `SSE_S3` und `SSE_KMS` sind in Aurora MySQL Version 3.05 und höher verfügbar.

Sie können auch die Sitzungsvariable `aurora_select_into_s3_encryption_default` anstelle der Klausel `ENCRYPTION` verwenden, wie im folgenden Beispiel gezeigt. Verwenden Sie entweder die SQL-Klausel oder die Sitzungsvariable, aber nicht beide.

```
set session set session aurora_select_into_s3_encryption_default={ON | OFF | SSE_S3
 | SSE_KMS};
```

Die Einstellungen `SSE_S3` und `SSE_KMS` sind in Aurora MySQL Version 3.05 und höher verfügbar.

Wenn Sie `aurora_select_into_s3_encryption_default` auf folgende Werte einstellen, gilt:

- `OFF`: Es wird die Standardverschlüsselungsrichtlinie des S3-Buckets befolgt. Der Standardwert von `aurora_select_into_s3_encryption_default` ist `OFF`.
- `ON` oder `SSE_S3`: Das S3-Objekt wird mit von Amazon S3 verwalteten Schlüsseln (SSE-S3) verschlüsselt.
- `SSE_KMS`: Das S3-Objekt wird mit einem AWS KMS key verschlüsselt.

In diesem Fall fügen Sie auch die Sitzungsvariable `aurora_s3_default_cmk_id` hinzu, beispielsweise:

```
set session aurora_select_into_s3_encryption_default={SSE_KMS};
set session aurora_s3_default_cmk_id={NULL | 'cmk_id'};
```

- Wenn `aurora_s3_default_cmk_id` `NULL` ist, wird das S3-Objekt mit einem Von AWS verwalteter Schlüssel verschlüsselt.
- Wenn es sich bei `aurora_s3_default_cmk_id` um eine nicht leere Zeichenfolge `cmk_id` handelt, wird das S3-Objekt mit einem vom Kunden verwalteten Schlüssel verschlüsselt.

  Der Wert von `cmk_id` kann keine leere Zeichenfolge sein.

Wenn Sie den Befehl `SELECT INTO OUTFILE S3` verwenden, bestimmt Aurora die Verschlüsselung wie folgt:

- Wenn die Klausel `ENCRYPTION` im SQL-Befehl vorhanden ist, stützt sich Aurora nur auf den Wert für `ENCRYPTION` und verwendet keine Sitzungsvariable.
- Wenn die Klausel `ENCRYPTION` nicht vorhanden ist, stützt sich Aurora auf den Wert der Sitzungsvariablen.

Weitere Informationen finden Sie unter Verwenden serverseitiger Verschlüsselung mit von Amazon S3 verwalteten Schlüsseln (SSE-S3) und Verwenden serverseitiger Verschlüsselung mit AWS KMS-Schlüsseln (SSE-KMS) im Benutzerhandbuch für Amazon Simple Storage Service.

Weitere Details zu anderen Parametern finden Sie unter SELECT-Anweisung und LOAD DATA-Anweisung in der MySQL-Dokumentation.

Überlegungen

Die Anzahl der im Amazon S3-Bucket geschriebenen Dateien hängt von der Datenmenge, die von der Anweisung `SELECT INTO OUTFILE S3` ausgewählt wurde, und dem Schwellenwert der Dateigröße für Aurora MySQL ab. Der Standard-Schwellenwert für die Dateigröße liegt bei 6 GB. Wenn die vom Statement ausgewählte Datei kleiner ist als der Schwellenwert für die Dateigröße, wird eine einzelne Datei erstellt, andernfalls werden mehrere Dateien erstellt. Außerdem sollte Folgendes für Dateien, die von diesem Statement erstellt wurden, beachtet werden:

- Aurora MySQL garantiert, dass Zeilen in Dateien nicht über Dateigrenzen aufgeteilt werden. Bei mehreren Dateien liegt die Dateigröße jeder Datei, außer der Letzten, normalerweise sehr

dicht am Schwellenwert. Jedoch kann es gelegentlich vorkommen, dass die Zeilen von Dateien, deren Größe unter dem Schwellenwert liegt, zwischen zwei Dateien aufgeteilt werden. In diesem Fall erstellt Aurora MySQL eine Datei, die die Zeile intakt hält, aber dadurch größer als der Schwellenwert ist.

- Da jede `SELECT`-Anweisung in Aurora MySQL als eine atomare Transaktion ausgeführt wird, kann eine `SELECT INTO OUTFILE S3`-Anweisung, die einen großen Datensatz auswählt, einige Zeit in Anspruch nehmen. Wenn die Anweisung aus irgendeinem Grund fehlschlägt, müssen Sie möglicherweise erneut starten und die Anweisung nochmals ausstellen. Wenn die Anweisung fehlschlägt, bleiben Dateien, die bereits in Amazon S3 hochgeladen wurden, im angegebenen Amazon S3-Bucket bestehen. Sie können ein anderes Statement verwenden, um die übrigen Daten hochzuladen, anstatt von Neuem zu starten.
- Wenn die auszuwählende Datenmenge groß ist (mehr als 25 GB), empfehlen wir Ihnen, mehrere `SELECT INTO OUTFILE S3`-Anweisungen zu verwenden, um die Daten in Amazon S3 zu speichern. Jedes Statement sollte eine andere Partition für die zu speichernden Daten auswählen und auch ein anderes `file_prefix` im `s3-uri`-Parameter angeben, wenn die Daten gespeichert werden. Die Partitionierung der zu selektierenden Daten mit mehreren Anweisungen erleichtert die Behebung eines Fehlers in einer Anweisung. Wenn bei einer Aussage ein Fehler auftritt, muss nur ein Teil der Daten neu ausgewählt und in Amazon S3 hochgeladen werden. Die Verwendung von mehreren Statements hilft auch eine lange dauernde Einzeltransaktion zu vermeiden, was die Leistung verbessern kann.
- Wenn mehrere `SELECT INTO OUTFILE S3`-Anweisungen, die das selbe `file_prefix` im `s3-uri`-Parameter verwenden, parallel ausgeführt werden, um Daten nach Amazon S3 auszuwählen, kann das Verhalten nicht eingeschätzt werden.
- Metadaten, wie z. B. Tabellenschemata oder Metadatendateien werden nicht von Aurora MySQL nach Amazon S3 hochgeladen.
- In einigen Fällen können Sie eine `SELECT INTO OUTFILE S3`-Abfrage erneut tätigen, um beispielsweise nach einem Fehler wiederherzustellen. In diesen Fällen müssen Sie entweder alle bestehenden Dateien mit dem selben Präfix, wie in `s3-uri` angegeben, aus dem Amazon S3-Bucket entfernen oder `OVERWRITE ON` in die `SELECT INTO OUTFILE S3`-Abfrage einbinden.

Das Statement `SELECT INTO OUTFILE S3` gibt eine typische MySQL-Fehlernummer und -Antwort bei Erfolg oder Fehler zurück. Wenn Sie keinen Zugang zur MySQL-Fehlernummer oder -Antwort haben, ist es der einfachste Weg, nach Abschluss `MANIFEST ON` im Statement anzugeben. Die Manifestdatei ist die letzte Datei, die vom Statement geschrieben wurde. Mit anderen Worten, wenn Sie über eine Manifestdatei verfügen, hat die Anweisung die Ausführung abgeschlossen.

Aktuell gibt es keine Möglichkeit, den Fortschritt der `SELECT INTO OUTFILE S3`-Anweisung bei der Ausführung direkt zu überwachen. Nehmen wir jedoch an, Sie schreiben mithilfe dieser Anweisung eine große Datenmenge aus Aurora MySQL zu Amazon S3 und Sie kennen die Größe der von der Anweisung ausgewählten Daten. In diesem Fall können Sie den Fortschritt schätzen, indem Sie die Erstellung der Dateien in Amazon S3 überwachen.

Dabei können Sie sich die Tatsache zunutze machen, dass eine Datei im festgelegten Amazon S3-Bucket für je 6 GB Daten, die vom Statement ausgewählt wurden, erstellt wird. Teilen Sie diese ausgewählte Datengröße durch 6 GB, um eine geschätzte Anzahl der zu erstellenden Dateien zu erhalten. So können Sie den Fortschritt der Anweisung einschätzen, wenn Sie die Anzahl der in Amazon S3 hochgeladenen Dateien während der Ausführung der Anweisung überwachen.

Beispiele

Die folgende Anweisung wählt alle Daten in der Tabelle `employees` aus und speichert die Daten in einem Amazon S3-Bucket, der sich in einer anderen Region als der Aurora MySQL-DB-Cluster befindet. Das Statement erstellt Dateien, bei denen jedes Feld durch ein Komma (`,`) und jede Zeile durch einen Zeilenumbruch (`\n`) beendet wird. Die Anweisung gibt einen Fehler zurück, wenn sich Dateien, die mit dem Dateipräfix `sample_employee_data` übereinstimmen, im angegebenen Amazon S3-Bucket befinden.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

Die folgende Anweisung wählt alle Daten in der Tabelle `employees` aus und speichert die Daten in einem Amazon S3-Bucket, der sich in der selben Region, wie der Aurora MySQL-DB-Cluster befindet. Das Statement erstellt Dateien bei denen jedes Feld durch ein Komma (`,`) und jede Zeile durch einen Zeilenumbruch (`\n`) beendet wird. Zudem wird eine Manifestdatei erstellt. Die Anweisung gibt einen Fehler zurück, wenn sich Dateien, die mit dem Dateipräfix `sample_employee_data` übereinstimmen, im angegebenen Amazon S3-Bucket befinden.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON;
```

Die folgende Anweisung wählt alle Daten in der Tabelle `employees` aus und speichert die Daten in einem Amazon S3-Bucket, der sich in einer anderen Region als der Aurora-DB-Cluster befindet. Das Statement erstellt Dateien, bei denen jedes Feld durch ein Komma (`,`) und jede Zeile durch einen Zeilenumbruch (`\n`) beendet wird. Die Anweisung überschreibt alle bestehenden Dateien, die mit dem Dateipräfix `sample_employee_data` im angegebenen Amazon S3-Bucket übereinstimmen.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    OVERWRITE ON;
```

Die folgende Anweisung wählt alle Daten in der Tabelle `employees` aus und speichert die Daten in einem Amazon S3-Bucket, der sich in der selben Region, wie der Aurora MySQL-DB-Cluster befindet. Das Statement erstellt Dateien bei denen jedes Feld durch ein Komma (`,`) und jede Zeile durch einen Zeilenumbruch (`\n`) beendet wird. Zudem wird eine Manifestdatei erstellt. Die Anweisung überschreibt alle bestehenden Dateien, die mit dem Dateipräfix `sample_employee_data` im angegebenen Amazon S3-Bucket übereinstimmen.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON
    OVERWRITE ON;
```

Aufrufen einer Lambda-Funktion aus einem Amazon Aurora MySQL-DB-Cluster

Sie können eine AWS Lambda-Funktion von einem Amazon Aurora MySQL-kompatiblen Edition-DB-Cluster mit der nativen Funktion `lambda_sync` oder `lambda_async` aufrufen. Damit Sie eine Lambda-Funktion von einem Aurora MySQL aufrufen können, benötigt der Aurora-DB-Cluster Zugriff auf Lambda. Details zum Gewähren des Zugriffs auf Aurora MySQL finden Sie unter Gewähren von Zugriff auf Lambda für Aurora. Informationen zu den gespeicherten `lambda_sync`- und `lambda_async`-Funktionen finden Sie unter Aufrufen einer Lambda-Funktion mit einer nativen Aurora MySQL-Funktion.

Sie können eine AWS Lambda-Funktion auch mithilfe einer gespeicherten Prozedur aufrufen. Die Verwendung einer gespeicherten Prozedur ist jedoch veraltet. Wir empfehlen nachdrücklich die

Verwendung einer Aurora MySQL-nativen Funktion, wenn Sie eine der folgenden Aurora MySQL-Versionen nutzen:

- Aurora MySQL Version 2 für MySQL 5.7-kompatible Cluster.
- Aurora MySQL Version 3.01 und höher für MySQL 8.0-kompatible Cluster. Die gespeicherte Prozedur ist in Aurora MySQL Version 3 nicht verfügbar.

Weitere Informationen dazu, wie Sie Aurora Zugriff auf Lambda gewähren und eine Lambda-Funktion aufrufen, finden Sie in den folgenden Themen.

Themen

- Gewähren von Zugriff auf Lambda für Aurora
- Aufrufen einer Lambda-Funktion mit einer nativen Aurora MySQL-Funktion
- Aufrufen einer Lambda-Funktion mit einer Aurora MySQL-Stored Procedure (veraltet)

Gewähren von Zugriff auf Lambda für Aurora

Bevor Sie Lambda-Funktionen von einem Aurora MySQL DB-Cluster aufrufen können, stellen Sie sicher, dass Sie Ihrem Cluster zuerst die Berechtigung für den Zugriff auf Lambda erteilen.

So gewähren Sie Aurora MySQL Zugriff auf Lambda:

1. Erstellen Sie eine AWS Identity and Access Management-(IAM)-Richtlinie, die die Berechtigungen bereitstellt, die Ihrem Aurora-MySQL-DB-Cluster gestatten, Lambda-Funktionen aufzurufen. Detaillierte Anweisungen finden Sie unter Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS Lambda.
2. Erstellen Sie eine IAM-Rolle und fügen Sie der neuen IAM-Rolle die IAM-Richtlinie hinzu, die Sie in Erstellen einer IAM-Richtlinie für den Zugriff auf Ressourcen AWS Lambda erstellt haben. Detaillierte Anweisungen finden Sie unter Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben.
3. Sie müssen für den Amazon-Ressourcennamen (ARN) der neuen IAM-Rolle den DB-Cluster-Parameter `aws_default_lambda_role` festlegen.

   Wenn das Cluster Teil einer globalen Aurora-Datenbank ist, wenden Sie dieselbe Einstellung auf jeden Aurora-Cluster in der globalen Datenbank an.

Weitere Informationen zu DB-Cluster-Parametern finden Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter.

4. Weisen Sie die Rolle, die Sie in Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben erstellt haben, diesem DB-Cluster zu, um Datenbankbenutzern in einem Aurora MySQL DB-Cluster den Aufruf von Lambda-Funktionen zu erlauben. Weitere Informationen zum Zuordnen einer IAM-Rolle zu einem DB-Cluster finden Sie unter Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster.

   Wenn das Cluster Teil einer globalen Aurora-Datenbank ist, verknüpfen Sie die Rolle mit jedem Aurora-Cluster in der globalen Datenbank.

5. Konfigurieren Sie Ihren Aurora MySQL-DB-Cluster so, dass er ausgehende Verbindungen zu Lambda zulässt. Detaillierte Anweisungen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

   Wenn das Cluster Teil einer globalen Aurora-Datenbank ist, aktivieren Sie ausgehende Verbindungen für jeden Aurora-Cluster in der globalen Datenbank.

Aufrufen einer Lambda-Funktion mit einer nativen Aurora MySQL-Funktion

> **Note**
>
> Sie können die nativen Funktionen `lambda_sync` und `lambda_async` aufrufen, wenn Sie Aurora MySQL Version 2 oder Aurora MySQL Version 3.01 und höher verwenden. Weitere Informationen zu den Aurora MySQL-Versionen erhalten Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.

Sie können eine AWS Lambda-Funktion in einem Aurora-MySQL-DB-Cluster durch den Aufruf der nativen Funktionen `lambda_sync` und `lambda_async` aufrufen. Dieser Ansatz kann hilfreich sein, wenn Sie Ihre Datenbank in Aurora MySQL in andere AWS-Services integrieren möchten. Beispielsweise soll jedes Mal, wenn eine Zeile in eine bestimmte Tabelle der Datenbank eingefügt wird, eine Benachrichtigung mit Amazon Simple Notification Service (Amazon SNS) gesendet werden.

Inhalt

- Arbeiten mit nativen Funktionen zum Aufrufen einer Lambda-Funktion

- Gewähren der Rolle in Aurora MySQL Version 3
- Gewähren der Berechtigung in Aurora MySQL Version 2
- Syntax für die lambda_sync-Funktion
- Parameter für die lambda_sync-Funktion
- Beispiel für die lambda_sync-Funktion
- Syntax für die lambda_async-Funktion
- Parameter für die lambda_async-Funktion
- Beispiel für die lambda_async-Funktion
- Aufrufen einer Lambda-Funktion innerhalb eines Auslösers

Arbeiten mit nativen Funktionen zum Aufrufen einer Lambda-Funktion

Die Funktionen `lambda_sync` und `lambda_async` sind eingebaute, native Funktionen, die eine Lambda-Funktion synchron oder asynchron aufrufen. Wenn Sie das Ergebnis der Ausführung der aufgerufenen Lambda-Funktion kennen müssen, bevor Sie zu einer anderen Aktion übergehen, verwenden Sie die synchrone Funktion `lambda_sync`. Wenn Sie das Ergebnis der Ausführung der aufgerufenen Lambda-Funktion nicht kennen müssen, bevor Sie zu einer anderen Aktion übergehen, verwenden Sie die asynchrone Funktion `lambda_async`.

Gewähren der Rolle in Aurora MySQL Version 3

In Aurora MySQL Version 3 muss dem Benutzer, der eine native Funktion aufruft, die `AWS_LAMBDA_ACCESS`-Rolle gewährt werden. Um einem Benutzer diese Rolle zuzuweisen, stellen Sie als Administrator eine Verbindung zur DB-Instance her und führen Sie die folgende Anweisung aus.

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

Sie können diese Rolle widerrufen, indem Sie die folgende Anweisung ausführen.

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

> **Tip**
>
> Wenn Sie die Rollentechnik in Aurora MySQL Version 3 verwenden, können Sie die Rolle auch mit der Anweisung `SET ROLE role_name` oder `SET ROLE ALL` aktivieren. Wenn

Sie mit dem MySQL 8.0-Rollensystem nicht vertraut sind, finden Sie in Rollenbasiertes Berechtigungsmodell weitere Informationen. Weitere Informationen finden Sie unter Verwenden von Rollen im MySQL-Referenzhandbuch.
Dies gilt nur für die aktuelle aktive Sitzung. Wenn Sie sich wieder verbinden, müssen Sie die `SET ROLE`-Anweisung noch einmal ausführen, um Berechtigungen zu gewähren. Weitere Informationen finden Sie unter SET ROLE-Anweisung im MySQL-Referenzhandbuch.
Sie können den DB-Cluster-Parameter `activate_all_roles_on_login` zum automatischen Aktivieren aller Rollen verwenden, wenn ein Benutzer eine Verbindung mit einer DB-Instance herstellt. Wenn dieser Parameter festgelegt ist, müssen Sie die `SET ROLE`-Anweisung in der Regel nicht explizit aufrufen, um eine Rolle zu aktivieren. Weitere Informationen finden Sie unter activate_all_roles_on_login im MySQL-Referenzhandbuch.
Sie müssen jedoch `SET ROLE ALL` zu Beginn einer gespeicherten Prozedur explizit aufrufen, um die Rolle zu aktivieren, wenn die gespeicherte Prozedur von einem anderen Benutzer aufgerufen wird.

Wenn Sie beim Versuch, eine Lambda-Funktion aufzurufen, eine Fehlermeldung wie die folgende erhalten, führen Sie eine `SET ROLE`-Anweisung aus.

```
SQL Error [1227] [42000]: Access denied; you need (at least one of) the Invoke Lambda
 privilege(s) for this operation
```

Stellen Sie sicher, dass Sie die Rolle dem richtigen Benutzer erteilen, wie in den `mysql.users`-Tabelleneinträgen gezeigt. Möglicherweise gibt es mehrere Benutzer mit demselben Namen, aber auf unterschiedlichen Hosts. Je nachdem, welche Anwendung oder welcher Host die Funktion `lambda_sync` aufruft, wählt MySQL anhand der `host`-Spalteneinträge den Benutzer mit der besten Übereinstimmung aus.

Gewähren der Berechtigung in Aurora MySQL Version 2

In Aurora MySQL Version 2 muss dem Benutzer, der eine native Funktion aufruft, die `INVOKE LAMBDA`-Berechtigung gewährt werden. Um einem Benutzer diese Berechtigung zu erteilen, stellen Sie als Administrator eine Verbindung zur DB-Instance her und führen Sie die folgende Anweisung aus.

```
GRANT INVOKE LAMBDA ON *.* TO user@domain-or-ip-address
```

Sie können das Recht für einen anderen Benutzer mithilfe der folgenden Anweisung erteilen.

```
REVOKE INVOKE LAMBDA ON *.* FROM user@domain-or-ip-address
```

Syntax für die lambda_sync-Funktion

Sie rufen die Funktion `lambda_sync` synchron zum Aufruftyp `RequestResponse` auf. Die Funktion gibt das Ergebnis des Aufrufs von Lambda in einer JSON-Nutzlast zurück. Die Funktion weist die folgende Syntax auf.

```
lambda_sync (
  lambda_function_ARN,
  JSON_payload
)
```

Parameter für die lambda_sync-Funktion

Die Funktion `lambda_sync` hat die folgenden Parameter.

lambda_function_ARN

Der Amazon-Ressourcenname (ARN) der aufzurufenden Lambda-Funktion.

JSON-Nutzlast

Die Nutzlast für die aufgerufene Lambda-Funktion im JSON-Format.

> **Note**
>
> Aurora MySQL Version 3 unterstützt die JSON-Parsingfunktionen von MySQL 8.0. Aurora MySQL-Version 2 enthält diese Funktionen jedoch nicht. JSON-Parsing ist nicht erforderlich, wenn eine Lambda-Funktion einen atomaren Wert zurückgibt, wie z. B. eine Zahl oder einen String.

Beispiel für die lambda_sync-Funktion

Die folgende Abfrage, die auf `lambda_sync` basiert, ruft die Lambda-Funktion `BasicTestLambda` synchron mithilfe der ARN-Funktion auf. Die Nutzlast für die Funktion ist `{"operation": "ping"}`.

```
SELECT lambda_sync(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
```

```
    '{"operation": "ping"}');
```

Syntax für die lambda_async-Funktion

Sie rufen die Funktion `lambda_async` asynchron zum Aufruftyp `Event` auf. Die Funktion gibt das Ergebnis des Aufrufs von Lambda in einer JSON-Nutzlast zurück. Die Funktion weist die folgende Syntax auf.

```
lambda_async (
  lambda_function_ARN,
  JSON_payload
)
```

Parameter für die lambda_async-Funktion

Die Funktion `lambda_async` hat die folgenden Parameter.

lambda_function_ARN

Der Amazon-Ressourcenname (ARN) der aufzurufenden Lambda-Funktion.

JSON-Nutzlast

Die Nutzlast für die aufgerufene Lambda-Funktion im JSON-Format.

> **Note**
>
> Aurora MySQL Version 3 unterstützt die JSON-Parsingfunktionen von MySQL 8.0. Aurora MySQL-Version 2 enthält diese Funktionen jedoch nicht. JSON-Parsing ist nicht erforderlich, wenn eine Lambda-Funktion einen atomaren Wert zurückgibt, wie z. B. eine Zahl oder einen String.

Beispiel für die lambda_async-Funktion

Die folgende Abfrage, die auf `lambda_async` basiert, ruft die Lambda-Funktion `BasicTestLambda` asynchron mithilfe der ARN-Funktion auf. Die Nutzlast für die Funktion ist `{"operation": "ping"}`.

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
```

```
    '{"operation": "ping"}');
```

Aufrufen einer Lambda-Funktion innerhalb eines Auslösers

Sie können Auslöser verwenden, um Lambda für datenmodifizierende Anweisungen aufzurufen. Das folgende Beispiel verwendet die native `lambda_async`-Funktion und speichert das Ergebnis in einer Variablen.

```
mysql>SET @result=0;
mysql>DELIMITER //
mysql>CREATE TRIGGER myFirstTrigger
      AFTER INSERT
          ON Test_trigger FOR EACH ROW
      BEGIN
      SELECT lambda_async(
          'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
          '{"operation": "ping"}')
          INTO @result;
      END; //
mysql>DELIMITER ;
```

> **Note**
>
> Auslöser werden nicht einmal pro SQL-Anweisung ausgeführt, sondern einmal pro geänderter Zeile, jeweils für eine einzelne Zeile. Wenn ein Trigger ausgeführt wird, ist der Prozess synchron. Die datenmodifizierende Anweisung wird nur zurückgegeben, wenn der Trigger abgeschlossen ist.
> Seien Sie vorsichtig, wenn Sie eine AWS Lambda-Funktion aus Auslösern in Tabellen aufrufen, die einen hohen Schreibdatenverkehr aufweisen. Die Auslöser `INSERT`, `UPDATE` und `DELETE` werden pro Zeile aktiviert. Ein Workload mit hohem Schreibdatenverkehr in einer Tabelle mit den Auslösern `INSERT`, `UPDATE` oder `DELETE` führt zu einer hohen Anzahl von Aufrufen Ihrer AWS Lambda-Funktion.

Aufrufen einer Lambda-Funktion mit einer Aurora MySQL-Stored Procedure (veraltet)

Sie können eine AWS Lambda-Funktion in einem Aurora-MySQL-DB-Cluster durch den Aufruf der Prozedur `mysql.lambda_async` aufrufen. Dieser Ansatz kann hilfreich sein, wenn Sie Ihre Datenbank in Aurora MySQL in andere AWS-Services integrieren möchten. Beispielsweise soll jedes

Mal, wenn eine Zeile in eine bestimmte Tabelle der Datenbank eingefügt wird, eine Benachrichtigung mit Amazon Simple Notification Service (Amazon SNS) gesendet werden.

Inhalt

- Überlegungen zu Aurora MySQL-Versionen
- Verwenden der Prozedur „mysql.lambda async“ zum Aufrufen einer Lambda-Funktion (veraltet)
  - Syntax
  - Parameter
  - Beispiele

Überlegungen zu Aurora MySQL-Versionen

Ab Aurora MySQL Version 2 können Sie die native Funktionsmethode anstelle dieser gespeicherten Prozeduren verwenden, um eine Lambda-Funktion aufzurufen. Weitere Informationen zu den nativen Funktionen finden Sie unter Arbeiten mit nativen Funktionen zum Aufrufen einer Lambda-Funktion.

In Aurora MySQL Version 2 wird die gespeicherte Prozedur `mysql.lambda_async` nicht mehr unterstützt. Es wird dringend empfohlen, stattdessen native Lambda-Funktionen zu verwenden.

In Aurora MySQL Version 3 ist die gespeicherte Prozedur nicht verfügbar.

Verwenden der Prozedur „mysql.lambda async“ zum Aufrufen einer Lambda-Funktion (veraltet)

Bei `mysql.lambda_async` handelt es sich um eine integrierte gespeicherte Prozedur, mit der eine Lambda-Funktion asynchron aufgerufen wird. Zur Nutzung dieser Prozedur muss der Datenbankbenutzer über die `EXECUTE`-Berechtigung für die gespeicherte Prozedur `mysql.lambda_async` verfügen.

Syntax

Die Prozedur `mysql.lambda_async` verwendet die folgende Syntax.

```
CALL mysql.lambda_async (
  lambda_function_ARN,
  lambda_function_input
)
```

Parameter

Die Prozedur `mysql.lambda_async` hat die folgenden Parameter.

lambda_function_ARN

Der Amazon-Ressourcenname (ARN) der aufzurufenden Lambda-Funktion.

lambda_function_input

Die Eingabezeichenfolge im JSON-Format für die aufgerufene Lambda-Funktion.

Beispiele

Als bewährte Methode wird empfohlen, Aufrufe der Prozedur `mysql.lambda_async` in einer gespeicherten Prozedur zu verpacken, die aus verschiedenen Quellen aufgerufen werden kann, wie zum Beispiel Auslöser oder Clientcode. Mit dieser Methode werden Probleme mit Impedanzfehlanpassungen vermieden, zudem wird der Aufruf von Lambda-Funktionen vereinfacht.

> **Note**
>
> Seien Sie vorsichtig, wenn Sie eine AWS Lambda-Funktion aus Auslösern in Tabellen aufrufen, die einen hohen Schreibdatenverkehr aufweisen. Die Auslöser `INSERT`, `UPDATE` und `DELETE` werden pro Zeile aktiviert. Ein Workload mit hohem Schreibdatenverkehr in einer Tabelle mit den Auslösern `INSERT`, `UPDATE` oder `DELETE` führt zu einer hohen Anzahl von Aufrufen Ihrer AWS Lambda-Funktion.
> Die Aufrufe der Prozedur `mysql.lambda_async` sind asynchron, Auslöser jedoch synchron. Eine Anweisung, die zu einer hohen Anzahl an Auslöseraktivierungen führt, wartet nicht, bis der AWS Lambda-Funktionsaufruf abgeschlossen ist – sie wartet jedoch, bis die Auslöser abgeschlossen sind, bevor die Kontrolle an den Client zurückgegeben wird.

Example Beispiel: Aufrufen einer AWS Lambda-Funktion zum Senden einer E-Mail

Im folgenden Beispiel wird eine gespeicherte Prozedur erstellt, die Sie in Ihrem Datenbankcode aufrufen können, um eine E-Mail mit einer Lambda-Funktion zu senden.

AWS Lambda-Funktion

```
import boto3

ses = boto3.client('ses')
```

```
def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
            'ToAddresses': [
            event['email_to'],
            ]
        },

        Message={
            'Subject': {
            'Data': event['email_subject']
            },
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

Gespeicherte Prozedur

```
DROP PROCEDURE IF EXISTS SES_send_email;
DELIMITER ;;
  CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
                                  IN email_to VARCHAR(255),
                                  IN subject VARCHAR(255),
                                  IN body TEXT) LANGUAGE SQL
  BEGIN
    CALL mysql.lambda_async(
         'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
         CONCAT('{"email_to" : "', email_to,
             '", "email_from" : "', email_from,
             '", "email_subject" : "', subject,
             '", "email_body" : "', body, '"}')
     );
  END
  ;;
DELIMITER ;
```

Abrufen der gespeicherten Prozedur zum Aufrufen der AWS Lambda-Funktion

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email
 subject', 'Email content');
```

Example Beispiel: Aufrufen einer AWS Lambda-Funktion zum Veröffentlichen eines Ereignisses von einem Auslöser

Im folgenden Beispiel wird eine gespeicherte Prozedur erstellt, die ein Ereignis mithilfe von Amazon SNS veröffentlicht. Der Code ruft die Prozedur von einem Auslöser auf, wenn eine Zeile zu einer Tabelle hinzugefügt wird.

AWS Lambda-Funktion

```
import boto3

sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

Gespeicherte Prozedur

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                      IN message TEXT) LANGUAGE SQL
BEGIN
  CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
     CONCAT('{ "subject" : "', subject,
            '", "message" : "', message, '" }')
     );
END
;;
DELIMITER ;
```

Tabelle

```
CREATE TABLE 'Customer_Feedback' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'customer_name' varchar(255) NOT NULL,
  'customer_feedback' varchar(1024) NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Auslöser

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
  AFTER INSERT ON Customer_Feedback
  FOR EACH ROW
BEGIN
  SELECT CONCAT('New customer feedback from ', NEW.customer_name),
 NEW.customer_feedback INTO @subject, @feedback;
  CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

Einfügen einer Tabellenzeile zum Auslösen der Benachrichtigung

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
 Customer', 'Good job guys!');
```

Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs

Sie können einen Aurora MySQL-DB-Cluster so konfigurieren, dass allgemeine, Slow-, Audit- und Fehlerprotokolldaten in einer Protokollgruppe in Amazon CloudWatch Logs veröffentlicht werden. Mit CloudWatch Logs können Sie Echtzeitanalysen der Protokolldaten durchführen, und mit CloudWatch können Sie Alarme und Metriken erstellen. Sie können CloudWatch Logs verwenden, um Ihre Protokolldatensätze in einem Speicher mit hoher Beständigkeit abzulegen.

Damit Protokolle in CloudWatch Logs veröffentlicht werden können, müssen die betreffenden Protokolle aktiviert sein. Fehlerprotokolle sind standardmäßig aktiviert, Protokolle anderer Typen müssen dagegen explizit aktiviert werden. Informationen zum Aktivieren der Protokolle in MySQL finden Sie unter Auswahl von allgemeinen Abfrage- und langsamen Abfrage-Protokollausgabezielen

in der MySQL-Dokumentation. Weitere Informationen über das Aktivieren von Aurora MySQL-Audit-Protokollen finden Sie unter Aktivieren von erweitertem Auditing.

Note

- Wenn der Export von Protokolldaten deaktiviert ist, löscht Aurora keine existierenden Protokollgruppen oder Protokollstreams. Wenn der Export von Protokolldaten deaktiviert ist, bleiben vorhandene Protokolldaten je nach Protokollaufbewahrungszeit in CloudWatch Logs verfügbar und es werden weiterhin Gebühren für die gespeicherten Audit-Protokolldaten berechnet. Sie können Protokollstreams und Protokollgruppen in der CloudWatch-Logs-Konsole, der AWS CLI oder der CloudWatch-Logs-API löschen.
- Eine alternative Möglichkeit zum Veröffentlichen von Audit-Protokollen in CloudWatch Logs besteht im Aktivieren des erweiterten Auditings, anschließend im Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe und dem Einstellen des Parameters `server_audit_logs_upload` auf `1`. Der Standardwert für diesen `server_audit_logs_upload`-DB-Cluster-Parameter ist `0`. Informationen zum Aktivieren des erweiterten Auditings finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.

  Wenn Sie diese alternative Methode verwenden, brauchen Sie eine IAM-Rolle, um auf CloudWatch Logs zuzugreifen, und müssen den `aws_default_logs_role`-Parameter auf Cluster-Ebene auf den ARN für diese Rolle setzen. Weitere Informationen zum Erstellen der Rolle finden Sie unter Einrichten von IAM-Rollen für den Zugriff auf AWS-Services. Wenn Sie jedoch die serviceverknüpfte Rolle `AWSServiceRoleForRDS` haben, bietet sie Zugriff auf CloudWatch Logs und überschreibt alle benutzerdefinierten Rollen. Informationen zu serviceverknüpften Rollen für Amazon RDS finden Sie unter Verwenden von serviceverknüpften Rollen für Amazon Aurora.
- Wenn Audit-Protokolle nicht in CloudWatch Logs exportiert werden sollen, müssen Sie alle Methoden zum Exportieren von Audit-Protokollen deaktivieren. Bei diesen Methoden handelt es sich um die AWS-Managementkonsole, die AWS CLI, die RDS-API und den Parameter `server_audit_logs_upload`.
- Das Verfahren ist für Aurora Serverless v1-DB-Cluster geringfügig anders als für DB-Cluster mit bereitgestellten oder Aurora Serverless v2-DB-Instances. Aurora Serverless v1-Cluster laden automatisch alle Protokolle hoch, die Sie über Konfigurationsparameter aktivieren.

Daher ermöglichen oder verhindern Sie ein Protokoll-Upload für Aurora Serverless v1-DB-Cluster, indem Sie andere Protokolltypen in der DB-Cluster-Parametergruppe aktivieren oder deaktivieren. Sie können die Einstellungen des Clusters selbst nicht über die AWS-Managementkonsole, AWS CLI oder RDS-API bearbeiten. Informationen zum Aktivieren und Deaktivieren von MySQL-Protokollen für Aurora Serverless v1-Cluster finden Sie unterParametergruppen für Aurora Serverless v1.

Konsole

Sie können Aurora MySQL-Protokolle für bereitgestellte Cluster mithilfe der Konsole in CloudWatch Logs veröffentlichen.

Veröffentlichen Sie Aurora MySQL-Protokolle mit der Konsole wie folgt:

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Aurora MySQL-DB-Cluster, für den die Protokolldaten veröffentlicht werden sollen.
4. Wählen Sie Ändern aus.
5. Wählen Sie im Bereich Protokollexporte die Protokolle aus, die Sie in CloudWatch Logs veröffentlichen möchten.
6. Wählen Sie Weiter und dann auf der Übersichtsseite DB-Cluster ändern.

AWS CLI

Sie können Aurora-MySQL-Protokolle für bereitgestellte Cluster mithilfe der AWS-CLI veröffentlichen. Hierzu führen Sie den AWS CLI-Befehl modify-db-cluster mit den folgenden Optionen aus:

- `--db-cluster-identifier` – Die DB-Cluster-Kennung.
- `--cloudwatch-logs-export-configuration`— Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster aktiviert werden sollen.

Sie können Aurora-MySQL-Protokolle auch veröffentlichen, indem Sie einen der folgenden AWS CLI-Befehle ausführen:

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Führen Sie einen dieser AWS CLI-Befehle mit den folgenden Optionen aus:

- `--db-cluster-identifier` – Die DB-Cluster-Kennung.
- `--engine`— Die Datenbank-Engine.
- `--enable-cloudwatch-logs-exports`— Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster aktiviert werden sollen.

Je nach ausgeführtem AWS CLI-Befehl müssen möglicherweise noch weitere Optionen angegeben werden.

Example

Der folgende Befehl modifiziert einen vorhandenen Aurora MySQL-DB-Cluster zum Veröffentlichen von Protokolldateien in CloudWatch Logs.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit","instance"]}'
```

Example

Der folgende Befehl erstellt einen Aurora MySQL-DB-Cluster zum Veröffentlichen von Protokolldateien in CloudWatch Logs.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine aurora \
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

Für Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine aurora ^
    --enable-cloudwatch-logs-exports
 '["error","general","slowquery","audit","instance"]'
```

RDS-API

Sie können Aurora MySQL-Protokolle für bereitgestellte Cluster mithilfe der RDS-API veröffentlichen. Hierzu führen Sie die Aktion ModifyDBCluster mit den folgenden Optionen aus:

- `DBClusterIdentifier` – Die DB-Cluster-Kennung.
- `CloudwatchLogsExportConfiguration`— Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster aktiviert werden sollen.

Sie können Aurora MySQL-Protokolle auch mit der RDS-API veröffentlichen, indem Sie eine der folgenden RDS-API-Operationen ausführen:

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

Verwenden Sie dazu die RDS-API-Operation mit den folgenden Parametern:

- `DBClusterIdentifier` – Die DB-Cluster-Kennung.
- `Engine`— Die Datenbank-Engine.
- `EnableCloudwatchLogsExports`— Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster aktiviert werden sollen.

Je nach ausgeführtem AWS CLI-Befehl müssen möglicherweise noch weitere Parameter angegeben werden.

Überwachung von Protokollereignissen in Amazon CloudWatch

Nach dem Aktivieren von Aurora MySQL-Protokollereignissen können Sie die Ereignisse in Amazon CloudWatch Logs überwachen. Eine neue Protokollgruppe für den Aurora-DB-Cluster wird automatisch mit folgendem Präfix erstellt. Dabei repräsentiert *`cluster-name`* den DB-Clusternamen und *`log_type`* den Protokolltyp.

```
/aws/rds/cluster/cluster-name/log_type
```

Wenn Sie beispielsweise die Exportfunktion so konfigurieren, dass das Slow-Query-Protokoll für einen DB-Cluster namens `mydbcluster` eingeschlossen wird, werden die Slow-Query-Daten in der Protokollgruppe `/aws/rds/cluster/mydbcluster/slowquery` gespeichert.

Die Ereignisse aus allen Instances in Ihrem Cluster werden von einer Protokollgruppe unter Verwendung verschiedener Protokollstreams erfasst. Das Verhalten hängt davon ab, welche der folgenden Bedingungen zutrifft:

- Eine Protokollgruppe mit dem angegebenen Namen existiert.

  Aurora verwendet die vorhandene Protokollgruppe, um Protokolldaten für den Cluster zu exportieren. Um Protokollgruppen mit vordefinierten Protokollaufbewahrungszeiten, Metrikfiltern und Kundenzugriff zu erstellen, können Sie die automatisierte Kommunikation verwenden, zum Beispiel AWS CloudFormation.
- Eine Protokollgruppe mit dem angegebenen Namen existiert nicht.

  Wenn in der Protokolldatei für die Instance ein übereinstimmender Protokolleintrag erkannt wird, erstellt Aurora MySQL automatisch eine neue Protokollgruppe in CloudWatch Logs. Die

Protokollgruppe nutzt den standardmäßigen Aufbewahrungszeitraum für Protokolle Never Expire (Läuft nie ab).

Um die Protokollaufbewahrungszeit zu ändern, können Sie die CloudWatch-Logs-Konsole, die AWS CLI oder die CloudWatch-Logs-API verwenden. Weitere Informationen zum Anpassen des Aufbewahrungszeitraums von Protokollen in CloudWatch Logs finden Sie unter Ändern der Aufbewahrung von Protokolldaten in CloudWatch Logs.

Um Informationen in Protokollereignissen für einen DB-Cluster zu suchen, können Sie die CloudWatch-Logs-Konsole, die AWS CLI oder die CloudWatch-Logs-API verwenden. Weitere Informationen finden Sie unter Suchen und Filtern von Protokolldaten.

Amazon Aurora MySQL-Labor-Modus

> **Important**
>
> Der Labormodus wurde in Aurora-MySQL-Version 2 eingeführt, um die Optimierung für schnelle DDL zu aktivieren, die die Effizienz bestimmter DDL-Operationen verbesserte. In Aurora-MySQL-Version 3 wurde der Labormodus entfernt und schnelle DDL wurde durch die MySQL-8.0-Funktion namens Sofortige DDL ersetzt.

Der Aurora-Labor-Modus wird verwendet, um Aurora-Funktionen zu aktivieren, die in der aktuellen Aurora-Datenbankversion vorhanden, jedoch standardmäßig deaktiviert sind. Auch wenn die Funktionen des Aurora-Labor-Modus nicht für den Einsatz in Produktions-DB-Clustern empfohlen werden, können Sie den Aurora-Labor-Modus verwenden, um diese Funktionen für DB-Cluster in Ihrer Entwicklungs- und Testumgebung zu aktivieren. Weitere Informationen über Aurora-Funktionen, die im Aurora-Labor-Modus zur Verfügung stehen, finden Sie unter Funktionen des Aurora-Labor-Modus.

Der Parameter `aurora_lab_mode` ist ein Parameter der Instance-Ebene, der sich in der Standardparametergruppe befindet. Der Parameter ist in der Standard-DB-Parametergruppe von Oracle auf `0` (deaktiviert) gesetzt. Um den Aurora-Labormodus zu aktivieren, erstellen Sie eine benutzerdefinierte Parametergruppe, setzen den Parameter `aurora_lab_mode` in der benutzerdefinierten Parametergruppe auf `1` (aktiviert) und ändern eine oder mehrere DB-Instances in Ihrem Aurora-Cluster so, dass die benutzerdefinierte Parametergruppe verwendet wird. Stellen Sie dann eine Verbindung zu dem jeweiligen Instance-Endpunkt her, um die Lab-Modus funktionen auszuprobieren. Weitere Informationen zum Ändern von DB-Parametergruppen finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora. Weitere Informationen zu Parametergruppen und Amazon Aurora finden Sie unter Aurora MySQL Konfigurationsparameter.

Funktionen des Aurora-Labor-Modus

Die folgende Aurora-Funktion ist derzeit verfügbar, wenn der Aurora-Labor-Modus aktiviert ist. Sie müssen den Aurora-Labor-Modus aktivieren, bevor Sie diese Funktionen verwenden können.

Schnelle DDL

Mit dieser Funktion können Sie eine `ALTER TABLE` *`tbl_name`* `ADD COLUMN` *`col_name column_definition`*-Operation praktisch augenblicklich ausführen. Die Operation wird

abgeschlossen, ohne dass ein Kopieren der Tabelle erforderlich wäre und ohne eine materielle Auswirkung auf andere DML-Statements zu haben. Da kein temporärer Speicher für eine Tabellenkopie benötigt wird, haben DDL-Anweisungen praktische Vorteile, sogar für große Tabellen auf kleinen Instance-Klassen.

Fast DDL wird zurzeit nur für das Hinzufügen löschbarer Spalten ohne Standardwert am Ende einer Tabelle unterstützt. Weitere Informationen zur Verwendung dieses Features finden Sie unter Ändern von Tabellen in Amazon Aurora mithilfe von Fast DDL.

Bewährte Methoden mit Amazon Aurora MySQL

Dieses Thema beinhaltet Informationen zu bewährten Methoden und Optionen für die Verwendung oder Migration von Daten in einem Amazon-Aurora-MySQL-DB-Cluster. Die Informationen in diesem Thema fassen einige der Richtlinien und Verfahren zusammen, die Sie unter Verwalten eines Amazon Aurora-DB-Clusters.

Inhalt

- Feststellen, mit welcher DB-Instance Sie verbunden sind
- Bewährte Verfahren für die Leistung und Skalierung von Aurora MySQL
  - Verwendung von T-Instance-Klassen für Entwicklung und Tests
  - Optimierung von mit Aurora MySQL indizierten Join-Abfragen mit asynchronem Key Prefetch
    - Asynchrones Key Prefetch aktivieren
    - Optimieren von Abfragen für asynchrones Key Prefetch
  - Optimierung von großen Aurora-MySQL-Join-Abfragen mit Hash-Joins
    - Aktivieren von Hash-Joins
    - Optimieren von Abfragen für Hash-Joins
  - Verwenden von Amazon Aurora für das Skalieren von Lesevorgängen in Ihrer MySQL-Datenbank
  - Optimierung von Zeitstempeloperationen
  - Überlauffehler bei virtuellen Index-IDs
- Bewährte Methoden für die Hochverfügbarkeit von Aurora MySQL
  - Verwenden von Amazon Aurora zur Notfallwiederherstellung Ihrer MySQL-Datenbanken
  - Migrieren von MySQL zu Amazon Aurora MySQL mit reduzierter Ausfallzeit
  - Vermeiden von Leistungseinbußen, automatischem Neustart und Failover für DB-Instances von Aurora MySQL
- Empfehlungen für MySQL-Funktionen in Aurora MySQL
  - Verwenden von Multithread-Replikation in Aurora MySQL
  - Aufrufen von AWS Lambda-Funktionen mit Hilfe von nativen MySQL-Funktionen
  - Vermeiden von XA-Transaktionen mit Amazon Aurora MySQL
  - Aktivieren von Fremdschlüsseln während DML-Anweisungen
  - Konfigurieren, wie oft der Protokollpuffer geleert wird

- Minimieren und Beheben von Aurora-MySQL-Deadlocks
  - Minimieren von InnoDB-Deadlocks
  - Überwachen von InnoDB-Deadlocks
- Bewerten der DB-Instance-Nutzung für Aurora MySQL mit Amazon-CloudWatch-Metriken

Feststellen, mit welcher DB-Instance Sie verbunden sind

Um herauszufinden, zu welcher DB-Instance in einem Aurora-MySQL-DB-Cluster eine Verbindung besteht, prüfen Sie die globale Variable `innodb_read_only` wie im folgenden Beispiel gezeigt.

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

Die Variable `innodb_read_only` wird auf `0N` gesetzt, wenn Sie mit einer Reader-DB-Instance verbunden sind. Diese Einstellung ist `0FF`, wenn Sie mit einer Writer-DB-Instance verbunden sind, z. B. als primäre Instance in einem bereitgestellten Cluster.

Dieser Ansatz kann hilfreich sein, wenn Sie zu Ihrem Anwendungscode eine Logik hinzufügen möchten, um den Workload auszugleichen, oder sicherzustellen, dass eine Schreiboperation die richtige Verbindung verwendet.

Bewährte Verfahren für die Leistung und Skalierung von Aurora MySQL

Sie können die folgenden bewährten Verfahren anwenden, um die Leistung und Skalierbarkeit Ihrer Aurora-MySQL-Cluster zu verbessern.

Themen

- Verwendung von T-Instance-Klassen für Entwicklung und Tests
- Optimierung von mit Aurora MySQL indizierten Join-Abfragen mit asynchronem Key Prefetch
- Optimierung von großen Aurora-MySQL-Join-Abfragen mit Hash-Joins
- Verwenden von Amazon Aurora für das Skalieren von Lesevorgängen in Ihrer MySQL-Datenbank
- Optimierung von Zeitstempeloperationen
- Überlauffehler bei virtuellen Index-IDs

Verwendung von T-Instance-Klassen für Entwicklung und Tests

Amazon Aurora MySQL-Instances, die die DB-Instance-Klassen `db.t2`, `db.t3` oder `db.t4g` verwenden, eignen sich am besten für Anwendungen, die nicht über einen längeren Zeitraum eine hohe Arbeitslast unterstützen. Die T-Instances sind so konzipiert, dass sie eine mäßige Basisleistung bieten und je nach Workload eine deutlich höhere Leistung erreichen können. Sie eignen sich für Workloads, die die volle CPU-Leistung selten oder uneinheitlich verwenden, jedoch gelegentlich Spitzenlasten verarbeiten müssen. Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter Burstable Performance Instances.

Wenn Ihr Aurora-Cluster größer als 40 TB ist, sollten Sie die T-Instance-Klassen nicht verwenden. Wenn Ihre Datenbank ein großes Datenvolumen hat, kann der Speicher-Overhead für die Verwaltung von Schemaobjekten die Kapazität einer T-Instance übersteigen.

Aktivieren Sie das MySQL-Leistungsschema nicht auf Amazon Aurora MySQL T-Instances. Wenn das Leistungsschema aktiviert ist, geht der Instance möglicherweise der Speicher aus.

> **Tip**
>
> Wenn Ihre Datenbank manchmal inaktiv ist, aber zu anderen Zeiten eine erhebliche Workload aufweist, können Sie Aurora Serverless v2 als Alternative zu T-Instances verwenden. Mit Aurora Serverless v2 definieren Sie einen Kapazitätsbereich und Aurora skaliert Ihre Datenbank je nach aktueller Workload automatisch nach oben oder unten. Details zur Verwendung finden Sie unter Verwenden von Aurora Serverless v2. Die Versionen der Datenbank-Engine, die Sie mit Aurora Serverless v2 verwenden können, finden Sie unter Anforderungen und Einschränkungen für Aurora Serverless v2.

Wenn Sie eine T-Instance als DB-Instance in einem Aurora-MySQL DB-Cluster verwenden, empfehlen wir Folgendes:

- Verwenden Sie für alle Instances in Ihrem DB–Cluster dieselbe DB-Instance-Klasse. Wenn Sie zum Beispiel `db.t2.medium` für Ihre Writer-Instance verwenden, empfehlen wir Ihnen, `db.t2.medium` auch für Ihre Reader-Instances zu verwenden.
- Nehmen Sie keine speicherbezogenen Konfigurationseinstellungen vor, wie z. B. `innodb_buffer_pool_size`. Aurora verwendet einen hochgradig abgestimmten Satz von Standardwerten für Speicherpuffer auf T-Instances. Diese speziellen Voreinstellungen

sind erforderlich, damit Aurora auf Instances mit begrenztem Speicherplatz läuft. Wenn Sie irgendwelche speicherbezogenen Einstellungen auf einer T-Instance ändern, ist es viel wahrscheinlicher, dass Sie auf Out-of-Memory-Bedingungen stoßen, selbst wenn Ihre Änderung darauf abzielt, die Puffergrößen zu erhöhen.

- Überwachen Sie Ihr CPU-Guthaben (`CPUCreditBalance`), um sicherzustellen, dass es auf einem akzeptablen Stand ist. CPU-Guthaben werden nämlich zu denselben Tarifen angesammelt, wie sie verbraucht werden.

  Wenn Sie Ihr CPU-Guthaben aufgebraucht haben, werden Sie sofort eine Leistungsminderung der verfügbaren CPU und erhöhte Latenzen der Lese- und Schreibvorgänge in der Instance feststellen. Diese Situation führt zu einem starken Rückgang der Gesamtperformance der Instance.

  Wenn Ihr CPU-Guthaben nicht ausreicht, empfehlen wir Ihnen, Ihre DB-Instance so zu ändern, dass sie eine der unterstützten R DB-Instance-Klassen (Scale Compute) verwendet.

  Weitere Informationen über das Überwachen von Metriken finden Sie unter Anzeigen von Metriken in der Amazon-RDS-Konsole.

- Überwachen Sie die Replikverzögerung (`AuroraReplicaLag`) zwischen der Writer-Instance und den Reader-Instances.

  Wenn einer Reader-Instance das CPU-Guthaben vor der Writer-Instance ausgeht, kann die daraus resultierende Verzögerung dazu führen, dass die Reader-Instance häufig neu gestartet werden muss. Dieses Ergebnis tritt häufig auf, wenn eine Anwendung eine hohe Anzahl von Lesevorgängen hat, die auf die Reader-Instance verteilt sind, während die Schreib-Instance eine minimale Anzahl von Schreibvorgängen hat.

  Wenn Sie einen anhaltenden Anstieg der Verzögerung bei der Replikation feststellen, stellen Sie sicher, dass Ihr CPU-Guthaben für die Reader-Instances in Ihrem DB-Cluster nicht erschöpft ist.

  Wenn Ihr CPU-Guthaben nicht ausreicht, empfehlen wir Ihnen, Ihre DB-Instance so zu modifizieren, dass sie eine der unterstützten R DB-Instance-Klassen (scale compute) verwendet.

- Halten Sie die Anzahl an Inserts pro Transaktion unter 1 Million für DB-Cluster, bei denen die Binärprotokollierung aktiviert ist.

  Wenn in der DB-Cluster-Parametergruppe für Ihr DB-Cluster der Parameter `binlog_format` auf einen anderen Wert als `OFF` eingestellt ist, könnte es in Ihrem DB-Cluster zu Speicherengpässen kommen, wenn Ihr DB-Cluster große Transaktionen mit mehr als 1 Million einzufügender Zeilen empfängt. Sie können den freien Speicher (`FreeableMemory`) metrisch überwachen, um

festzustellen, ob Ihrem DB-Cluster der verfügbare Speicher ausgeht. Sie überprüfen dann die Metrik der Schreibvorgänge (`VolumeWriteIOPS`), um zu sehen, ob eine Writer-Instance eine große Last von Schreibvorgängen empfängt. Wenn dies der Fall ist, empfehlen wir, Ihre Anwendung zu aktualisieren, um die Anzahl von Einfügungen in einer Transaktion auf weniger als 1 Million zu begrenzen. Alternativ können Sie Ihre Instance so modifizieren, dass sie eine der unterstützten R DB-Instance-Klassen verwendet (scale compute).

Optimierung von mit Aurora MySQL indizierten Join-Abfragen mit asynchronem Key Prefetch

Aurora MySQL kann die asynchrone Key Prefetch (AKP)-Funktion verwenden, um die Performance von Abfragen zu verbessern, die Tabellen über Indizes hinweg verknüpfen. Diese Funktion verbessert die Leistung, indem sie die Zeilen vorwegnimmt, die zum Ausführen von Abfragen benötigt werden, bei denen eine JOIN-Abfrage die Verwendung des Batched Key Access (BKA) Join-Algorithmus und der Multi-Range Read (MRR)-Optimierungsfunktionen erfordert. Weitere Informationen zu BKA und MRR finden Sie unter Block Nested-Loop und Batched Key Access Joins und Multi-Range Read Optimization in der MySQL-Dokumentation.

Um einen Vorteil aus der AKP-Funktion zu ziehen, muss eine Abfrage sowohl BKA als auch MRR verwenden. Typischerweise tritt eine solche Abfrage auf, wenn die JOIN-Klausel einer Abfrage einen Sekundärindex verwendet, aber auch einige Spalten aus dem Primärindex benötigt. Sie können beispielsweise AKP verwenden, wenn eine JOIN-Klausel einen Equijoin auf Indexwerten zwischen einer kleinen äußeren und einer großen inneren Tabelle darstellt und der Index auf der größeren Tabelle hoch selektiv ist. AKP arbeitet mit BKA und MRR zusammen, um während der Evaluierung der JOIN-Klausel einen sekundären Index-Lookup durchzuführen. AKP identifiziert die Zeilen, die während der Auswertung der JOIN-Klausel benötigt werden, um die Abfrage auszuführen. Anschließend werden die Seiten, die diese Zeilen enthalten, mit einem Hintergrund-Thread asynchron in den Speicher geladen, bevor die Abfrage ausgeführt wird.

AKP ist für Aurora MySQL Version 2.10 und höher sowie Version 3 verfügbar. Weitere Informationen zu den Aurora MySQL-Versionen erhalten Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.

Asynchrones Key Prefetch aktivieren

Sie können die AKP-Funktion aktivieren, indem Sie `aurora_use_key_prefetch`, eine MySQL-Servervariable, auf on setzen. Standardmäßig ist dieser Wert auf on festgelegt. AKP kann jedoch erst aktiviert werden, wenn Sie auch den BKA-Join-Algorithmus aktivieren und die preisbasierte

MRR-Funktionalität deaktivieren. Dazu müssen Sie die folgenden Werte für `optimizer_switch` setzen, eine Variable von MySQL-Server:

- Setzen Sie `batched_key_access` auf `on`. Dieser Wert steuert die Verwendung des BKA-Join-Algorithmus. Standardmäßig ist dieser Wert auf `off` festgelegt.
- Setzen Sie `mrr_cost_based` auf `off`. Dieser Wert steuert die Nutzung der kostenbasierten MRR-Funktionalität. Standardmäßig ist dieser Wert auf `on` festgelegt.

Derzeit können Sie diese Werte nur auf Sitzungsebene festlegen. Das folgende Beispiel veranschaulicht, wie diese Werte gesetzt werden können, um AKP für die aktuelle Sitzung durch Ausführen von SET-Anweisungen zu aktivieren.

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

Ähnlich können Sie mit SET-Anweisungen AKP und den BKA Join-Algorithmus deaktivieren und kostenbasierte MRR-Funktionalität für die aktuelle Sitzung wieder aktivieren, wie im folgenden Beispiel gezeigt.

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

Weitere Informationen über die Optimierungsschalter batched_key_access und mrr_cost_based finden Sie unter Switchable Optimizations in der MySQL-Dokumentation.

Optimieren von Abfragen für asynchrones Key Prefetch

Sie können bestätigen, ob eine Abfrage die AKP-Funktion nutzen können soll. Verwenden Sie dazu die `EXPLAIN`-Anweisung, um die Abfrage vor der Ausführung zu profilieren. Die `EXPLAIN`-Anweisung stellt Informationen über den Ausführungsplan bereit, der für eine angegebene Abfrage verwendet werden soll.

In der Ausgabe der `EXPLAIN`-Anweisung beschreibt die Spalte `Extra` zusätzliche Informationen, die im Ausführungsplan eingeschlossen sind. Wenn das AKP-Feature auf eine in der Abfrage verwendete Tabelle zutrifft, enthält diese Spalte einen der folgenden Werte:

- `Using Key Prefetching`

- `Using join buffer (Batched Key Access with Key Prefetching)`

Das folgende Beispiel zeigt die Verwendung von `EXPLAIN`, um den Ausführungsplan für eine Abfrage anzuzeigen, die von AKP profitieren kann.

```
mysql> explain select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
    -> from
    ->     partsupp,
    ->     supplier,
    ->     nation
    -> where
    ->     ps_suppkey = s_suppkey
    ->     and s_nationkey = n_nationkey
    ->     and n_name = 'ETHIOPIA'
    -> group by
    ->     ps_partkey having
    ->         sum(ps_supplycost * ps_availqty) > (
    ->             select
    ->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
    ->             from
    ->                 partsupp,
    ->                 supplier,
    ->                 nation
    ->             where
    ->                 ps_suppkey = s_suppkey
    ->                 and s_nationkey = n_nationkey
    ->                 and n_name = 'ETHIOPIA'
    ->         )
    -> order by
    ->     value desc;
+----+-------------+----------+------+----------------------+--------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
| id | select_type | table    | type | possible_keys        | key          | key_len
 | ref                              | rows | filtered | Extra
                            |
+----+-------------+----------+------+----------------------+--------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
```

```
|  1 | PRIMARY     | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where; Using temporary;
 Using filesort                 |
|  1 | PRIMARY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                              |
|  1 | PRIMARY     | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
|  2 | SUBQUERY    | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where
                             |
|  2 | SUBQUERY    | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                              |
|  2 | SUBQUERY    | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

Weitere Informationen über das `EXPLAIN`-Ausgabeformat finden Sie unter Extended EXPLAIN Output Format in der MySQL-Dokumentation.

Optimierung von großen Aurora-MySQL-Join-Abfragen mit Hash-Joins

Wenn Sie eine große Datenmenge mit Hilfe eines Equijoins verknüpfen müssen, kann ein Hash-Join die Abfrageleistung verbessern. Sie können Hash-Joins für Aurora MySQL aktivieren.

Eine Hash-Join-Spalte kann ein beliebiger komplexer Ausdruck sein. In einer Hash-Join-Spalte haben Sie folgende Möglichkeiten, Datentypen übergreifend zu vergleichen:

- Sie können alles über die Kategorie der präzisen numerischen Datentypen hinweg vergleichen, wie z. B. `int`, `bigint`, `numeric` und `bit`.
- Sie können alles über die Kategorie der ungefähren numerischen Datentypen hinweg vergleichen, wie z. B. `float` und `double`.
- Sie können Elemente über String-Typen hinweg vergleichen, wenn die String-Typen den gleichen Zeichensatz und die gleiche Sortierreihenfolge haben.

- Sie können Elemente mit Datums- und Zeitstempel-Datentypen vergleichen, wenn die Typen identisch sind.

> **Note**
>
> Sie können Datentypen verschiedener Kategorien nicht miteinander vergleichen.

Die folgenden Einschränkungen gelten für Hash-Joins für Aurora MySQL:

- Links-Rechts-Außen-Joins werden für Aurora MySQL-Version 2 nicht unterstützt, für Version 3 hingegen schon.
- Semijoins wie Subqueries werden nicht unterstützt, es sei denn, die Subqueries erfolgen zuerst.
- Mehrfach-Tabellen-Updates oder -Löschungen werden nicht unterstützt.

> **Note**
>
> Einzel-Tabellen-Updates oder -Löschungen werden unterstützt.

- BLOB- und Geodatentyp-Spalten können keine Join-Spalten in einem Hash-Join sein.

Aktivieren von Hash-Joins

So aktivieren Sie Hash-Joins:

- Aurora-MySQL-Version 2 – Stellen Sie den DB-Parameter oder den DB-Cluster-Parameter `aurora_disable_hash_join` auf `0` ein. Durch Deaktivierung von `aurora_disable_hash_join` wird der Wert von `optimizer_switch` auf `hash_join=on` gesetzt.
- Aurora-MySQL-Version 3 – Stellen Sie den MySQL-Serverparameter `optimizer_switch` auf `block_nested_loop=on` ein.

Hash-Joins sind standardmäßig in Aurora-MySQL-Version 3 aktiviert und in Aurora-MySQL-Version 2 deaktiviert. Das folgende Beispiel zeigt, wie man Hash-Joins für Aurora-MySQL-Version 3 aktivieren kann. Sie können zuerst die Anweisung `select @@optimizer_switch` ausgeben, um zu sehen, welche anderen Einstellungen in der `SET`-Parameterzeichenfolge vorhanden sind. Das Aktualisieren

einer Einstellung im Parameter `optimizer_switch` löscht oder ändert die anderen Einstellungen nicht.

```
mysql> SET optimizer_switch='block_nested_loop=on';
```

> **Note**
>
> Für Aurora-MySQL-Version 3 ist Hash-Join-Support in allen Nebenversionen verfügbar und standardmäßig aktiviert.
> Für Aurora MySQL-Version-3 ist Hash-Join-Support in allen Nebenversionen verfügbar und standardmäßig aktiviert. In Aurora-MySQL-Version 2 wird die Hash-Join-Funktion immer durch den `aurora_disable_hash_join`-Wert gesteuert.

Mit dieser Einstellung wählt der Optimierer einen Hash-Join auf der Grundlage von Kosten, Abfragemerkmalen und Ressourcenverfügbarkeit. Wenn die Kalkulation fehlerhaft ist, können Sie den Optimierer zwingen, einen bestimmten Hash-Join zu wählen. Sie erreichen dies, indem Sie `hash_join_cost_based`, eine MySQL-Servervariable, auf `off` setzen. Das folgende Beispiel zeigt, wie Sie den Optimierer zwingen können, einen Hash-Join zu wählen.

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

> **Note**
>
> Diese Einstellung setzt die Entscheidungen des kostenbasierten Optimierers außer Kraft. Während die Einstellung für Tests und Entwicklung nützlich sein kann, empfehlen wir, sie nicht in der Produktion zu verwenden.

Optimieren von Abfragen für Hash-Joins

Um herauszufinden, ob eine Abfrage einen Hash-Join nutzen kann, verwenden Sie die `EXPLAIN`-Anweisung, um die Abfrage zuerst zu profilieren. Die `EXPLAIN`-Anweisung stellt Informationen über den Ausführungsplan bereit, der für eine angegebene Abfrage verwendet werden soll.

In der Ausgabe der EXPLAIN-Anweisung beschreibt die Spalte `Extra` zusätzliche Informationen, die im Ausführungsplan eingeschlossen sind. Wenn ein Hash-Join für die in der Abfrage verwendeten Tabellen gilt, enthält diese Spalte Werte, die den folgenden ähnlich sind:

- `Using where; Using join buffer (Hash Join Outer table` *`table1_name`*`)`
- `Using where; Using join buffer (Hash Join Inner table` *`table2_name`*`)`

Das folgende Beispiel zeigt die Verwendung von EXPLAIN, um den Ausführungsplan für eine Hash-Join-Abfrage anzuzeigen.

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
    ->     WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows |
 Extra                                                              |
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | hj_small | ALL  | NULL          | NULL | NULL    | NULL |    6 |
 Using temporary; Using filesort                                    |
|  1 | SIMPLE      | hj_big   | ALL  | NULL          | NULL | NULL    | NULL |   10 |
 Using where; Using join buffer (Hash Join Outer table hj_big)      |
|  1 | SIMPLE      | hj_big2  | ALL  | NULL          | NULL | NULL    | NULL |   15 |
 Using where; Using join buffer (Hash Join Inner table hj_big2)     |
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
3 rows in set (0.04 sec)
```

In der Ausgabe ist der `Hash Join Inner table` die Tabelle, die zum Aufbau der Hash-Tabelle verwendet wird, und der `Hash Join Outer table` ist die Tabelle, die zum Prüfen der Hash-Tabelle verwendet wird.

Weitere Informationen über das erweiterte EXPLAIN-Ausgabeformat finden Sie unter Extended EXPLAIN Output Format in der MySQL-Produktdokumentation.

In Aurora MySQL 2.08 und höher können Sie mithilfe von SQL-Hinweisen beeinflussen, ob eine Abfrage einen Hash-Join verwendet oder nicht und welche Tabellen für die Build- und Testseite des Joins verwendet werden sollen. Details hierzu finden Sie unter Aurora-MySQL-Hinweise.

Verwenden von Amazon Aurora für das Skalieren von Lesevorgängen in Ihrer MySQL-Datenbank

Sie können Amazon Aurora mit Ihrer MySQL-DB-Instance verwenden, um die Möglichkeiten der Skalierung von Lesevorgängen von Amazon Aurora zu nutzen und den Lese-Workload für Ihre MySQL-DB-Instance zu erweitern. Erstellen Sie einen Aurora-MySQL-DB-Cluster und machen Sie ihn zum Read Replica Ihrer MySQL-DB-Instance, um Aurora für die Skalierung von Lesevorgängen Ihrer MySQL-DB-Instance zu verwenden. Verbinden Sie sich dann mit dem Aurora-MySQL-Cluster, um die Leseabfragen zu verarbeiten. Die Quelldatenbank kann eine RDS für MySQL-DB-Instance oder eine MySQL-Datenbank sein, die außerhalb von Amazon RDS ausgeführt wird. Weitere Informationen finden Sie unter Skalieren von Lesevorgängen für Ihre MySQL-Datenbank mit Amazon Aurora.

Optimierung von Zeitstempeloperationen

Wenn der Wert der Systemvariablen `time_zone` auf `SYSTEM` gesetzt ist, führt jeder MySQL-Funktionsaufruf, der eine Zeitzonenberechnung erfordert, einen Systembibliotheksaufruf aus. Wenn Sie SQL-Anweisungen ausführen, die solche `TIMESTAMP`-Werte mit hoher Nebenläufigkeit zurückgeben oder ändern, kann es zu einer höheren Latenz und CPU-Auslastung sowie zu Sperrkonflikten kommen. Weitere Informationen finden Sie unter time_zone in der MySQL-Dokumentation.

Wir empfehlen, den Wert des DB-Cluster-Parameters `time_zone` in `UTC` zu ändern, um dieses Verhalten zu vermeiden. Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

Der Parameter `time_zone` ist zwar dynamisch (er erfordert keinen Neustart des Datenbankservers), der neue Wert wird jedoch nur für neue Verbindungen verwendet. Um sicherzustellen, dass alle Verbindungen so aktualisiert werden, dass sie den neuen `time_zone`-Wert verwenden, empfehlen wir, Ihre Anwendungsverbindungen nach der Aktualisierung des DB-Cluster-Parameters wiederzuverwenden.

Überlauffehler bei virtuellen Index-IDs

Aurora MySQL begrenzt die Werte für virtuelle Index-IDs auf 8 Bit und verhindert ein Problem, das durch das Undo-Format in MySQL verursacht wird. Wenn ein Index das Limit für virtuelle Index-IDs überschreitet, ist Ihr Cluster möglicherweise nicht verfügbar. Wenn sich ein Index dem Limit für virtuelle Index-IDs nähert oder wenn Sie versuchen, einen Index zu erstellen, der über dem Limit für virtuelle Index-IDs liegt, gibt RDS möglicherweise einen `63955`-Fehler oder `63955`-

Warnungscode aus. Zum Beheben eines Fehlers beim Limit für virtuelle Index-IDs empfehlen wir Ihnen, Ihre Datenbank mit einem logischen Dump und einer Wiederherstellung neu zu erstellen.

Weitere Informationen zu logischen Dumps und Wiederherstellungen für Amazon Aurora MySQL finden Sie unter Migrieren sehr großer Datenbanken zu Amazon Aurora MySQL mit MyDumper und MyLoader. Weitere Informationen zum Zugriff auf Fehlerprotokolle in Amazon Aurora finden Sie unter Überwachen von Amazon Aurora-Protokolldateien.

Bewährte Methoden für die Hochverfügbarkeit von Aurora MySQL

Sie können die folgenden bewährten Verfahren anwenden, um die Verfügbarkeit Ihrer Aurora MySQL-Cluster zu verbessern.

Themen

- Verwenden von Amazon Aurora zur Notfallwiederherstellung Ihrer MySQL-Datenbanken
- Migrieren von MySQL zu Amazon Aurora MySQL mit reduzierter Ausfallzeit
- Vermeiden von Leistungseinbußen, automatischem Neustart und Failover für DB-Instances von Aurora MySQL

Verwenden von Amazon Aurora zur Notfallwiederherstellung Ihrer MySQL-Datenbanken

Sie können Amazon Aurora mit Ihrer MySQL-DB-Instance verwenden, um ein außerhalb liegendes Backup für eine Notfallwiederherstellung zu erstellen. Erstellen Sie einen Amazon-Aurora-DB-Cluster und legen Sie ihn als Read Replica Ihrer MySQL-DB-Instance fest, um Aurora für die Notfallwiederherstellung Ihrer MySQL-DB-Instance zu verwenden. Dies gilt für eine RDS für MySQL-DB-Instance oder eine MySQL-Datenbank, die außerhalb von Amazon RDS ausgeführt wird.

> **Important**
>
> Wenn Sie eine Replikation zwischen Ihrer MySQL-DB-Instance und einem Amazon-Aurora-MySQL-DB-Cluster einrichten, sollten Sie die Replikation prüfen, um sicherzustellen, dass sie sich in einem guten Zustand befindet, und sie bei Bedarf reparieren.

Anweisungen zum Erstellen eines Amazon-Aurora-MySQL-DB-Clusters und Festlegen als Read Replica Ihrer MySQL-DB-Instance finden Sie im Abschnitt Verwenden von Amazon Aurora für das Skalieren von Lesevorgängen in Ihrer MySQL-Datenbank.

Weitere Hinweise zu Notfallwiederherstellungsmodellen finden Sie unter So wählen Sie die beste Notfallwiederherstellungsoption für Ihren Amazon-Aurora-MySQL-Cluster aus.

Migrieren von MySQL zu Amazon Aurora MySQL mit reduzierter Ausfallzeit

Beim Importieren von Daten aus einer MySQL-Datenbank, die eine Live-Anwendung unterstützt, in einen Amazon-Aurora-MySQL-DB-Cluster möchten Sie möglicherweise die Zeit reduzieren, in der der Service während der Migration unterbrochen wird. Hierfür können Sie das Verfahren verwenden, das im Abschnitt zum Importieren von Daten in eine DB-Instance von Amazon RDS für MySQL mit reduzierter Ausfallzeit im Benutzerhandbuch zu Amazon Relational Database Service dokumentiert ist. Diese Prozedur kann insbesondere dann hilfreich sein, wenn Sie mit einer sehr großen Datenbank arbeiten. Mit dieser Prozedur können Sie die Kosten des Imports reduzieren, indem Sie die Menge der Daten, die über das Netzwerk an AWS weitergeleitet werden, minimieren.

Das Verfahren führt Schritte auf, um eine Kopie Ihrer Datenbankdaten auf eine EC2 Amazon-Instance zu übertragen und die Daten in eine neue RDS for MySQL-DB-Instance zu importieren. Da Amazon Aurora mit MySQL kompatibel ist, können Sie stattdessen einen Amazon-Aurora-DB-Cluster für die Amazon-RDS-MySQL-DB-Ziel-Instance verwenden.

Vermeiden von Leistungseinbußen, automatischem Neustart und Failover für DB-Instances von Aurora MySQL

Wenn Sie hohe Workloads ausführen, die über die zugewiesenen Ressourcen Ihrer DB-Instance hinausgehen, können Sie die Ressourcen, auf denen Sie Ihre Anwendung und die Aurora-Datenbank ausführen, aufbrauchen. Um Metriken zu Ihrer Datenbank-Instance wie CPU-Auslastung, Speicherauslastung und Anzahl der verwendeten Datenbankverbindungen zu erhalten, können Sie auf die von Amazon CloudWatch, Performance Insights und Enhanced Monitoring bereitgestellten Metriken zurückgreifen. Informationen zur Überwachung Ihrer DB-Instance finden Sie unter Überwachung von Metriken in einem Amazon-Aurora-Cluster.

Wenn Ihre Workload die von Ihnen verwendeten Ressourcen aufbraucht, wird Ihre DB-Instance möglicherweise langsamer, neu gestartet oder es wird sogar ein Failover auf eine andere DB-Instance durchgeführt. Dies können Sie vermeiden, indem Sie Ihre Ressourcenauslastung überwachen, die Workload, die auf Ihrer DB-Instance ausgeführt wird, untersuchen und gegebenenfalls Optimierungen vornehmen. Wenn Optimierungen keine Verbesserungen der Instance-Metriken und keine Verringerung der Ressourcenauslastung ergeben, sollten Sie erwägen, Ihre DB-Instance hochzuskalieren, bevor diese an ihre Grenzen stößt. Weitere Hinweise zu verfügbaren DB-Instance-Klassen und ihren Spezifikationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Empfehlungen für MySQL-Funktionen in Aurora MySQL

Die folgenden Funktionen sind in Aurora MySQL für die MySQL-Kompatibilität verfügbar. Sie haben jedoch Probleme mit der Leistung, Skalierbarkeit, Stabilität oder Kompatibilität in der Aurora-Umgebung. Wir empfehlen daher, dass Sie bei der Verwendung dieser Funktionen bestimmte Richtlinien einhalten. Wir empfehlen zum Beispiel, bestimmte Funktionen nicht für Aurora-Produktionseinsätze zu verwenden.

Themen

- Verwenden von Multithread-Replikation in Aurora MySQL
- Aufrufen von AWS Lambda-Funktionen mit Hilfe von nativen MySQL-Funktionen
- Vermeiden von XA-Transaktionen mit Amazon Aurora MySQL
- Aktivieren von Fremdschlüsseln während DML-Anweisungen
- Konfigurieren, wie oft der Protokollpuffer geleert wird
- Minimieren und Beheben von Aurora-MySQL-Deadlocks

Verwenden von Multithread-Replikation in Aurora MySQL

Bei der Multithread-Replikation für binäre Protokolle liest ein SQL-Thread Ereignisse aus dem Relay-Log und stellt sie in die Warteschlange, damit SQL-Worker-Threads angewendet werden können. Die SQL-Worker-Threads werden von einem Koordinator-Thread verwaltet. Die binären Protokollereignisse werden nach Möglichkeit parallel angewendet.

Die Multithread-Replikation wird in Aurora-MySQL-Version 3 und Aurora-MySQL-Version 2.12.1 und höher unterstützt.

Für Aurora-MySQL-Versionen unter 3.04 verwendet Aurora standardmäßig die Single-Thread-Replikation, wenn ein Aurora-MySQL-DB-Cluster als Read Replica für die Binärprotokollreplikation verwendet wird.

Frühere Versionen von Aurora-MySQL-Version 2 haben mehrere Probleme in Bezug auf die Multi-Thread-Replikation aus der MySQL Community Edition geerbt. Für diese Versionen empfehlen wir, die Multithread-Replikation in der Produktion nicht zu verwenden.

Wenn Sie die Multi-Thread-Replikation verwenden, empfehlen wir Ihnen, diese vorher gründlich zu testen.

Weitere Informationen zur Verwendung der Replikation in Amazon Aurora finden Sie unter Replikation mit Amazon Aurora. Weitere Informationen zur Multithread-Replikation in Aurora MySQL finden Sie unter Multithread-Replikation für binäre Protokolle.

Aufrufen von AWS Lambda-Funktionen mit Hilfe von nativen MySQL-Funktionen

Wir empfehlen, die nativen MySQL-Funktionen `lambda_sync` und `lambda_async` zu verwenden, um Lambda-Funktionen aufzurufen.

Wenn Sie die veraltete Prozedur `mysql.lambda_async` verwenden, empfehlen wir, dass Sie die Aufrufe an die Prozedur `mysql.lambda_async` in einer gespeicherten Prozedur übergeben. Sie können diese gespeicherte Prozedur aus verschiedenen Quellen aufrufen, wie z. B. Trigger oder Client-Code. Dieser Ansatz kann helfen, Probleme hinsichtlich Impedanz-Unstimmigkeiten zu vermeiden und macht es Ihren Datenbank-Programmieren einfacher Lambda-Funktionen aufzurufen.

Weitere Informationen über das Aufrufen von Lambda-Funktionen in Amazon Aurora finden Sie unter Aufrufen einer Lambda-Funktion aus einem Amazon Aurora MySQL-DB-Cluster.

Vermeiden von XA-Transaktionen mit Amazon Aurora MySQL

Wir empfehlen Ihnen, keine eXtended Architecture (XA)-Transaktionen mit Aurora MySQL zu verwenden, da diese lange Wiederherstellungszeiten verursachen können, wenn sich die XA im Status `PREPARED` befunden hat. Wenn Sie XA-Transaktionen mit Aurora MySQL verwenden müssen, befolgen Sie diese bewährten Verfahren:

- Lassen Sie eine XA-Transaktion nicht im Status `PREPARED` offen.
- Halten Sie XA-Transaktionen so klein wie möglich.

Weitere Informationen zur Verwendung von XA-Transaktionen mit MySQL finden Sie unter XA Transactions in der MySQL-Dokumentation.

Aktivieren von Fremdschlüsseln während DML-Anweisungen

Wir empfehlen Ihnen dringend, keine DDL-Anweisungen (Data Definition Language) auszuführen, wenn die Variable `foreign_key_checks` auf `0` (aus) gesetzt ist.

Wenn Sie Zeilen einfügen oder aktualisieren müssen, die eine vorübergehende Verletzung von Fremdschlüsseln bedingen, gehen Sie wie folgt vor:

1. Setzen Sie `foreign_key_checks` auf `0`.
2. Nehmen Sie Ihre DML-Änderungen (Data Manipulation Language) vor.
3. Stellen Sie sicher, dass Ihre durchgeführten Änderungen keine Fremdschlüsselbedingungen verletzen.
4. Setzen Sie `foreign_key_checks` auf `1` (ein).

Darüber hinaus halten Sie die folgenden anderen bewährten Methoden für Fremdschlüsselbedingungen ein:

- Stellen Sie sicher, dass Ihre Client-Anwendungen die Variable `foreign_key_checks` nicht auf `0` als Teil der Variablen `init_connect` setzen.
- Wenn eine Wiederherstellung aus einer logischen Sicherung, wie beispielsweise `mysqldump`, fehlschlägt oder unvollständig ist, stellen Sie sicher, dass `foreign_key_checks` auf `1` gesetzt ist, bevor Sie innerhalb derselben Sitzung andere Operationen starten. Eine logische Sicherung setzt `foreign_key_checks` beim Start auf `0`.

Konfigurieren, wie oft der Protokollpuffer geleert wird

In der MySQL Community Edition muss der InnoDB-Protokollpuffer in einen dauerhaften Speicher geleert werden, um Transaktionen dauerhaft zu machen. Verwenden Sie den Parameter `innodb_flush_log_at_trx_commit`, um zu konfigurieren, wie oft der Protokollpuffer geleert und auf die Festplatte übertragen wird.

Wenn Sie den Parameter `innodb_flush_log_at_trx_commit` auf den Standardwert 1 festlegen, wird der Protokollpuffer bei jedem Transaktions-Commit geleert. Diese Einstellung hilft, die Datenbank ACID-konform zu halten. Wir empfehlen, die Standardeinstellung 1 beizubehalten.

Wenn Sie `innodb_flush_log_at_trx_commit` in einen nicht standardmäßigen Wert ändern, kann dies dazu beitragen, die Latenz der Data Manipulation Language (DML) zu verringern, beeinträchtigt jedoch die Haltbarkeit der Protokolldatensätze. Durch diese mangelnde Haltbarkeit ist die Datenbank nicht ACID-konform. Ihre Datenbanken sollten ACID-konform sein, um das Risiko von Datenverlusten bei einem Serverneustart zu vermeiden. Weitere Informationen zu diesem Parameter finden Sie unter innodb_flush_log_at_trx_commit in der MySQL-Dokumentation.

In Aurora MySQL wird die Redo-Protokollverarbeitung auf die Speicherschicht verlagert, sodass auf der DB-Instance kein Leeren in Protokolldateien erfolgt. Wenn ein Schreibvorgang ausgeführt wird, werden Redo-Protokolle von der Writer-DB-Instance direkt an das Aurora-Cluster-Volume gesendet.

Die einzigen Schreibvorgänge, die das Netzwerk passieren, sind Redo-Protokolldatensätze. Auf der Datenbankebene werden grundsätzlich keine Seiten geschrieben.

Standardmäßig wartet jeder Thread, der einen Commit für eine Transaktion ausführt, auf die Bestätigung vom Aurora-Cluster-Volume. Diese Bestätigung gibt an, dass dieser Datensatz und alle vorherigen Redo-Protokolldatensätze geschrieben wurden und das Quorum erreicht haben. Dadurch, dass die Protokolldatensätze beibehalten und das Quorum erreicht wird, ist die Transaktion dauerhaft gemacht worden, sei es durch Autocommit oder explizites Commit. Weitere Informationen zur Aurora-Speicherarchitektur finden Sie unter Amazon Aurora storage demystified (Amazon-Aurora-Speicher entmystifiziert).

Aurora MySQL leert keine Protokolle in Datendateien, wie dies bei der MySQL Community Edition der Fall ist. Sie können den Parameter `innodb_flush_log_at_trx_commit` jedoch verwenden, um Haltbarkeitsbeschränkungen beim Schreiben von Redo-Protokolldatensätzen auf das Aurora-Clustervolume zu lockern.

Für Aurora-MySQL-Version 2:

- `innodb_flush_log_at_trx_commit` = 0 oder 2 – Die Datenbank wartet nicht auf die Bestätigung, dass die Redo-Protokolldatensätze auf das Aurora-Cluster-Volume geschrieben wurden.
- `innodb_flush_log_at_trx_commit` = 1 – Die Datenbank wartet auf die Bestätigung, dass die Redo-Protokolldatensätze auf das Aurora-Cluster-Volume geschrieben wurden.

Für Aurora-MySQL-Version 3:

- `innodb_flush_log_at_trx_commit` = 0 – Die Datenbank wartet nicht auf die Bestätigung, dass die Redo-Protokolldatensätze auf das Aurora-Cluster-Volume geschrieben wurden.
- `innodb_flush_log_at_trx_commit` = 1 oder 2 – Die Datenbank wartet auf die Bestätigung, dass die Redo-Protokolldatensätze auf das Aurora-Cluster-Volume geschrieben wurden.

Um in Aurora-MySQL-Version 3 dasselbe nicht standardmäßige Verhalten zu erzielen, das Sie mit dem Wert 0 oder 2 in Aurora-MySQL-Version 2 erzielen würden, setzen Sie den Parameter daher auf 0.

Diese Einstellungen können zwar die DML-Latenz für den Client verringern, sie können im Falle eines Failovers oder Neustarts aber auch zu Datenverlust führen. Daher empfehlen wir, für den Parameter `innodb_flush_log_at_trx_commit` den Standardwert 1 beizubehalten.

Während Datenverlust sowohl bei der MySQL Community Edition als auch bei Aurora MySQL auftreten kann, unterscheidet sich das Verhalten in jeder Datenbank aufgrund ihrer unterschiedlichen Architekturen. Diese Unterschiede in der Architektur können zu unterschiedlich starkem Datenverlust führen. damit sichergestellt wird, dass Ihre Datenbank ACID-konform ist, legen Sie `innodb_flush_log_at_trx_commit` immer auf den Wert 1 fest.

> **Note**
>
> Bevor Sie `innodb_flush_log_at_trx_commit` in Aurora-MySQL-Version 3 in einen anderen Wert als 1 ändern können, müssen Sie zuerst den Wert von `innodb_trx_commit_allow_data_loss` auf 1 ändern. Auf diese Weise erkennen Sie das Risiko eines Datenverlusts an.

Minimieren und Beheben von Aurora-MySQL-Deadlocks

Bei Benutzern, die Workloads ausführen, bei denen regelmäßig Einschränkungen für eindeutige sekundäre Indizes oder Fremdschlüssel verletzt werden, kann es bei der gleichzeitigen Änderung von Datensätzen auf derselben Datenseite zu erhöhten Deadlocks und Wartezeitüberschreitungen bei Sperren kommen. Diese Deadlocks und Zeitüberschreitungen sind auf einen Bugfix der MySQL Community Edition zurückzuführen.

Dieser Bugfix ist in den MySQL-Community-Edition-Versionen 5.7.26 und höher enthalten und wurde in die Aurora-MySQL-Versionen 2.10.3 und höher zurückportiert. Der Bugfix ist notwendig, um die Serialisierbarkeit zu erzwingen, indem zusätzliche Sperren für diese Arten von Data Manipulation Language (DML)-Operationen für Änderungen an Datensätzen in einer InnoDB-Tabelle implementiert werden. Dieses Problem wurde im Rahmen einer Untersuchung von Deadlock-Problemen aufgedeckt, die durch einen früheren Bugfix der MySQL Community Edition verursacht wurden.

Mit dem Bugfix wurde die interne Behandlung für das teilweise Rollback eines Tupel-(Zeilen-)Updates in der InnoDB-Speicher-Engine geändert. Operationen, die zu Einschränkungsverstößen bei Fremdschlüsseln oder eindeutigen Sekundärindizes führen, verursachen ein partielles Rollback. Dies beinhaltet, ist aber nicht beschränkt auf gleichzeitige `INSERT...ON DUPLICATE KEY UPDATE`-, `REPLACE INTO`,- und `INSERT IGNORE`-Anweisungen (upserts).

In diesem Zusammenhang bezieht sich partielles Rollback nicht auf das Rollback von Transaktionen auf Anwendungsebene, sondern auf ein internes InnoDB-Rollback von Änderungen an einem

gruppierten Index, wenn ein Einschränkungsverstoß auftritt. Beispielsweise wird während einer Upsert-Operation ein doppelter Schlüsselwert gefunden.

In einem normalen Einfügevorgang erstellt InnoDB automatisch gruppierte und sekundäre Indexeinträge für jeden Index. Wenn InnoDB während einer Upsert-Operation einen doppelten Wert in einem eindeutigen sekundären Index erkennt, muss der eingefügte Eintrag im gruppierten Index rückgängig gemacht werden (partielles Rollback), und die Aktualisierung muss dann auf die vorhandene doppelte Zeile angewendet werden. Während dieses internen partiellen Rollback-Schritts muss InnoDB jeden Datensatz sperren, der als Teil des Vorgangs angezeigt wird. Der Bugfix gewährleistet die Serialisierbarkeit von Transaktionen, indem nach dem partiellen Rollback eine zusätzliche Sperrung eingeführt wird.

Minimieren von InnoDB-Deadlocks

Sie können die folgenden Ansätze verwenden, um die Häufigkeit von Deadlocks in Ihrer Datenbank-Instance zu reduzieren. Weitere Beispiele finden Sie in der MySQL-Dokumentation.

1. Um die Wahrscheinlichkeit von Deadlocks zu verringern, sollten Sie für Transaktionen sofort einen Commit ausführen, nachdem Sie die entsprechenden Änderungen vorgenommen haben. Teilen Sie dazu große Transaktionen (mehrere Zeilenaktualisierungen zwischen Commits) in kleinere Transaktionen auf. Wenn Sie Zeilen stapelweise einfügen, versuchen Sie, die Größe der Stapeleinfügungen zu reduzieren, insbesondere wenn Sie die zuvor genannten Upsert-Operationen verwenden.

   Um die Anzahl möglicher partieller Rollbacks zu reduzieren, können Sie einige der folgenden Ansätze ausprobieren:

   a. Ersetzen Sie Batch-Einfügeoperationen durch das Einfügen einer Zeile nach der anderen. Dadurch kann die Zeit reduziert werden, in der Sperren aufgrund von Transaktionen, die möglicherweise zu Konflikten führen, aufrechterhalten bleiben.

   b. Anstatt `REPLACE INTO` zu verwenden, schreiben Sie die SQL-Anweisung in eine Transaktion mit mehreren Anweisungen um, z. B. die folgende:

   ```
   BEGIN;
   DELETE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

   c. Anstatt `INSERT...ON DUPLICATE KEY UPDATE` zu verwenden, schreiben Sie die SQL-Anweisung in eine Transaktion mit mehreren Anweisungen um, z. B. die folgende:

```
BEGIN;
SELECT rows that conflict on secondary indexes;
UPDATE conflicting rows;
INSERT new rows;
COMMIT;
```

2. Vermeiden Sie lang dauernde Transaktionen, ob aktiv oder inaktiv, die Sperren möglicherweise aufrechterhalten. Dazu gehören interaktive MySQL-Client-Sitzungen, die möglicherweise über einen längeren Zeitraum geöffnet sind, wenn für eine Transaktion kein Commit ausgeführt wird. Bei der Optimierung von Transaktionsgrößen oder Batch-Größen können die Auswirkungen in Abhängigkeit von einer Reihe von Faktoren wie Parallelität, Anzahl der Duplikate und Tabellenstruktur variieren. Alle Änderungen sollten auf der Grundlage Ihrer Workload implementiert und getestet werden.
3. In einigen Situationen können Deadlocks auftreten, wenn zwei Transaktionen versuchen, auf dieselben Datensätze, entweder in einer oder mehreren Tabellen, in unterschiedlicher Reihenfolge zuzugreifen. Um dies zu verhindern, können Sie die Transaktionen so ändern, dass sie in derselben Reihenfolge auf die Daten zugreifen, wodurch der Zugriff serialisiert wird. Erstellen Sie beispielsweise eine Warteschlange mit Transaktionen, die abgeschlossen werden sollen. Dieser Ansatz kann dazu beitragen, Deadlocks zu vermeiden, wenn mehrere Transaktionen gleichzeitig stattfinden.
4. Durch Hinzufügen sorgfältig ausgewählter Indizes zu Ihren Tabellen lässt sich die Selektivität verbessern und die Notwendigkeit, auf Zeilen zuzugreifen, reduzieren, was zu weniger Sperren führt.
5. Wenn Sie auf eine Lückensperre stoßen, können Sie die Transaktionsisolationsstufe für die Sitzung oder Transaktion in `READ COMMITTED` ändern, um dies zu verhindern. Weitere Informationen zu InnoDB-Isolationsstufen und ihrem Verhalten finden Sie unter Transaktionsisolationsstufen in der MySQL-Dokumentation.

> **Note**
>
> Sie können zwar Vorkehrungen treffen, um die Wahrscheinlichkeit von Deadlocks zu verringern, Deadlocks sind jedoch ein erwartetes Datenbankverhalten und können dennoch auftreten. Anwendungen sollten über die erforderliche Logik zum Umgang mit Deadlocks verfügen, wenn diese auftreten. Implementieren Sie beispielsweise die Wiederholungs- und Back-Off-Logik in der Anwendung. Es ist am besten, die Ursache des Problems zu beheben.

Wenn jedoch ein Deadlock auftritt, hat die Anwendung die Möglichkeit, zu warten und es erneut zu versuchen.

Überwachen von InnoDB-Deadlocks

Deadlocks können in MySQL auftreten, wenn Anwendungstransaktionen versuchen, Sperren auf Tabellen- und Zeilenebene so zu umgehen, dass zirkuläres Warten entsteht. Ein gelegentlicher InnoDB-Deadlock ist nicht unbedingt ein Problem, da die InnoDB-Speicher-Engine den Zustand sofort erkennt und für eine der Transaktionen automatisch ein Rollback durchführt. Wenn Sie häufig auf Deadlocks stoßen, empfehlen wir, Ihre Anwendung zu überprüfen und zu ändern, um Leistungsprobleme zu verringern und Deadlocks zu vermeiden. Wenn die Deadlock-Erkennung aktiviert ist (Standard), erkennt InnoDB automatisch Transaktions-Deadlocks und führt ein Rollback für eine oder mehrere Transaktionen durch, um den Deadlock zu durchbrechen. InnoDB versucht, kleine Transaktionen für das Rollback auszuwählen, wobei die Größe einer Transaktion durch die Anzahl der eingefügten, aktualisierten oder gelöschten Zeilen bestimmt wird.

- `SHOW ENGINE`-Anweisung – Die `SHOW ENGINE INNODB STATUS \G`-Anweisung enthält Details zum letzten Deadlock, der seit dem letzten Neustart in der Datenbank aufgetreten ist.
- MySQL-Fehlerprotokoll – Wenn Sie häufig auf Deadlocks stoßen, bei denen die Ausgabe der `SHOW ENGINE`-Anweisung unangemessen ist, können Sie den DB-Cluster-Parameter innodb_print_all_deadlocks aktivieren.

  Wenn dieser Parameter aktiviert ist, werden Informationen über alle Deadlocks in InnoDB-Benutzertransaktionen im Fehlerprotokoll von Aurora MySQL aufgezeichnet.
- Amazon-CloudWatch-Metriken – Wir empfehlen außerdem, Deadlocks mithilfe der CloudWatch-Metrik `Deadlocks` proaktiv zu überwachen. Weitere Informationen finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.
- Amazon CloudWatch Logs – Mit CloudWatch Logs können Sie Metriken anzeigen, Protokolldaten analysieren und Echtzeitalarme erstellen. Weitere Informationen finden Sie unter Überwachen von Fehlern in Amazon Aurora MySQL und Amazon RDS für MySQL mithilfe von Amazon CloudWatch und Senden von Benachrichtigungen mithilfe von Amazon SNS.

  Wenn CloudWatch Logs mit aktiviertem Parameter `innodb_print_all_deadlocks` verwendet wird, können Sie Alarme so konfigurieren, dass Sie benachrichtigt werden, wenn die Anzahl der Deadlocks einen bestimmten Schwellenwert überschreitet. Wenn Sie einen Schwellenwert

definieren möchten, empfehlen wir Ihnen, Ihre Trends zu beobachten und einen Wert zu verwenden, der auf Ihrer normalen Workload basiert.

- Performance Insights – Wenn Sie Performance Insights verwenden, können Sie die Metriken `innodb_deadlocks` und `innodb_lock_wait_timeout` überwachen. Weitere Informationen zu diesen Metriken, finden Sie unter Nicht-native Zähler für Aurora MySQL.

Bewerten der DB-Instance-Nutzung für Aurora MySQL mit Amazon-CloudWatch-Metriken

Sie können CloudWatch-Metriken verwenden, um Ihren DB-Instance-Durchsatz zu überwachen und festzustellen, ob Ihre DB-Instance-Klasse genügend Ressourcen für Ihre Anwendungen bereitstellt. Weitere Informationen zu Ihren DB-Instance-Klassenlimits finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora. Suchen Sie die Spezifikationen für Ihre DB-Instance-Klasse, um die Netzwerkleistung zu ermitteln.

Wenn sich Ihre DB-Instance-Nutzung dem Instance-Klassenlimit nähert, kann sich die Leistung verschlechtern. Die CloudWatch-Metriken können diese Situation bestätigen, sodass Sie eine manuelle Hochskalierung auf eine größere Instance-Klasse planen können.

Kombinieren Sie die folgenden CloudWatch-Metrikwerte, um herauszufinden, ob Sie sich dem Instance-Klassenlimit nähern:

- NetworkThroughput – Die Menge des von Clients empfangenen und gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster. Dieser Durchsatzwert berücksichtigt nicht den Netzwerkdatenverkehr zwischen den Instances im DB-Cluster und dem Cluster-Volume.
- StorageNetworkThroughput – Die Menge des vom Aurora-Speichersubsystem erhaltenen und gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster.

Fügen Sie StorageNetworkThroughput NetworkThroughput hinzu, um den Netzwerkdurchsatz zu ermitteln, der für jede Instance im Aurora-DB-Cluster vom Aurora-Speichersubsystem erhalten und gesendet wurde. Das Instance-Klassenlimit für Ihre Instance sollte größer sein als die Summe dieser beiden kombinierten Metriken.

Sie können die folgenden Metriken verwenden, um zusätzliche Details des Netzwerkverkehrs Ihrer Client-Anwendungen beim Senden und Empfangen zu überprüfen:

- NetworkReceiveThroughput – Die Menge des von Clients erhaltenen Netzwerkdurchsatzes für jede DB-Instance im DB-Cluster von Aurora MySQL. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances im -DB-Cluster und dem Cluster-Volume.
- NetworkTransmitThroughput – Die Menge des von Clients gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances im -DB-Cluster und dem Cluster-Volume.
- StorageNetworkReceiveThroughput – Die Menge des vom Aurora-Speichersubsystem erhaltenen Netzwerkdurchsatzes für jede Instance im DB-Cluster.
- StorageNetworkTransmitThroughput – Die Menge des an das Aurora-Speichersubsystem gesendeten Netzwerkdurchsatzes für jede Instance im DB-Cluster.

Addieren Sie all diese Metriken, um zu bewerten, wie Ihre Netzwerknutzung im Vergleich zum DB-Instance-Klassenlimit abschneidet. Das Instance-Klassenlimit sollte größer sein als die Summe dieser kombinierten Metriken.

Die Netzwerklimits und die CPU-Auslastung für Speicher stehen in direktem Zusammenhang. Wenn der Netzwerkdurchsatz steigt, steigt auch die CPU-Auslastung. Die Überwachung der CPU- und Netzwerknutzung gibt Aufschluss darüber, wie und warum die Ressourcen erschöpft werden.

Sie können Folgendes in Betracht ziehen, um die Netzwerknutzung zu minimieren:

- Verwenden einer größeren DB-Instance-Klasse
- Aufteilen der Schreibanforderungen in Batches, um die Gesamttransaktionen zu reduzieren
- Weiterleiten der schreibgeschützten Workload an eine schreibgeschützte Instance
- Löschen aller ungenutzten Indizes

Beheben von Problemen mit der Datenbankleistung von Amazon Aurora MySQL

Bei diesem Thema geht es um einige häufig auftretende Leistungsprobleme mit Aurora MySQL DB und darum, wie Sie diese Probleme beheben oder Informationen zur Behebung dieser Probleme erfassen können. Wir unterteilen die Datenbankleistung in zwei große Kategorien:

- Serverleistung – Der gesamte Datenbankserver ist langsamer.
- Abfrageleistung – Die Ausführung einer oder mehrerer Abfragen dauert länger.

AWS-Überwachungsoptionen

Wir empfehlen Ihnen zwecks effizienterer Problembehebung die folgenden AWS-Überwachungsoptionen.

- Amazon CloudWatch – Amazon CloudWatch überwacht Ihre AWS-Ressourcen und die in AWS ausgeführten Anwendungen in Echtzeit. Sie können CloudWatch verwenden, um Metriken zu erfassen und nachzuverfolgen, die Variablen sind, die Sie für Ihre Ressourcen und Anwendungen messen können. Weitere Informationen finden Sie unter Was ist Amazon CloudWatch?.

  Sie können alle Systemmetriken und Prozessinformationen für Ihre DB-Instances in der AWS-Managementkonsole anzeigen. Sie können einen Aurora MySQL-DB-Cluster so konfigurieren, dass allgemeine, Slow-, Audit- und Fehlerprotokolldaten in einer Protokollgruppe in Amazon CloudWatch Logs veröffentlicht werden. Auf diese Weise können Sie Trends einsehen, Protokolle führen, falls ein Host betroffen ist, und eine Ausgangsbasis für eine „normale“ Leistung erstellen, um Anomalien oder Änderungen leichter zu erkennen. Weitere Informationen finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.
- Enhanced Monitoring – Um weitere Amazon-CloudWatch-Metriken für eine Aurora-MySQL-Datenbank zu verwenden, aktivieren Sie Enhanced Monitoring. Wenn Sie einen Aurora-DB-Cluster erstellen oder ändern, wählen Sie Enhanced Monitoring aktivieren aus. Auf diese Weise kann Aurora Leistungsmetriken in CloudWatch veröffentlichen. Zu den wichtigsten verfügbaren Metriken gehören CPU-Auslastung, Datenbankverbindungen, Speichernutzung und Abfragelatenz. Diese können helfen, Leistungsengpässe zu identifizieren.

  Die Menge der für eine DB-Instance übertragenen Informationen ist direkt proportional zur definierten Granularität für Enhanced Monitoring. Ein kürzeres Überwachungsintervall führt zu häufigeren Berichten über Betriebssystem-Metriken und erhöht Ihre Überwachungskosten.

Um Kosten zu verwalten, legen Sie unterschiedliche Granularitäten für verschiedene Instances in Ihren AWS-Konten fest. Die Standardgranularität bei der Erstellung einer Instance beträgt 60 Sekunden. Weitere Informationen finden Sie unter Kosten für „Enhanced Monitoring“ (Erweiterte Überwachung).

- Performance Insights – Sie können alle Metriken für Datenbankaufrufe anzeigen. Dazu gehören DB-Sperren, Wartezeiten und die Anzahl der verarbeiteten Zeilen. All diese Angaben können Sie zur Problembehebung verwenden. Wenn Sie einen Aurora-DB-Cluster erstellen oder ändern, wählen Sie Performance Insights aktivieren aus. Standardmäßig unterstützt Performance Insights einen Datenaufbewahrungszeitraum von 7 Tagen. Dieser kann jedoch angepasst werden, um längerfristige Leistungstrends zu analysieren. Für eine Aufbewahrung von mehr als 7 Tagen müssen Sie zu einer kostenpflichtigen Stufe wechseln. Weitere Informationen finden Sie unter Performance Insights – Preise. Sie können den Datenaufbewahrungszeitraum für jede Aurora-DB-Instance separat festlegen. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .

Die häufigsten Gründe für Leistungsprobleme mit der Aurora-MySQL-Datenbank

Sie können die folgenden Schritte durchführen, um Leistungsprobleme bei Ihrer Aurora-MySQL-Datenbank zu beheben. Wir listen diese Schritte in der logischen Reihenfolge der Untersuchung auf, sie müssen jedoch nicht linear ausgeführt werden. Bei einer Entdeckung kann zwischen den Schritten hin- und hergesprungen werden, was eine Reihe von Ermittlungswegen ermöglicht.

1. Workload – Machen Sie sich mit Ihrem Datenbank-Workload vertraut.
2. Protokollierung – Überprüfen Sie alle Datenbankprotokolle.
3. Datenbankverbindungen – Stellen Sie sicher, dass die Verbindungen zwischen Ihren Anwendungen und Ihrer Datenbank zuverlässig sind.
4. Abfrageleistung – Überprüfen Sie Ihre Pläne zur Abfrageausführung, um festzustellen, ob sie sich geändert haben. Codeänderungen können dazu führen, dass sich Pläne ändern.

Beheben von Workload-Problemen bei Aurora MySQL-Datenbanken

Der Datenbank-Workload lässt sich in Lese- und Schreibvorgänge unterteilen. Wenn Sie sich mit „normalen“ Datenbank-Workloads auskennen, können Sie Abfragen und Datenbankserver an eine sich ändernde Nachfrage anpassen. Es gibt eine Reihe verschiedener Gründe, warum sich die Leistung ändern kann. Der erste Schritt besteht also darin, zu ermitteln, was sich geändert hat.

- Gab es ein Upgrade der Haupt- oder Nebenversion?

  Ein Upgrade der Hauptversion beinhaltet Änderungen am Engine-Code, insbesondere am Optimierer, die den Ausführungsplan der Abfrage ändern können. Bei der Aktualisierung von Datenbankversionen, insbesondere von Hauptversionen, ist es sehr wichtig, dass Sie den Datenbank-Workload analysieren und entsprechend optimieren. Abhängig von den Testergebnissen kann das Verbessern das Optimieren und Neuschreiben von Abfragen oder das Hinzufügen und Aktualisieren von Parametereinstellungen beinhalten. Wenn Sie wissen, was die Auswirkungen verursacht, können Sie sich auf diesen speziellen Bereich konzentrieren.

  Weitere Informationen finden Sie unter Was ist neu in MySQL 8.0 und unter Server- und Statusvariablen sowie hinzugefügte, veraltete oder entfernten Optionen in MySQL 8.0 in der MySQL-Dokumentation und unter Vergleich von Aurora-MySQL-Version 2 und Aurora-MySQL-Version 3.
- Hat die Menge der verarbeiteten Daten zugenommen (Zeilenanzahl)?
- Werden mehr Abfragen gleichzeitig ausgeführt?
- Gibt es Schema- oder Datenbankänderungen?
- Gab es Codefehler oder -korrekturen?

Inhalt

- Instance-Host-Metriken
  - CPU-Verwendung
  - Speicherauslastung
  - Netzwerkdurchsatz
- Datenbankmetriken
- Beheben von Problemen mit der Speichernutzung bei Aurora-MySQL-Datenbanken
  - Beispiel 1: Anhaltend hohe Speichernutzung
  - Beispiel 2: Vorübergehende Speicherspitzen

  - Beispiel 3: Der freie Speicherplatz sinkt kontinuierlich und wird nicht wieder zurückgewonnen
- Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken

Instance-Host-Metriken

Überwachen Sie Metriken des Instance-Hosts wie CPU, Arbeitsspeicher und Netzwerkaktivität, um festzustellen, ob es zu einer Änderung des Workloads gekommen ist. Es gibt zwei Hauptkonzepte für das Ermitteln von Workload-Änderungen:

- Auslastung – Die Nutzung eines Geräts, z. B. einer CPU oder Festplatte. Sie kann zeit- oder kapazitätsbasiert sein.
  - Zeitbasiert – Die Zeit, in der eine Ressource während eines bestimmten Beobachtungszeitraums ausgelastet ist
  - Kapazitätsbasiert – Der Durchsatz, den ein System oder eine Komponente liefern kann, als Prozentsatz von dessen Kapazität.
- Sättigung – Der Grad, in dem von einer Ressource mehr Arbeit verlangt wird, als sie verarbeiten kann Wenn die kapazitätsbasierte Nutzung 100 % erreicht, kann die zusätzliche Last nicht verarbeitet werden und muss in die Warteschlange gestellt werden.

CPU-Verwendung

Mit den folgenden Tools können Sie die CPU-Auslastung und -Sättigung identifizieren:

- CloudWatch stellt die `CPUUtilization`-Metrik bereit. Wenn diese 100 % erreicht, ist die Instance gesättigt. CloudWatch-Metriken werden jedoch über einen Zeitraum von 1 Minute gemittelt und sind nicht sehr detailliert.

  Weitere Informationen zu CloudWatch-Metriken finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.
- Enhanced Monitoring stellt Metriken bereit, die vom Betriebssystembefehl `top`zurückgegeben werden. Er zeigt die durchschnittliche Auslastung und die folgenden CPU-Status mit einer Genauigkeit von 1 Sekunde an:
  - `Idle (%)` = Leerlaufzeit
  - `IRQ (%)` = Softwareunterbrechungen
  - `Nice (%)` = Nice-Zeit für Prozesse mit einer niced-Priorität

- `Steal (%)` = Zeit, die für das Bedienen anderer Mandanten aufgewendet wurde (virtualisierungsbezogen)
- `System (%)` = Systemzeit
- `User (%)` = Benutzerzeit
- `Wait (%)` = I/O-Wartezeit

Weitere Informationen zu Enhanced-Monitoring-Metriken finden Sie unter Betriebssystemmetriken für Aurora.

Speicherauslastung

Wenn das System unter Speicherdruck steht und die Ressourcennutzung die Sättigungsgrenze erreicht, sollten Sie ein hohes Maß an Seitenscan-, Paging-, Swapping- und Speicherfehlern beobachten.

Mit den folgenden Tools können Sie Speicherverbrauch und -sättigung ermitteln:

CloudWatch stellt die `FreeableMemory`-Metrik bereit, die zeigt, wie viel Speicher durch das Leeren einiger Betriebssystem-Caches und des aktuell freien Speichers zurückgewonnen werden kann.

Weitere Informationen zu CloudWatch-Metriken finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

Enhanced Monitoring bietet die folgenden Metriken, anhand derer Sie Probleme mit der Speichernutzung identifizieren können:

- `Buffers (KB)` – Umfang des verwendeten Arbeitsspeichers für die Pufferung von I/O-Anfragen vor dem Schreiben auf das Speichergerät, in Kilobyte
- `Cached (KB)` – Umfang des verwendeten Arbeitsspeichers für das Caching von Dateisystem-basierter I/O
- `Free (KB)` – Umfang des nicht zugewiesenen Arbeitsspeichers in Kilobyte
- `Swap` – Zwischengespeichert, Frei und Insgesamt

Wenn Sie beispielsweise feststellen, dass Ihre DB-Instance `Swap`-Speicher verbraucht, ist der Gesamtspeicher, den Ihr Workload benötigt, größer als die Menge, die Ihre Instance aktuell zur Verfügung hat. Wir empfehlen, die Größe Ihrer DB-Instance zu erhöhen oder Ihren Workload so zu optimieren, dass weniger Speicher verbraucht wird.

Weitere Informationen zu Metriken für erweiterte Überwachung finden Sie unter Betriebssystemmetriken für Aurora.

Ausführlichere Informationen zur Verwendung des Leistungsschemas und des `sys`-Schemas zur Bestimmung der Verbindungen und Komponenten, die Speicher verbrauchen, finden Sie unter Beheben von Problemen mit der Speichernutzung bei Aurora-MySQL-Datenbanken.

Netzwerkdurchsatz

CloudWatch bietet die folgenden Metriken für den Netzwerkdurchsatz insgesamt, alle gemittelt über 1 Minute:

- `NetworkReceiveThroughput` – Der Umfang des von Clients erhaltenen Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster
- `NetworkTransmitThroughput` – Der Umfang des von Clients gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster
- `NetworkThroughput` – Die Menge des von Clients empfangenen und gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster
- `StorageNetworkReceiveThroughput` – Der Umfang des vom Aurora-Speicheruntersystem empfangenen Netzwerkdurchsatzes für jede Instance im DB-Cluster
- `StorageNetworkTransmitThroughput` – Der Umfang des an das Aurora-Speicheruntersystem gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster.
- `StorageNetworkThroughput` – Der Umfang des vom Aurora-Speicheruntersystem erhaltenen und an dieses gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster

Weitere Informationen zu CloudWatch-Metriken finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

Enhanced Monitoring stellt die von `network` empfangenen (RX)- und übertragenen (TX)-Diagramme mit einer Genauigkeit von bis zu 1 Sekunde bereit.

Weitere Informationen zu Metriken für erweiterte Überwachung finden Sie unter Betriebssystemmetriken für Aurora.

Datenbankmetriken

Untersuchen Sie die folgenden CloudWatch-Metriken auf Workload-Änderungen:

- `BlockedTransactions` – Die durchschnittliche Anzahl an gesperrten Transaktionen in der Datenbank pro Sekunde
- `BufferCacheHitRatio` – Der Prozentsatz der vom Buffer-Cache bedienten Anfragen
- `CommitThroughput` – Die durchschnittliche Anzahl der Commit-Operationen pro Sekunde
- `DatabaseConnections` – Die Anzahl der Clientnetzwerkverbindungen zur Datenbank-Instance
- `Deadlocks` – Die durchschnittliche Anzahl an Deadlocks in der Datenbank pro Sekunde
- `DMLThroughput` – Die durchschnittliche Anzahl von Einfügungen, Updates und Löschungen pro Sekunde
- `ResultSetCacheHitRatio` – Der Prozentsatz der vom Buffer-Cache bedienten Anfragen
- `RollbackSegmentHistoryListLength` – Die Protokolle für das Rückgängigmachen, die festgeschriebene Transaktionen mit zum Löschen markierten Datensätzen aufzeichnen
- `RowLockTime` – Die Gesamtzeit für das Erfassen von Zeilensperren für InnoDB-Tabellen
- `SelectThroughput` – Die durchschnittliche Anzahl von Auswahlabfragen pro Sekunde

Weitere Informationen zu CloudWatch-Metriken finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

Berücksichtigen Sie bei der Workload-Untersuchung die folgenden Fragen:

1. Gab es kürzlich Änderungen an der DB-Instance-Klasse, z. B. eine Reduzierung der Instance-Größe von 8xlarge auf 4xlarge oder eine Änderung von db.r5 in db.r6?
2. Können Sie einen Klon erstellen und das Problem reproduzieren, oder tritt es nur auf dieser einen Instance auf?
3. Sind die Serverressourcen erschöpft oder gibt es eine hohe CPU- oder Speicherauslastung? Falls ja, könnte dies bedeuten, dass zusätzliche Hardware erforderlich ist.
4. Dauern eine oder mehrere Abfragen länger?
5. Werden die Änderungen durch ein Upgrade verursacht, insbesondere durch ein Upgrade einer Hauptversion? Falls ja, vergleichen Sie die Metriken vor und nach dem Upgrade.
6. Gibt es Änderungen bei der Anzahl der Reader-DB-Instances?
7. Haben Sie die allgemeine Protokollierung, die Auditprotokollierung oder die binäre Protokollierung aktiviert? Weitere Informationen finden Sie unter Protokollieren bei Aurora-MySQL-Datenbanken.
8. Haben Sie die Verwendung der Binärprotokoll-Replikation (Binlog) aktiviert, deaktiviert oder geändert?

9. Gibt es Transaktionen mit langer Laufzeit, die eine große Anzahl von Zeilensperren enthalten? Untersuchen Sie die Länge der InnoDB-Verlaufsliste (HLL) auf Hinweise bezüglich lang andauernder Transaktionen.

   Weitere Informationen finden Sie unter Die Länge der InnoDB-Verlaufsliste wurde deutlich erhöht und im Blogbeitrag Warum wird meine SELECT-Abfrage auf meinem DB-Cluster von Amazon Aurora MySQL so langsam ausgeführt?.

   a. Wenn eine große HLL durch eine Schreibtransaktion verursacht wird, bedeutet dies, dass sich UNDO-Protokolle ansammeln (nicht regelmäßig bereinigt werden). Bei einer großen Schreibtransaktion kann diese Anhäufung anwachsen. In MySQL wird UNDO im SYSTEM-Tablespace gespeichert. Der SYSTEM-Tablespace kann nicht verkleinert werden. Das UNDO-Protokoll kann dazu führen, dass der SYSTEM-Tablespace auf mehrere GB oder sogar TB anwächst. Geben Sie nach dem Löschen den zugewiesenen Speicherplatz frei, indem Sie ein logisches Backup (Dump) der Daten erstellen und diesen Dump anschließend in eine neue DB-Instance importieren.

   b. Wenn eine große HLL durch eine Lesetransaktion (lang andauernde Abfrage) verursacht wird, kann dies bedeuten, dass die Abfrage eine große Menge an temporärem Speicherplatz belegt. Geben Sie den temporären Speicherplatz durch einen Neustart frei. Untersuchen Sie die DB-Metriken von Performance Insights auf Änderungen im Temp-Abschnitt, beispielsweise `created_tmp_tables`. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .

10. Können Sie Transaktionen mit langer Laufzeit in kleinere Transaktionen aufteilen, bei denen weniger Zeilen geändert werden?

11. Gibt es Änderungen bei blockierten Transaktionen oder eine Zunahme von Deadlocks? Untersuchen Sie die DB-Metriken von Performance Insights auf Änderungen bei den Statusvariablen im Locks-Abschnitt, beispielsweise `innodb_row_lock_time`, `innodb_row_lock_waits` und `innodb_dead_locks`. Verwenden Sie Intervalle von 1 Minute oder 5 Minuten.

12. Gibt es einen Anstieg bei den Warteereignissen? Untersuchen Sie die Warteereignisse und Wartetypen von Performance Insights anhand von 1- oder 5-minütigen Intervallen. Analysieren Sie die wichtigsten Warteereignisse und finden Sie heraus, ob sie mit Workload-Änderungen oder Datenbankkonflikten korrelieren. `buf_pool mutex` weist beispielsweise auf einen Pufferpoolkonflikt hin. Weitere Informationen finden Sie unter Optimieren von Aurora MySQL mit Warteereignissen.

Beheben von Problemen mit der Speichernutzung bei Aurora-MySQL-Datenbanken

CloudWatch, Enhanced Monitoring und Performance Insights bieten zwar einen guten Überblick über die Speichernutzung auf Betriebssystemebene, z. B. darüber, wie viel Speicher der Datenbankprozess verbraucht, allerdings können Sie nicht aufschlüsseln, welche Verbindungen oder Komponenten innerhalb der Engine diesen Speichenverbrauch verursachen.

Zur Behebung können Sie das Leistungsschema und das `sys`-Schema verwenden. In Version 3 von Aurora MySQL ist die Speicherinstrumentierung standardmäßig aktiviert, wenn das Leistungsschema aktiviert ist. In Version 2 von Aurora MySQL ist standardmäßig nur die Speicherinstrumentierung für die Speichernutzung des Leistungsschemas aktiviert. Informationen zu Tabellen, die im Leistungsschema verfügbar sind, um die Speichernutzung nachzuverfolgen und die Speicherinstrumentierung des Leistungsschemas zu aktivieren, finden Sie unter Speicherzusammenfassungstabellen in der MySQL-Dokumentation. Weitere Informationen über die Verwendung des Leistungsschemas mit Performance Insights finden Sie unter Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL.

Das Leistungsschema enthält detaillierte Informationen zur Nachverfolgung der aktuellen Speichernutzung, das sys-Schema von MySQL dagegen bietet zusätzlich zu den Leistungsschematabellen Ansichten, anhand derer Sie schnell feststellen können, wo Speicher genutzt wird.

Im `sys`-Schema sind die folgenden Ansichten verfügbar, mit denen die Speichernutzung nach Verbindung, Komponente und Abfrage nachverfolgt werden kann.

Anzeigen	Beschreibung
memory_by_host_by_current_bytes	Diese Ansicht stellt Informationen zur Engine-Speichernutzung nach Host bereit. Dies kann nützlich sein, um festzustellen, welche Anwendungsserver oder Client-Hosts Speicher verbrauchen.
memory_by_thread_by_current_bytes	Diese Ansicht stellt Informationen zur Engine-Speichernutzung nach Thread-ID bereit. Die Thread-ID in MySQL kann eine Client-Verbindung oder ein Hintergrund-Thread sein. Sie können Thread-IDs MySQL-Verbindungs-IDs

Anzeigen	Beschreibung
	zuordnen, indem Sie die Ansicht sys.processlist oder die Tabelle performance_schema.threads verwenden.
memory_by_user_by_current_bytes	Diese Ansicht stellt Informationen zur Engine-Speichernutzung nach Benutzer bereit. Dies kann nützlich sein, um festzustellen, welche Benutzerkonten oder Clients Speicher verbrauchen.
memory_global_by_current_bytes	Diese Ansicht stellt Informationen zur Engine-Speichernutzung nach Engine-Komponente bereit. Dies kann nützlich sein, um die Speichernutzung global nach Engine-Puffer oder Komponente zu ermitteln. So werden möglicherweise das `memory/innodb/buf_buf_pool` -Ereignis für den InnoDB-Pufferpool oder das `memory/sql/Prepared_statement::main_mem_root` -Ereignis für vorbereitete Anweisungen angezeigt.
memory_global_total	Diese Ansicht bietet einen Überblick über die insgesamt nachverfolgte Speichernutzung in der Engine der Datenbank.

In den Versionen 3.05 und höher von Aurora MySQL können Sie die maximale Speichernutzung auch nach Anweisungs-Digest in den Übersichtstabellen der Leistungsschemaanweisungen nachverfolgen. Die Übersichtstabellen der Anweisungen enthalten normalisierte Anweisungs-Digests und aggregierte Statistiken zu deren Ausführung. Anhand der `MAX_TOTAL_MEMORY`-Spalte können Sie ermitteln, wie viel Speicher je nach Abfrage-Digest seit dem letzten Zurücksetzen der Statistiken oder seit dem Neustart der Datenbank-Instance belegt wurde. Dies kann nützlich sein, um bestimmte Abfragen zu identifizieren, die möglicherweise viel Speicher verbrauchen.

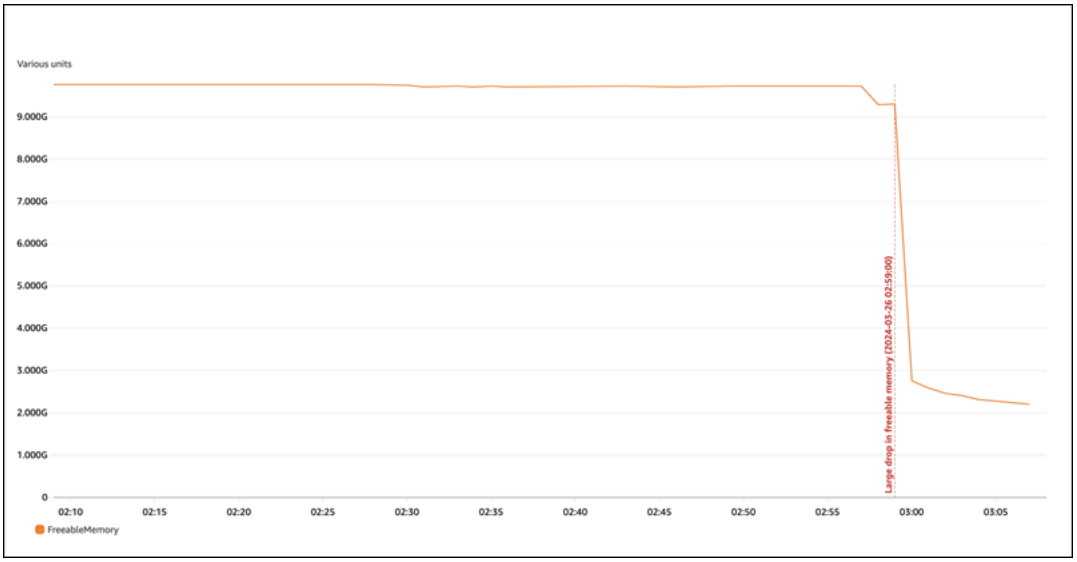
> **Note**
>
> Das Leistungsschema und das `sys`-Schema zeigen Ihnen die aktuelle Speichernutzung auf dem Server sowie die Höchstwerte für den Speicherverbrauch pro Verbindung und Engine-Komponente. Da das Leistungsschema im Arbeitsspeicher gespeichert wird, werden die Informationen zurückgesetzt, wenn die DB-Instance neu gestartet wird. Zum Verwalten eines chronologischen Verlaufs empfehlen wir, den Abruf und die Speicherung dieser Daten außerhalb des Leistungsschemas zu konfigurieren.

Themen

- Beispiel 1: Anhaltend hohe Speichernutzung
- Beispiel 2: Vorübergehende Speicherspitzen
- Beispiel 3: Der freie Speicherplatz sinkt kontinuierlich und wird nicht wieder zurückgewonnen

Beispiel 1: Anhaltend hohe Speichernutzung

Wenn wir uns `FreeableMemory` in CloudWatch näher ansehen, stellen wir fest, dass die Speichernutzung am 26.03.2024 um 02:59 UTC stark zugenommen hat.



Dies vermittelt uns kein vollständiges Bild. Sie können sich bei der Datenbank anmelden und sich `sys.memory_global_by_current_bytes` anzeigen lassen, um festzustellen, welche Komponente den meisten Speicher beansprucht. Diese Tabelle enthält eine Liste von Speicherereignissen, die MySQL nachverfolgt, zusammen mit Informationen zur Speicherzuweisung

pro Ereignis. Jedes Speicherverfolgungsereignis beginnt mit `memory/%`, gefolgt von weiteren Informationen darüber, mit welcher Engine-Komponente oder -Funktion das Ereignis verknüpft ist.

`memory/performance_schema/%` gilt beispielsweise für Speicherereignisse, die sich auf das Leistungsschema beziehen, `memory/innodb/%` für InnoDB und so weiter. Informationen zu den Benennungskonventionen für Ereignisse finden Sie unter Leistungsschema – Instrument-Benennungskonventionen in der MySQL-Dokumentation.

Anhand der folgenden Abfrage können wir basierend auf `current_alloc` den wahrscheinlichen Verursacher ermitteln, sehen aber auch viele `memory/performance_schema/%`-Ereignisse.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/sql/Prepared_statement::main_mem_root                                  |
 512817 | 4.91 GiB      | 10.04 KiB         |     512823 | 4.91 GiB   | 10.04 KiB
 |
| memory/performance_schema/prepared_statements_instances                       |
   252 | 488.25 MiB    | 1.94 MiB          |        252 | 488.25 MiB | 1.94 MiB       |
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error            |
  1028 | 52.27 MiB     | 52.06 KiB         |       1028 | 52.27 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     4 | 47.25 MiB     | 11.81 MiB         |          4 | 47.25 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/performance_schema/memory_summary_by_thread_by_event_name              |
     4 | 31.64 MiB     | 7.91 MiB          |          4 | 31.64 MiB  | 7.91 MiB       |
| memory/innodb/memory                                                          |
 15227 | 27.44 MiB     | 1.85 KiB          |      20619 | 33.33 MiB  | 1.66 KiB       |
| memory/sql/String::value                                                      |
 74411 | 21.85 MiB     |  307 bytes        |      76867 | 25.54 MiB  |  348 bytes     |
| memory/sql/TABLE                                                              |
  8381 | 21.03 MiB     | 2.57 KiB          |       8381 | 21.03 MiB  | 2.57 KiB       |
```

```
+-----------------------------------------------------------------------------
+--------------+--------------+------------------+-----------+-----------
+----------------+
10 rows in set (0.02 sec)
```

Wir haben bereits erwähnt, dass das Leistungsschema im Arbeitsspeicher gespeichert wird, was bedeutet, dass es auch in der `performance_schema`-Speicherinstrumentierung nachverfolgt wird.

> **Note**
>
> Sie können Datenbankparameter an Ihre Anforderungen anpassen, wenn Sie feststellen, dass das Leistungsschema viel Speicher beansprucht und Sie dessen Speichernutzung einschränken möchten. Weitere Informationen finden Sie unter Das Speicherzuweisungsmodell des Leistungsschemas in der MySQL-Dokumentation.

Zum Zwecke einer besseren Lesbarkeit können Sie dieselbe Abfrage erneut ausführen, dabei aber die Ereignisse des Leistungsschemas ausschließen. Die Ausgabe zeigt:

- `memory/sql/Prepared_statement::main_mem_root` verbraucht am meisten Speicher.
- Aus der `current_alloc`-Spalte geht hervor, dass MySQL diesem Ereignis derzeit 4,91 GiB zugewiesen hat.
- `high_alloc column` gibt an, dass 4,91 GiB den Höchststand von `current_alloc` seit dem letzten Zurücksetzen oder dem Neustart des Servers markieren. Das bedeutet, dass `memory/sql/Prepared_statement::main_mem_root` seinen höchsten Wert erreicht hat.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name NOT LIKE
 'memory/performance_schema/%' LIMIT 10;

+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root  |        512817 | 4.91 GiB      | 10.04
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
| memory/innodb/hash0hash                       |             4 | 79.07 MiB     | 19.77
 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
```

```
| memory/innodb/memory                              |         17096 | 31.68 MiB     | 1.90
 KiB          |      22498 | 37.60 MiB  | 1.71 KiB       |
| memory/sql/String::value                          |        122277 | 27.94 MiB     |  239
 bytes        |     124699 | 29.47 MiB  |  247 bytes     |
| memory/sql/TABLE                                  |          9927 | 24.67 MiB     | 2.55
 KiB          |       9929 | 24.68 MiB  | 2.55 KiB       |
| memory/innodb/lock0lock                           |          8888 | 19.71 MiB     | 2.27
 KiB          |       8888 | 19.71 MiB  | 2.27 KiB       |
| memory/sql/Prepared_statement::infrastructure |        257623 | 16.24 MiB     |   66
 bytes        |     257631 | 16.24 MiB  |   66 bytes     |
| memory/mysys/KEY_CACHE                            |             3 | 16.00 MiB     | 5.33
 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/innodb/sync0arr                            |             3 | 7.03 MiB      | 2.34
 MiB          |          3 | 7.03 MiB   | 2.34 MiB       |
| memory/sql/THD::main_mem_root                     |           815 | 6.56 MiB      | 8.24
 KiB          |        849 | 7.19 MiB   | 8.67 KiB       |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
10 rows in set (0.06 sec)
```

Am Namen des Ereignisses können wir erkennen, dass dieser Speicher für vorbereitete Anweisungen verwendet wird. Wenn Sie erfahren möchten, welche Verbindungen diesen Speicher verwenden, überprüfen Sie memory_by_thread_by_current_bytes.

Im folgenden Beispiel sind jeder Verbindung ungefähr 7 MiB zugewiesen, der Höchststand beträgt ca. 6,29 MiB (`current_max_alloc`). Dies ergibt Sinn, da in diesem Beispiel `sysbench` mit 80 Tabellen und 800 Verbindungen mit vorbereiteten Anweisungen verwendet wird. Wenn Sie in diesem Szenario die Speichernutzung reduzieren möchten, können Sie die Verwendung von vorbereiteten Anweisungen in Ihrer Anwendung optimieren, um den Speicherverbrauch zu senken.

```
mysql> SELECT * FROM sys.memory_by_thread_by_current_bytes;

+-----------+-----------------------------------------------+-------------------
+------------------+------------------+------------------+-----------------+
| thread_id | user                                          | current_count_used |
 current_allocated | current_avg_alloc | current_max_alloc | total_allocated |
+-----------+-----------------------------------------------+-------------------
+------------------+------------------+------------------+-----------------+
|        46 | rdsadmin@localhost                          |                405 | 8.47 MiB
          | 21.42 KiB         | 8.00 MiB          | 155.86 MiB      |
|        61 | reinvent@10.0.4.4                           |               1749 | 6.72 MiB
          | 3.93 KiB          | 6.29 MiB          | 14.24 MiB       |
```

```
|       101 | reinvent@10.0.4.4                        |              1845 | 6.71 MiB
          | 3.72 KiB          | 6.29 MiB          | 14.50 MiB       |
|        55 | reinvent@10.0.4.4                        |              1674 | 6.68 MiB
          | 4.09 KiB          | 6.29 MiB          | 14.13 MiB       |
|        57 | reinvent@10.0.4.4                        |              1416 | 6.66 MiB
          | 4.82 KiB          | 6.29 MiB          | 13.52 MiB       |
|       112 | reinvent@10.0.4.4                        |              1759 | 6.66 MiB
          | 3.88 KiB          | 6.29 MiB          | 14.17 MiB       |
|        66 | reinvent@10.0.4.4                        |              1428 | 6.64 MiB
          | 4.76 KiB          | 6.29 MiB          | 13.47 MiB       |
|        75 | reinvent@10.0.4.4                        |              1389 | 6.62 MiB
          | 4.88 KiB          | 6.29 MiB          | 13.40 MiB       |
|       116 | reinvent@10.0.4.4                        |              1333 | 6.61 MiB
          | 5.08 KiB          | 6.29 MiB          | 13.21 MiB       |
|        90 | reinvent@10.0.4.4                        |              1448 | 6.59 MiB
          | 4.66 KiB          | 6.29 MiB          | 13.58 MiB       |
|        98 | reinvent@10.0.4.4                        |              1440 | 6.57 MiB
          | 4.67 KiB          | 6.29 MiB          | 13.52 MiB       |
|        94 | reinvent@10.0.4.4                        |              1433 | 6.57 MiB
          | 4.69 KiB          | 6.29 MiB          | 13.49 MiB       |
|        62 | reinvent@10.0.4.4                        |              1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.48 MiB       |
|        87 | reinvent@10.0.4.4                        |              1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.25 MiB       |
|        99 | reinvent@10.0.4.4                        |              1346 | 6.54 MiB
          | 4.98 KiB          | 6.29 MiB          | 13.24 MiB       |
|       105 | reinvent@10.0.4.4                        |              1347 | 6.54 MiB
          | 4.97 KiB          | 6.29 MiB          | 13.34 MiB       |
|        73 | reinvent@10.0.4.4                        |              1335 | 6.54 MiB
          | 5.02 KiB          | 6.29 MiB          | 13.23 MiB       |
|        54 | reinvent@10.0.4.4                        |              1510 | 6.53 MiB
          | 4.43 KiB          | 6.29 MiB          | 13.49 MiB       |
.                                                                    .
.                                                                    .
.                                                                    .
|       812 | reinvent@10.0.4.4                        |              1259 | 6.38 MiB
          | 5.19 KiB          | 6.29 MiB          | 13.05 MiB       |
|       214 | reinvent@10.0.4.4                        |              1279 | 6.38 MiB
          | 5.10 KiB          | 6.29 MiB          | 12.90 MiB       |
|       325 | reinvent@10.0.4.4                        |              1254 | 6.38 MiB
          | 5.21 KiB          | 6.29 MiB          | 12.99 MiB       |
```

```
|       705 | reinvent@10.0.4.4                          |              1273 | 6.37 MiB
          | 5.13 KiB          | 6.29 MiB          | 13.03 MiB       |
|       530 | reinvent@10.0.4.4                          |              1268 | 6.37 MiB
          | 5.15 KiB          | 6.29 MiB          | 12.92 MiB       |
|       307 | reinvent@10.0.4.4                          |              1263 | 6.37 MiB
          | 5.17 KiB          | 6.29 MiB          | 12.87 MiB       |
|       738 | reinvent@10.0.4.4                          |              1260 | 6.37 MiB
          | 5.18 KiB          | 6.29 MiB          | 13.00 MiB       |
|       819 | reinvent@10.0.4.4                          |              1252 | 6.37 MiB
          | 5.21 KiB          | 6.29 MiB          | 13.01 MiB       |
|        31 | innodb/srv_purge_thread                    |             17810 | 3.14 MiB
          |  184 bytes        | 2.40 MiB          | 205.69 MiB      |
|        38 | rdsadmin@localhost                         |               599 | 1.76 MiB
          | 3.01 KiB          | 1.00 MiB          | 25.58 MiB       |
|         1 | sql/main                                   |              3756 | 1.32 MiB
          |  367 bytes        | 355.78 KiB        | 6.19 MiB        |
|       854 | rdsadmin@localhost                         |                46 | 1.08 MiB
          | 23.98 KiB         | 1.00 MiB          | 5.10 MiB        |
|        30 | innodb/clone_gtid_thread                   |              1596 | 573.14
 KiB        |  367 bytes        | 254.91 KiB        | 970.69 KiB      |
|        40 | rdsadmin@localhost                         |               235 | 245.19
 KiB        | 1.04 KiB          | 128.88 KiB        | 808.64 KiB      |
|       853 | rdsadmin@localhost                         |                96 | 94.63
 KiB        | 1009 bytes        | 29.73 KiB         | 422.45 KiB      |
|        36 | rdsadmin@localhost                         |                33 | 36.29
 KiB        | 1.10 KiB          | 16.08 KiB         | 74.15 MiB       |
|        33 | sql/event_scheduler                        |                 3 | 16.27
 KiB        | 5.42 KiB          | 16.04 KiB         | 16.27 KiB       |
|        35 | sql/compress_gtid_table                    |                 8 | 14.20
 KiB        | 1.77 KiB          | 8.05 KiB          | 18.62 KiB       |
|        25 | innodb/fts_optimize_thread                 |                12 | 1.86 KiB
          |  158 bytes        |  648 bytes        | 1.98 KiB        |
|        23 | innodb/srv_master_thread                   |                11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 24.40 KiB       |
|        24 | innodb/dict_stats_thread                   |                11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 1.35 KiB        |
|         5 | innodb/io_read_thread                      |                 1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         6 | innodb/io_read_thread                      |                 1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         2 | sql/aws_oscar_log_level_monitor            |                 0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         4 | innodb/io_ibuf_thread                      |                 0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
```

```
|         7 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         8 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         9 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        10 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        11 | innodb/srv_lra_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        12 | innodb/srv_akp_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        18 | innodb/srv_lock_timeout_thread             |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |  248 bytes      |
|        19 | innodb/srv_error_monitor_thread            |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        20 | innodb/srv_monitor_thread                  |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        21 | innodb/buf_resize_thread                   |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        22 | innodb/btr_search_sys_toggle_thread        |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        32 | innodb/dict_persist_metadata_table_thread  |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        34 | sql/signal_handler                         |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
+-----------+--------------------------------------------+--------------------
+-------------------+------------------+------------------+-----------------+
831 rows in set (2.48 sec)
```

Wie bereits erwähnt, kann sich der Wert der Thread-ID (`thd_id`) hier auf Serverhintergrund-Threads oder Datenbankverbindungen beziehen. Wenn Sie Thread-ID-Werte Datenbankverbindungs-IDs zuordnen möchten, können Sie die `performance_schema.threads`-Tabelle oder die `sys.processlist`-Ansicht verwenden, wobei `conn_id` die Verbindungs-ID darstellt.

```
mysql> SELECT thd_id,conn_id,user,db,command,state,time,last_wait FROM sys.processlist
 WHERE user='reinvent@10.0.4.4';

+--------+---------+------------------+----------+---------+----------------+------
+-----------------------------------------------+
| thd_id | conn_id | user             | db       | command | state          | time |
 last_wait                                      |
```

```
+--------+---------+-------------------+----------+---------+----------------+------
+-----------------------------------------------+
|    590 |     562 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    578 |     550 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    579 |     551 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    580 |     552 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    581 |     553 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    582 |     554 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    583 |     555 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    584 |     556 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    585 |     557 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    586 |     558 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    587 |     559 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
.
                                                 .
.
                                                 .
.
                                                 .
|    323 |     295 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    324 |     296 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    325 |     297 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    326 |     298 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    438 |     410 | reinvent@10.0.4.4 | sysbench | Execute | System lock    |    0 |
 wait/lock/table/sql/handler                     |
|    280 |     252 | reinvent@10.0.4.4 | sysbench | Sleep   | starting       |    0 |
 wait/io/socket/sql/client_connection            |
|     98 |      70 | reinvent@10.0.4.4 | sysbench | Query   | freeing items  |    0 |
 NULL                                            |
```

```
+--------+---------+-------------------+----------+---------+---------------+------
+-----------------------------------------------+
804 rows in set (5.51 sec)
```

Wir beenden jetzt den `sysbench`-Workload, wodurch die Verbindungen geschlossen und der Speicher freigegeben wird. Wenn wir die Ereignisse erneut überprüfen, können wir bestätigen, dass Speicher freigegeben wurde, aber `high_alloc` lässt uns nach wie vor wissen, was der Höchststand ist. Die `high_alloc`-Spalte kann sehr nützlich sein, um kurze Spitzen bei der Speichernutzung zu identifizieren, die Sie möglicherweise nicht sofort anhand von `current_alloc` ermitteln können, da diese Spalte nur den derzeit zugewiesenen Speicher anzeigt.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+--------------
+------------------+------------+------------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+--------------
+------------------+------------+------------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
+-----------------------------------------------+---------------+--------------
+------------------+------------+------------+----------------+
1 row in set (0.00 sec)
```

Wenn Sie `high_alloc` zurücksetzen möchten, können Sie die `performance_schema`-Speicherübersichtstabellen kürzen. Dadurch wird jedoch die gesamte Speicherinstrumentierung zurückgesetzt. Weitere Informationen finden Sie unter Leistungsschema – allgemeine Tabellenmerkmale in der MySQL-Dokumentation.

Im folgenden Beispiel sehen wir, dass `high_alloc` nach der Kürzung zurückgesetzt wird.

```
mysql> TRUNCATE `performance_schema`.`memory_summary_global_by_event_name`;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+--------------
+------------------+------------+------------+----------------+
```

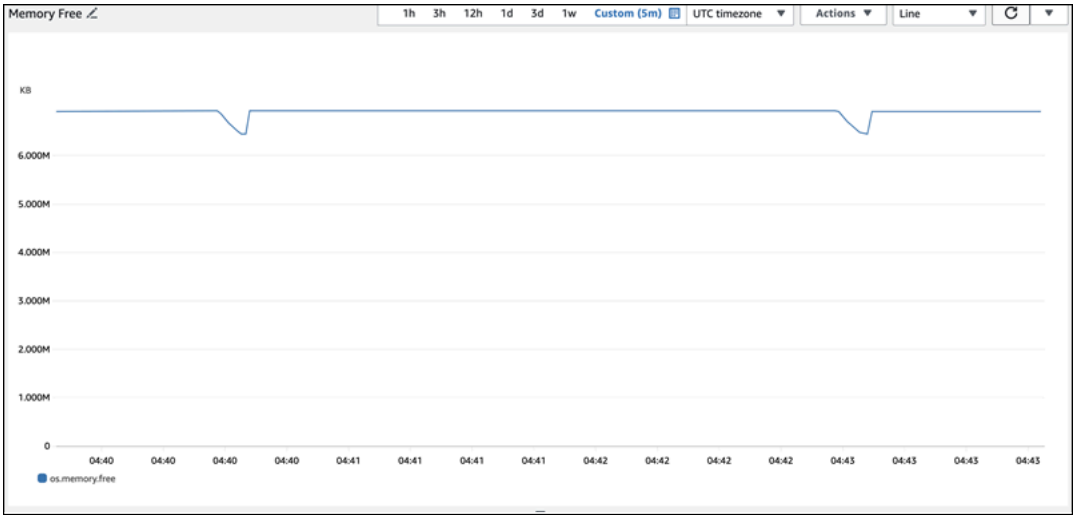
```
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+------------------+------------+------------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |         17 | 253.80 KiB | 14.93 KiB      |
+-----------------------------------------------+---------------+---------------
+------------------+------------+------------+----------------+
1 row in set (0.00 sec)
```

Beispiel 2: Vorübergehende Speicherspitzen

Ein weiteres häufiges Ereignis sind kurze Spitzen bei der Speichernutzung auf einem Datenbankserver. Dabei kann es sich um periodische Einbrüche beim freigebbaren Speicher handeln, die durch `current_alloc` in `sys.memory_global_by_current_bytes` nur schwer zu beheben sind, da der Speicher bereits freigegeben wurde.

> **Note**
>
> Wurden die Statistiken des Leistungsschemas zurückgesetzt oder wurde die Instance der Datenbank neu gestartet, sind diese Informationen im `sys` oder `erformance_schema` nicht verfügbar. Zur Beibehaltung dieser Informationen sollten Sie die Erfassung externer Metriken konfigurieren.

Das folgende Diagramm der `os.memory.free`-Metrik in Enhanced Monitoring zeigt kurze Spitzen bei der Speichernutzung von 7 Sekunden. Enhanced Monitoring ermöglicht Ihnen eine Überwachung in Intervallen von nur 1 Sekunde. Dies ist ideal, um vorübergehende Spitzen wie diese abzufangen.



Um hier die Ursache der Speichernutzung zu ermitteln, können wir eine Kombination aus `high_alloc` in den `sys`-Speicherübersichtsansichten und den Übersichtstabellen der Leistungsschemaanweisungen verwenden, um zu versuchen, problematische Sitzungen und Verbindungen zu identifizieren.

Da der Speicherverbrauch derzeit nicht hoch ist, können wir in der `sys`-Schemaansicht unter `current_alloc` erwartungsgemäß keine größeren Problemverursacher erkennen.

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/os0event                                                        |
 439372 | 60.34 MiB     |  144 bytes        |     439372 | 60.34 MiB  |  144 bytes
 |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/mysys/KEY_CACHE                                                        |
     3 | 16.00 MiB     | 5.33 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/performance_schema/events_statements_history_long                      |
     1 | 14.34 MiB     | 14.34 MiB         |          1 | 14.34 MiB  | 14.34 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error            |
   257 | 13.07 MiB     | 52.06 KiB         |        257 | 13.07 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     1 | 11.81 MiB     | 11.81 MiB         |          1 | 11.81 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest.digest_text    |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.digest_text         |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.sql_text            |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
```

```
10 rows in set (0.01 sec)
```

Wenn wir die Ansicht erweitern, um nach `high_alloc` zu sortieren, sehen wir, dass die `memory/temptable/physical_ram`-Komponente hier ein möglicher Kandidat ist. Deren höchster Verbrauch lag bei 515,00 MiB.

Wie der Name schon sagt, instrumentiert `memory/temptable/physical_ram` die Speichernutzung für die `TEMP`-Speicher-Engine in MySQL, die in MySQL 8.0 eingeführt wurde. Weitere Informationen darüber, wie MySQL temporäre Tabellen verwendet, finden Sie unter Verwenden von internen temporären Tabellen in MySQL in der MySQL-Dokumentation.

> **Note**
>
> In diesem Beispiel verwenden wir die `sys.x$memory_global_by_current_bytes`-Ansicht.

```
mysql> SELECT event_name, format_bytes(current_alloc) AS "currently allocated",
 sys.format_bytes(high_alloc) AS "high-water mark"
FROM sys.x$memory_global_by_current_bytes ORDER BY high_alloc DESC LIMIT 10;

+-------------------------------------------------------------------------------
+---------------------+-----------------+
| event_name                                                                    |
 currently allocated | high-water mark |
+-------------------------------------------------------------------------------
+---------------------+-----------------+
| memory/temptable/physical_ram                                                 | 4.00
 MiB             | 515.00 MiB      |
| memory/innodb/hash0hash                                                       | 79.07
 MiB            | 79.07 MiB       |
| memory/innodb/os0event                                                        | 63.95
 MiB            | 63.95 MiB       |
| memory/performance_schema/events_statements_summary_by_digest                | 40.28
 MiB            | 40.28 MiB       |
| memory/mysys/KEY_CACHE                                                        | 16.00
 MiB            | 16.00 MiB       |
| memory/performance_schema/events_statements_history_long                      | 14.34
 MiB            | 14.34 MiB       |
| memory/performance_schema/events_errors_summary_by_thread_by_error           | 13.07
 MiB            | 13.07 MiB       |
```

```
| memory/performance_schema/events_statements_summary_by_thread_by_event_name | 11.81
 MiB           | 11.81 MiB       |
| memory/performance_schema/events_statements_summary_by_digest.digest_text   | 9.77
 MiB            | 9.77 MiB        |
| memory/performance_schema/events_statements_history_long.sql_text           | 9.77
 MiB            | 9.77 MiB        |
+------------------------------------------------------------------------------
+--------------------+----------------+
10 rows in set (0.00 sec)
```

In Beispiel 1: Anhaltend hohe Speichernutzung haben wir die aktuelle Speichernutzung jeder Verbindung überprüft, um festzustellen, welche Verbindung für die Nutzung des fraglichen Speichers verantwortlich ist. In diesem Beispiel ist der Speicher bereits freigegeben, sodass es nicht hilfreich ist, die Speichernutzung für aktuelle Verbindungen zu überprüfen.

Um tiefer in die Materie einzutauchen und die problematischen Anweisungen, Benutzer und Hosts zu ermitteln, verwenden wir das Leistungsschema. Das Leistungsschema enthält mehrere Übersichtstabellen zu Anweisungen, die in verschiedene Dimensionen wie Ereignisname, Anweisungs-Digest, Host, Thread und Benutzer unterteilt sind. Jede Ansicht ermöglicht es Ihnen, genauer zu untersuchen, wo bestimmte Anweisungen ausgeführt werden und was sie bewirken. Dieser Abschnitt konzentriert sich auf `MAX_TOTAL_MEMORY`. Weitere Informationen zu allen verfügbaren Spalten finden Sie in der Dokumentation Leistungsschema – Übersichtstabellen für Anweisungen.

```
mysql> SHOW TABLES IN performance_schema LIKE 'events_statements_summary_%';

+-----------------------------------------------------------+
| Tables_in_performance_schema (events_statements_summary_%) |
+-----------------------------------------------------------+
| events_statements_summary_by_account_by_event_name         |
| events_statements_summary_by_digest                        |
| events_statements_summary_by_host_by_event_name            |
| events_statements_summary_by_program                       |
| events_statements_summary_by_thread_by_event_name          |
| events_statements_summary_by_user_by_event_name            |
| events_statements_summary_global_by_event_name             |
+-----------------------------------------------------------+
7 rows in set (0.00 sec)
```

Zuerst überprüfen wir `events_statements_summary_by_digest`, um `MAX_TOTAL_MEMORY` zu sehen.

Aus diesen Daten ergibt sich Folgendes:

- Die Abfrage mit dem `20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a`-Digest scheint ein möglicher Kandidat für diese Speichernutzung zu sein. `MAX_TOTAL_MEMORY` beträgt 537 450 710, was dem Höchststand entspricht, den wir bei dem `memory/temptable/physical_ram`-Ereignis in `sys.x$memory_global_by_current_bytes` gesehen haben.
- Eine Ausführung fand viermal (`COUNT_STAR`) statt, zuerst am 26.03.2024 um 04:08:34,943256 und zuletzt um 26.03.2024 um 04:43:06,998310.

```
mysql> SELECT SCHEMA_NAME,DIGEST,COUNT_STAR,MAX_TOTAL_MEMORY,FIRST_SEEN,LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest ORDER BY MAX_TOTAL_MEMORY
 DESC LIMIT 5;

+-------------+----------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
| SCHEMA_NAME | DIGEST                                                          |
 COUNT_STAR | MAX_TOTAL_MEMORY | FIRST_SEEN                 | LAST_SEEN
  |
+-------------+----------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
| sysbench    | 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a |
     4 |        537450710 | 2024-03-26 04:08:34.943256 | 2024-03-26 04:43:06.998310 |
| NULL        | f158282ea0313fefd0a4778f6e9b92fc7d1e839af59ebd8c5eea35e12732c45d |
     4 |          3636413 | 2024-03-26 04:29:32.712348 | 2024-03-26 04:36:26.269329 |
| NULL        | 0046bc5f642c586b8a9afd6ce1ab70612dc5b1fd2408fa8677f370c1b0ca3213 |
     2 |          3459965 | 2024-03-26 04:31:37.674008 | 2024-03-26 04:32:09.410718 |
| NULL        | 8924f01bba3c55324701716c7b50071a60b9ceaf17108c71fd064c20c4ab14db |
     1 |          3290981 | 2024-03-26 04:31:49.751506 | 2024-03-26 04:31:49.751506 |
| NULL        | 90142bbcb50a744fcec03a1aa336b2169761597ea06d85c7f6ab03b5a4e1d841 |
     1 |          3131729 | 2024-03-26 04:15:09.719557 | 2024-03-26 04:15:09.719557 |
+-------------+----------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
5 rows in set (0.00 sec)
```

Jetzt, da wir den problematischen Digest kennen, können wir mehr Details abrufen, z. B. den Abfragetext, den Benutzer und die Stelle, an der er ausgeführt wurde. Anhand des zurückgegebenen

Digest-Texts können wir erkennen, dass es sich um einen allgemeinen Tabellenausdruck (CTE) handelt, der 4 temporäre Tabellen erstellt und 4 Tabellenscans durchführt, was sehr ineffizient ist.

```
mysql> SELECT
 SCHEMA_NAME,DIGEST_TEXT,QUERY_SAMPLE_TEXT,MAX_TOTAL_MEMORY,SUM_ROWS_SENT,SUM_ROWS_EXAMINED,SUM
FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST='20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a'\G;

*************************** 1. row ***************************
           SCHEMA_NAME: sysbench
           DIGEST_TEXT: WITH RECURSIVE `cte` ( `n` ) AS ( SELECT ? FROM `sbtest1` UNION
 ALL SELECT `id` + ? FROM `sbtest1` ) SELECT * FROM `cte`
     QUERY_SAMPLE_TEXT: WITH RECURSIVE cte (n) AS (   SELECT 1  from sbtest1 UNION ALL
  SELECT id + 1 FROM sbtest1) SELECT * FROM cte
      MAX_TOTAL_MEMORY: 537450710
         SUM_ROWS_SENT: 80000000
     SUM_ROWS_EXAMINED: 80000000
SUM_CREATED_TMP_TABLES: 4
     SUM_NO_INDEX_USED: 4
1 row in set (0.01 sec)
```

Weitere Informationen zur `events_statements_summary_by_digest`-Tabelle und zu anderen Übersichtstabellen für Leistungsschemaanweisungen finden Sie unter Übersichtstabellen für Anweisungen in der MySQL-Dokumentation.

Sie können auch eine EXPLAIN- oder EXPLAIN ANALYZE-Anweisung ausführen, um weitere Informationen zu erhalten.

> **Note**
>
> `EXPLAIN ANALYZE` kann mehr Informationen bereitstellen als `EXPLAIN`, führt aber auch die Abfrage aus. Seien Sie also vorsichtig.

```
-- EXPLAIN
mysql> EXPLAIN WITH RECURSIVE cte (n) AS (SELECT 1  FROM sbtest1 UNION ALL SELECT id +
 1 FROM sbtest1) SELECT * FROM cte;

+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
```

```
| id | select_type | table      | partitions | type  | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra       |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL   | NULL          | NULL | NULL    |
 NULL | 19221520 |   100.00 | NULL        |
|  2 | DERIVED     | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
|  3 | UNION       | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)

-- EXPLAIN format=tree
mysql> EXPLAIN format=tree WITH RECURSIVE cte (n) AS (SELECT 1 FROM sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
1 row in set (0.00 sec)

-- EXPLAIN ANALYZE
mysql> EXPLAIN ANALYZE WITH RECURSIVE cte (n) AS (SELECT 1 from sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6) (actual
 time=6666..9201 rows=20e+6 loops=1)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6) (actual
 time=6666..6666 rows=20e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0365..2006 rows=10e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0311..2494 rows=10e+6 loops=1)
1 row in set (10.53 sec)
```

Aber wer ist für die Ausführung verantwortlich? Wir können im Leistungsschema sehen, dass der `destructive_operator`-Benutzer den `MAX_TOTAL_MEMORY`-Wert von 537 450 710 hatte, was wiederum den vorherigen Ergebnissen entspricht.

> **Note**
>
> Das Leistungsschema wird im Arbeitsspeicher gespeichert und sollte daher nicht als alleinige Quelle für Überprüfungen verwendet werden. Wenn Sie einen Verlauf der ausgeführten Anweisungen und der Benutzer, von denen diese stammen, verwalten müssen, sollten Sie das Advanced Auditing von Aurora aktivieren. Wenn Sie auch Informationen zur Speichernutzung verwalten müssen, sollten Sie die Überwachung so konfigurieren, dass diese Werte exportiert und gespeichert werden.

```
mysql> SELECT USER,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_user_by_event_name
ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+----------------------+---------------------------+------------+------------------+
| USER                 | EVENT_NAME                | COUNT_STAR | MAX_TOTAL_MEMORY |
+----------------------+---------------------------+------------+------------------+
| destructive_operator | statement/sql/select      |          4 |        537450710 |
| rdsadmin             | statement/sql/select      |       4172 |          3290981 |
| rdsadmin             | statement/sql/show_tables |          2 |          3615821 |
| rdsadmin             | statement/sql/show_fields |          2 |          3459965 |
| rdsadmin             | statement/sql/show_status |         75 |          1914976 |
+----------------------+---------------------------+------------+------------------+
5 rows in set (0.00 sec)

mysql> SELECT HOST,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_host_by_event_name
WHERE HOST != 'localhost' AND COUNT_STAR>0 ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+------------+----------------------+------------+------------------+
| HOST       | EVENT_NAME           | COUNT_STAR | MAX_TOTAL_MEMORY |
+------------+----------------------+------------+------------------+
| 10.0.8.231 | statement/sql/select |          4 |        537450710 |
+------------+----------------------+------------+------------------+
1 row in set (0.00 sec)
```

Beispiel 3: Der freie Speicherplatz sinkt kontinuierlich und wird nicht wieder zurückgewonnen

Die InnoDB-Datenbank-Engine verwendet eine Reihe spezialisierter Ereignisse zur Speichernachverfolgung für verschiedene Komponenten. Diese spezifischen Ereignisse ermöglichen

eine detaillierte Nachverfolgung der Speichernutzung in wichtigen InnoDB-Subsystemen, zum Beispiel:

- `memory/innodb/buf0buf` – Dieses Ereignis dient zur Überwachung der Speicherzuweisungen für den InnoDB-Pufferpool.
- `memory/innodb/ibuf0ibuf` – Dieses Ereignis verfolgt speziell Speicheränderungen im Zusammenhang mit dem InnoDB-Änderungspuffer nach.

Zum Ermitteln der primären Speicherverbraucher können wir `sys.memory_global_by_current_bytes` abfragen:

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
| memory/innodb/memory                                            | 5.28 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                          | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                | 388.95 MiB    |
| memory/innodb/std                                               | 226.88 MiB    |
| memory/innodb/fil0fil                                           | 198.49 MiB    |
| memory/sql/binlog_io_cache                                      | 128.00 MiB    |
| memory/innodb/mem0mem                                           | 96.82 MiB     |
| memory/innodb/dict0dict                                         | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                      | 88.00 MiB     |
+-----------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

Die Ergebnisse zeigen, dass `memory/innodb/memory` der primäre Verbraucher ist und 5,28 GiB des aktuell zugewiesenen Speichers nutzt. Dieses Ereignis dient als eine Kategorie für Speicherzuweisungen innerhalb verschiedener InnoDB-Komponenten, die nicht mit spezifischeren Warteereignissen verknüpft sind, wie dem zuvor erwähnten `memory/innodb/buf0buf`.

Nachdem wir festgestellt haben, dass InnoDB-Komponenten die primären Speicherverbraucher sind, befassen wir uns nun mithilfe des folgenden MySQL-Befehls näher mit den Einzelheiten:

```
SHOW ENGINE INNODB STATUS \G;
```

Der Befehl SHOW ENGINE INNODB STATUS liefert einen umfassenden Statusbericht für die InnoDB-Speicher-Engine, einschließlich detaillierter Speichernutzungsstatistiken für verschiedene InnoDB-Komponenten. Mit ihm lässt sich herausfinden, welche spezifischen InnoDB-Strukturen oder -Operationen den meisten Speicher verbrauchen. Weitere Informationen finden Sie unter InnoDB-In-Memory-Strukturen in der MySQL-Dokumentation.

Wenn wir den Abschnitt `BUFFER POOL AND MEMORY` des InnoDB-Statusberichts analysieren, stellen wir fest, dass 5 051 647 748 Bytes (4,7 GiB) dem Wörterbuchobjekt-Cache zugewiesen sind, was 89 % des von `memory/innodb/memory` nachverfolgten Speichers ausmacht.

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5051647748
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

Der Wörterbuchobjekt-Cache ist ein gemeinsam genutzter globaler Cache, der Datenwörterbuchobjekte im Arbeitsspeicher speichert, auf die zuvor zugegriffen wurde, um die Wiederverwendung von Objekten zu ermöglichen und die Leistung zu verbessern. Die hohe Speicherzuweisung für den Wörterbuchobjekt-Cache deutet auf eine große Anzahl von Datenbankobjekten im Cache des Datenwörterbuchs hin.

Da wir nun wissen, dass der Datenwörterbuch-Cache ein primärer Verbraucher ist, untersuchen wir diesen nun auf offene Tabellen. Zum Ermitteln der Anzahl der Tabellen im Tabellendefinitions-Cache fragen Sie die globale Statusvariable open_table_definitions ab.

```
mysql> show global status like 'open_table_definitions';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Open_table_definitions | 20000 |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Weitere Informationen finden Sie unter How MySQL Opens and Closes Tables in der MySQL-Dokumentation.

Sie können die Anzahl der Tabellendefinitionen im Datenwörterbuch-Cache begrenzen, indem Sie den `table_definition_cache`-Parameter im DB-Cluster oder in der DB-Instance-Parametergruppe begrenzen. In Aurora MySQL dient dieser Wert als weicher Grenzwert hinsichtlich der Anzahl der Tabellen im Tabellendefinitions-Cache. Der Standardwert hängt von der Instance-Klasse ab und ist wie folgt festgelegt:

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

Wenn die Anzahl der Tabellen den `table_definition_cache`-Grenzwert überschreitet, bereinigt ein Least Recently Used (LRU)-Mechanismus Tabellen im Cache und entfernt diese. Tabellen, die an Fremdschlüsselbeziehungen beteiligt sind, werden jedoch nicht in die LRU-Liste aufgenommen, wodurch verhindert wird, dass sie entfernt werden.

In unserem aktuellen Szenario führen wir FLUSH TABLES aus, um den Tabellendefinitions-Cache zu leeren. Diese Aktion führt zu einem deutlichen Rückgang der globalen Statusvariable Open_Table_Definitions von 20 000 auf 12, wie hier dargestellt:

```
mysql> show global status like 'open_table_definitions';

+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Open_table_definitions | 12    |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Trotz dieser Reduzierung stellen wir fest, dass die Speicherzuweisung für `memory/innodb/memory` mit 5,18 GiB hoch bleibt. Auch am zugewiesenen Wörterbuchspeicher ändert sich nichts. Dies geht aus den folgenden Abfrageergebnissen hervor:

```
mysql> SELECT event_name,current_alloc FROM sys.memory_global_by_current_bytes LIMIT
 10;

+-----------------------------------------------------------------+---------------+
| event_name                                                      | current_alloc |
+-----------------------------------------------------------------+---------------+
```

```
| memory/innodb/memory                                                | 5.18 GiB      |
| memory/performance_schema/table_io_waits_summary_by_index_usage | 495.00 MiB    |
| memory/performance_schema/table_shares                              | 488.00 MiB    |
| memory/sql/TABLE_SHARE::mem_root                                    | 388.95 MiB    |
| memory/innodb/std                                                   | 226.88 MiB    |
| memory/innodb/fil0fil                                               | 198.49 MiB    |
| memory/sql/binlog_io_cache                                          | 128.00 MiB    |
| memory/innodb/mem0mem                                               | 96.82 MiB     |
| memory/innodb/dict0dict                                             | 96.76 MiB     |
| memory/performance_schema/rwlock_instances                          | 88.00 MiB     |
+---------------------------------------------------------------------+---------------+
10 rows in set (0.00 sec)
```

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 0
Dictionary memory allocated 5001599639
Buffer pool size 170512
Free buffers 142568
Database pages 27944
Old database pages 10354
Modified db pages 6
Pending reads 0
```

Diese anhaltend hohe Speichernutzung kann auf Tabellen zurückgeführt werden, die an Fremdschlüsselbeziehungen beteiligt sind. Diese Tabellen werden nicht zur Entfernung in die LRU-Liste aufgenommen, was erklärt, warum die Speicherzuweisung auch nach dem Leeren des Tabellendefinitions-Cache hoch bleibt.

So beheben Sie dieses Problem:

1. Überprüfen und optimieren Sie Ihr Datenbankschema, insbesondere im Hinblick auf Fremdschlüsselbeziehungen.
2. Erwägen Sie, zu einer größeren DB-Instance-Klasse zu wechseln, die mehr Speicher für Ihre Wörterbuchobjekte bietet.

Wenn Sie diese Schritte befolgen und die Muster der Speicherzuweisung kennen, können Sie die Speichernutzung in Ihrer DB-Instance von Aurora MySQL besser verwalten und potenziellen Leistungsproblemen aufgrund von Speicherknappheit vorbeugen.

Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken

Mithilfe des Parameters der Instance-Ebene Aurora MySQL `aurora_oom_response` kann die DB-Instance den Arbeitsspeicher des Systems überwachen und den von verschiedenen Anweisungen und Verbindungen verbrauchten Arbeitsspeicher schätzen. Wenn der Arbeitsspeicher des Systems knapp wird, können eine Reihe von Aktionen durchgeführt werden, um zu versuchen, Arbeitsspeicher freizugeben. Das System führt diese Aktionen aus, um einen Neustart der Datenbank aufgrund von Speicherproblemen (Out-Of-Memory, OOM) zu vermeiden. Der Parameter der Instance-Ebene gibt eine Zeichenfolge mit durch Kommata getrennten Aktionen an, die eine DB-Instance durchführt, wenn nur wenig Arbeitsspeicher zur Verfügung steht. Der Parameter `aurora_oom_response` wird für die Aurora-MySQL-Versionen 2 und 3 unterstützt.

Die folgenden Werte und deren Kombinationen können für den Parameter `aurora_oom_response` verwendet werden. Eine leere Zeichenfolge bedeutet, dass keine Aktion ausgeführt wird und die Funktion effektiv deaktiviert wird, wodurch die Datenbank anfällig für OOM-Neustarts wird.

- `decline` – Lehnt neue Abfragen ab, wenn der DB-Instance wenig Speicherplatz zur Verfügung steht
- `kill_connect` – Schließt Datenbankverbindungen, die viel Speicher verbrauchen, und beendet aktuelle Transaktionen und Anweisungen für die Datendefinitionssprache (Data Definition Language, DDL). Diese Antwort wird nicht für Aurora-MySQL-Version 2 unterstützt.

  Weitere Informationen finden Sie unter KILL-Anweisung in der MySQL-Dokumentation.
- `kill_query` – Beendet die Abfragen in absteigender Reihenfolge des Speicherverbrauchs, bis der Instance-Speicher oberhalb des unteren Schwellenwerts liegt. DDL-Anweisungen werden nicht beendet.

  Weitere Informationen finden Sie unter KILL-Anweisung in der MySQL-Dokumentation.
- `print` – Druckt nur die Abfragen mit hohem Speicherbedarf
- `tune` – Stellt die Caches der internen Tabellen so ein, dass etwas Arbeitsspeicher für das System freigegeben wird. Aurora MySQL verringert den Speicher, der für Caches wie `table_open_cache` und `table_definition_cache` bei geringem Speicherbedarf verwendet wird. Schließlich setzt Aurora MySQL ihre Speichernutzung wieder auf den Normalwert zurück, wenn das System nicht mehr zu wenig Arbeitsspeicher hat.

  Weitere Informationen finden Sie unter table_open_cache und table_definition_cache in der MySQL-Dokumentation.

- `tune_buffer_pool` – Verringert die Größe des Pufferpools, um Arbeitsspeicher freizugeben und ihn dem Datenbankserver zur Verarbeitung von Verbindungen zur Verfügung zu stellen. Diese Antwort wird für Aurora-MySQL-Version 3.06 und höher unterstützt.

  Sie müssen `tune_buffer_pool` entweder mit `kill_query` oder `kill_connect` im Parameterwert `aurora_oom_response` kombinieren. Andernfalls erfolgt die Größenanpassung des Pufferpools nicht, selbst wenn Sie `tune_buffer_pool` in den Parameterwert mit einbeziehen.

Für Aurora-MySQL-Versionen unter 3.06 werden bei DB-Instance-Klassen mit maximal 4 GiB Arbeitsspeicher, wenn die Instance unter Arbeitsspeicherdruck steht, standardmäßig die Aktionen `print`, `tune`, `decline` und `kill_query` ausgeführt. Für DB-Instance-Klassen mit mehr als 4 GiB Arbeitsspeicher ist standardmäßig kein Parameterwert angegeben (deaktiviert).

In Aurora-MySQL-Version 3.06 und höher schließt Aurora MySQL für DB-Instance-Klassen mit einem Arbeitsspeicher von weniger als oder gleich 4 GiB auch die Verbindungen mit dem höchsten Speicherverbrauch (`kill_connect`). Für DB-Instance-Klassen mit mehr als 4 GiB Arbeitsspeicher ist der Standardwert des Parameters `print`.

In Aurora-MySQL-Version 3.09 und höher ist für DB-Instance-Klassen mit mehr als 4 GiB Arbeitsspeicher der Standardwert des Parameters `print,decline,kill_connect`.

Wenn Sie häufig auf Speicherprobleme stoßen, kann die Speichernutzung mithilfe von Speicherübersichtstabellen überwacht werden, sofern `performance_schema` aktiviert ist.

Informationen zu Amazon-CloudWatch-Metriken im Zusammenhang mit OOM finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora. Informationen zu globalen Statusvariablen im Zusammenhang mit OOM finden Sie unter Globale Statusvariablen von Aurora MySQL.

Protokollieren bei Aurora-MySQL-Datenbanken

Aurora-MySQL-Protokolle liefern wichtige Informationen über Datenbankaktivitäten und -fehler. Durch die Aktivierung dieser Protokolle können Sie Probleme identifizieren und beheben, mehr über die Datenbankleistung erfahren und Datenbankaktivitäten überprüfen. Wir empfehlen Ihnen, diese Protokolle für all Ihre DB-Instances von Aurora MySQL zu aktivieren, um eine optimale Leistung und Verfügbarkeit der Datenbanken sicherzustellen. Die folgenden Arten der Protokollierung können aktiviert werden. Jedes Protokoll enthält spezifische Informationen, anhand derer Auswirkungen auf die Datenbankverarbeitung aufgedeckt werden können.

- Fehler – Aurora MySQL schreibt das Fehlerprotokoll nur beim Startup, Herunterfahren und beim Auftreten von Fehlern. Eine DB-Instance kann Stunden oder Tage lang laufen, ohne dass neue Einträge in das Fehlerprotokoll geschrieben werden. Wenn Sie keine neuen Einträge sehen, sind im Server keine Fehler aufgetreten, die zu einem Eintrag in das Protokoll führen würden. Die Fehlerprotokollierung ist standardmäßig aktiviert. Weitere Informationen finden Sie unter Aurora-MySQL-Fehlerprotokolle.
- Allgemein – Das allgemeine Protokoll enthält detaillierte Informationen über Datenbankaktivitäten, einschließlich aller SQL-Anweisungen, die von der Datenbank-Engine ausgeführt werden. Weitere Informationen zur Aktivierung der allgemeinen Protokollierung und zur Festlegung von Protokollparametern finden Sie unter Aurora-MySQL-Protokolle für langsame Abfragen und allgemeine Protokolle und unter Das allgemeine Abfrageprotokoll in der MySQL-Dokumentation.

  > **Note**
  >
  > Allgemeine Protokolle können sehr umfangreich werden und viel Speicherplatz beanspruchen. Weitere Informationen finden Sie unter Protokollrotation und -aufbewahrung für Aurora MySQL.

- Slow Query – Das Slow-Query-Protokoll besteht aus SQL-Anweisungen, für deren Ausführung mehr als long_query_time Sekunden benötigt werden und mindestens min_examined_row_limit Zeilen überprüft werden müssen. Sie können das Slow-Query-Protokoll verwenden, um Abfragen zu finden, deren Ausführung viel Zeit in Anspruch nimmt und die daher für eine Optimierung in Frage kommen.

  Der Standardwert für `long_query_time` ist 10 Sekunden. Wir empfehlen, mit einem hohen Wert zu beginnen, um die langsamsten Abfragen zu identifizieren, und sich dann zwecks Optimierung nach unten vorzuarbeiten.

Sie können auch verwandte Parameter wie `log_slow_admin_statements` und `log_queries_not_using_indexes` verwenden. Vergleichen Sie `rows_examined` mit `rows_returned`. Wenn `rows_examined` viel größer ist als `rows_returned`, können diese Abfragen möglicherweise blockierend wirken.

In Aurora MySQL-Version 3 können Sie `log_slow_extra` aktivieren, um weitere Informationen zu erhalten. Weitere Informationen finden Sie unter Slow-Query-Protokollinhalte in der MySQL-Dokumentation. Sie können `long_query_time` auch auf Sitzungsebene ändern, um die Abfrageausführung interaktiv zu debuggen. Dies ist besonders nützlich, wenn `log_slow_extra` global aktiviert ist.

Weitere Informationen zur Slow-Query-Aktivierung und der Festlegung der Protokollparameter finden Sie unter Aurora-MySQL-Protokolle für langsame Abfragen und allgemeine Protokolle und unter Das Slow-Query-Protokoll in der MySQL-Dokumentation.

- Audit – Das Auditprotokoll überwacht und protokolliert Datenbankaktivitäten. Die Auditprotokollierung für Aurora MySQL heißt Advanced Auditing. Zum Aktivieren von Advanced Auditing legen Sie bestimmte DB-Cluster-Parameter fest. Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.
- Binär – Das Binärprotokoll (Binlog) enthält Ereignisse, die Datenbankänderungen beschreiben, beispielsweise Tabellenerstellungsvorgänge und Änderungen an Tabellendaten. Es enthält auch Ereignisse für Anweisungen, die möglicherweise Änderungen bewirken können (z. B. ein DELETE-Befehl ohne Zeilenübereinstimmung), sofern nicht die zeilenbasierte Protokollierung verwendet wird. Das Binärprotokoll enthält auch Informationen darüber, wie lange jede Anweisung gedauert hat, die Daten aktualisiert hat.

  Wenn Sie einen Server mit aktivierter Binärprotokollierung ausführen, ist die Leistung etwas geringer. Die Vorteile des Binärprotokolls, das Ihnen die Einrichtung der Replikation sowie Wiederherstellungsvorgänge ermöglicht, überwiegen jedoch im Allgemeinen den Nachteil dieser geringfügigen Leistungseinbußen.

  > **Note**
  >
  > Aurora MySQL benötigt keine binäre Protokollierung für Wiederherstellungsvorgänge.

Weitere Informationen zum Aktivieren der Binärprotokollierung und zum Festlegen des Binlog-Formats finden Sie unter Konfigurieren der Aurora MySQL-Binärprotokollierung für Single-AZ-Datenbanken und unter Das Binärprotokoll in der MySQL-Dokumentation.

Sie können die Fehler-, allgemeinen, Slow-Query- und Auditprotokolle in Amazon CloudWatch Logs veröffentlichen. Weitere Informationen finden Sie unter Veröffentlichen von Datenbankprotokollen in Amazon CloudWatch Logs.

Ein weiteres nützliches Tool zur Zusammenfassung der Dateien der allgemeinen Protokolle, der Slow-Query-Protokolle und der Binärprotokolle ist pt-query-digest.

Beheben von Verbindungsproblemen für Datenbanken von Aurora MySQL

Die Sicherstellung einer zuverlässigen Verbindung zwischen Ihren Anwendungen und Ihrer RDS-DB-Instance ist entscheidend für den reibungslosen Betrieb Ihrer Workloads. Verbindungsprobleme können jedoch aufgrund verschiedener Faktoren wie Netzwerkkonfigurationen, Authentifizierungsprobleme oder Ressourcenbeschränkungen auftreten. Dieses Handbuch soll einen umfassenden Ansatz zur Behebung von Verbindungsproblemen mit Aurora MySQL bieten.

Inhalt

- Identifizieren von Problemen mit der Datenbankverbindung für Aurora MySQL
- Erfassen von Daten zu Verbindungsproblemen für Aurora MySQL
- Überwachen von Datenbankverbindungen für Aurora MySQL
  - Weitere Überwachung für Aurora MySQL
- Verbindungsfehlercodes für Aurora MySQL
- Empfehlungen zur Parameteroptimierung für Aurora MySQL
- Beispiele für die Behebung von Datenbankverbindungsproblemen für Aurora MySQL
  - Beispiel 1: Fehlerbehebung bei fehlgeschlagenen Verbindungsversuchen
  - Beispiel 2: Behebung ungewöhnlicher Trennungen von Client-Verbindungen
  - Beispiel 3: Problembehandlung bei fehlgeschlagenen IAM-Verbindungsversuchen

Identifizieren von Problemen mit der Datenbankverbindung für Aurora MySQL

Die Identifizierung der spezifischen Kategorie des Verbindungsproblems kann dabei helfen, die möglichen Ursachen einzugrenzen und den Fehlerbehebungsprozess zu steuern. Für jede Kategorie sind möglicherweise unterschiedliche Ansätze und Techniken zur Diagnose und Lösung erforderlich. Probleme mit der Datenbankverbindung lassen sich grob in die folgenden Kategorien einteilen.

Verbindungsfehler und Ausnahmen

> Verbindungsfehler und Ausnahmen können aus verschiedenen Gründen auftreten, darunter falsche Verbindungszeichenfolgen, Authentifizierungsfehler, Netzwerkunterbrechungen oder Datenbankserverprobleme. Zu den Ursachen können falsch konfigurierte Verbindungsparameter, ungültige Anmeldeinformationen, Netzwerkausfälle oder Abstürze oder Neustarts von Datenbankservern zählen. Falsch konfigurierte Sicherheitsgruppen, VPC-Einstellungen (Virtual Private Cloud), Netzwerk-Zugriffskontrolllisten (ACLs) und Routentabellen, die Subnetzen zugeordnet sind, können ebenfalls zu Verbindungsproblemen führen.

Verbindungslimit erreicht

Dieses Problem tritt auf, wenn die Anzahl der gleichzeitigen Verbindungen zum Datenbankserver das maximal zulässige Limit überschreitet. Datenbankserver haben in der Regel ein konfigurierbares maximales Verbindungslimit, das durch den Parameter „max_connections“ in den Clustern und Instance-Parametergruppen definiert wird. Durch die Festlegung eines Verbindungslimits stellt der Datenbankserver sicher, dass er über ausreichende Ressourcen (z. B. Arbeitsspeicher, CPU und Datei-Handles) verfügt, um die vorhandenen Verbindungen effizient zu verarbeiten und eine akzeptable Leistung zu bieten. Zu den Ursachen können Verbindungslecks in der Anwendung, ineffizientes Verbindungspooling oder ein unerwarteter Anstieg der Verbindungsanforderungen gehören.

Verbindungs-Timeouts

Verbindungs-Timeouts treten auf, wenn die Clientanwendung innerhalb eines angegebenen Timeout-Zeitraums keine Verbindung mit dem Datenbankserver herstellen kann. Zu den häufigsten Ursachen zählen Netzwerkprobleme, Serverüberlastung, Firewall-Regeln und falsch konfigurierte Verbindungseinstellungen.

Timeout für Leerlaufverbindungen

Leerlaufverbindungen, die über einen längeren Zeitraum inaktiv bleiben, können vom Datenbankserver automatisch geschlossen werden, um Ressourcen zu schonen. Dieses Timeout ist in der Regel mit `wait_timeout` und `interactive_timeout parameters` konfigurierbar und sollte an die Verbindungsnutzungsmuster der Anwendung angepasst werden. Zu den Ursachen können eine Anwendungslogik gehören, die Verbindungen für längere Zeiträume inaktiv lässt, oder eine unsachgemäße Verbindungsverwaltung.

Zeitweilige Trennung vorhandener Verbindungen

Diese Fehlerklasse bezieht sich auf ein Szenario, in dem bestehende Verbindungen zwischen einer Clientanwendung und der Datenbank unerwartet beendet oder in unregelmäßigen Abständen getrennt werden, obwohl sie aktiv sind und verwendet werden. Diese Trennungen treten zeitweise auf, d. h., sie treten in unregelmäßigen Intervallen und nicht konsistent auf. Die Ursachen können Folgendes beinhalten:

- Probleme mit Datenbankservern wie Neustarts oder Failover
- Unsachgemäße Behandlung von Anwendungsverbindungen
- Load Balancing- und Proxy-Probleme
- Netzwerkinstabilität

- Probleme mit Komponenten oder Middleware von Drittanbietern, die am Verbindungspfad beteiligt sind
- Timeouts für die Abfrageausführung
- Server- oder clientseitige Ressourceneinschränkungen

Die Identifizierung der Ursache durch umfassende Überwachung, Protokollierung und Analyse ist von entscheidender Bedeutung. Die Implementierung einer geeigneten Fehlerbehandlung, von Verbindungspooling und von Wiederholungsmechanismen kann jedoch dazu beitragen, die Auswirkungen dieser zeitweiligen Trennungen auf die Funktionalität und die Benutzererfahrung der Anwendung zu mindern.

Erfassen von Daten zu Verbindungsproblemen für Aurora MySQL

Das Sammeln umfassender Daten zu den Anwendungs-, Datenbank-, Netzwerk- und Infrastrukturkomponenten ist entscheidend für die effektive Behebung von Verbindungsproblemen zwischen einer Anwendung und einer Datenbank von Aurora MySQL. Durch das Sammeln relevanter Protokolle, Konfigurationen und Diagnoseinformationen gewinnen Sie wertvolle Erkenntnisse, die Ihnen helfen können, die Ursache der Verbindungsprobleme zu identifizieren und Sie zu einer geeigneten Lösung zu führen.

Netzwerkprotokolle und -konfigurationen, wie Sicherheitsgruppenregeln, VPC-Einstellungen und Routing-Tabellen, sind wichtig, um potenzielle netzwerkbezogene Engpässe oder Fehlkonfigurationen zu identifizieren, die verhindern können, dass die Anwendung eine erfolgreiche Verbindung mit der Datenbank herstellt. Durch die Analyse dieser Netzwerkkomponenten können Sie sicherstellen, dass die erforderlichen Ports geöffnet, IP-Adressen zulässig und die Routing-Konfigurationen korrekt eingerichtet sind.

Zeitstempel

Notieren Sie sich die genauen Zeitstempel, zu denen die Verbindungsprobleme auftreten. Auf diese Weise können Sie Muster erkennen oder Probleme mit anderen Ereignissen oder Aktivitäten korrelieren.

DB-Engine-Protokolle

Überprüfen Sie zusätzlich zu den allgemeinen Datenbankprotokollen auch die Datenbank-Engine-Protokolle (z. B. das MySQL-Fehlerprotokoll und das Protokoll für langsame Abfragen) auf relevante Informationen oder Fehler, die möglicherweise mit den zeitweiligen

Verbindungsproblemen zusammenhängen. Weitere Informationen finden Sie unter Protokollieren bei Aurora-MySQL-Datenbanken.

Clientanwendungsprotokolle

Sammeln Sie detaillierte Protokolle von den Clientanwendungen, die eine Verbindung zur Datenbank herstellen. Anwendungsprotokolle bieten Einblick in die Verbindungsversuche, Fehler und alle relevanten Informationen aus Sicht der Anwendung. Dadurch können Probleme im Zusammenhang mit Verbindungszeichenfolgen, Anmeldeinformationen für die Authentifizierung oder die Behandlung von Verbindungen auf Anwendungsebene aufgedeckt werden.

Datenbankprotokolle hingegen bieten Einblicke in datenbankseitige Fehler, langsame Abfragen oder Ereignisse, die zu Verbindungsproblemen beitragen können. Weitere Informationen finden Sie unter Protokollieren bei Aurora-MySQL-Datenbanken.

Clientumgebungsvariablen

Prüfen Sie, ob Umgebungsvariablen oder Konfigurationseinstellungen auf der Clientseite die Datenbankverbindung beeinflussen könnten, z. B. Proxyeinstellungen, SSL/TLS Einstellungen oder andere relevante Variablen.

Versionen der Clientbibliothek

Stellen Sie sicher, dass der Client die neuesten Versionen aller Datenbanktreiber, Bibliotheken oder Frameworks verwendet, die für die Datenbankverbindung verwendet werden. Veraltete Versionen können bekannte Probleme oder Kompatibilitätsprobleme aufweisen.

Clientnetzwerkerfassung

Führen Sie eine Netzwerkerfassung auf der Clientseite mit einem Tool wie Wireshark oder `tcpdump` durch, wenn Verbindungsprobleme auftreten. Auf diese Weise können netzwerkbezogene Probleme oder Anomalien auf der Clientseite identifiziert werden.

Netzwerktopologie des Clients

Machen Sie sich mit der Netzwerktopologie des Clients vertraut, einschließlich aller Firewalls, Load Balancer oder anderer Komponenten wie RDS-Proxy oder Proxy SQL, die Verbindungen zur Datenbank herstellen, anstatt dass der Client direkt Verbindungen herstellt.

Einstellungen für das Client-Betriebssystem

Überprüfen Sie die Betriebssystemeinstellungen des Clients, die sich auf die Netzwerkkonnektivität auswirken können, z. B. Firewall-Regeln, Netzwerkadaptereinstellungen und alle anderen relevanten Einstellungen.

Konfiguration für Verbindungspooling

Wenn Sie in Ihrer Anwendung einen Mechanismus für das Verbindungspooling verwenden, überprüfen Sie die Konfigurationseinstellungen und überwachen Sie die Poolmetriken (z. B. aktive Verbindungen, inaktive Verbindungen und Verbindungs-Timeouts), um sicherzustellen, dass der Pool ordnungsgemäß funktioniert. Überprüfen Sie auch die Pooleinstellungen, z. B. die maximale Poolgröße, die minimale Poolgröße und die Einstellungen für die Verbindungsvalidierung, um sicherzustellen, dass sie ordnungsgemäß konfiguriert sind.

Verbindungszeichenfolge

Die Verbindungszeichenfolge umfasst in der Regel Parameter wie den Hostnamen oder den Endpunkt, die Portnummer, den Datenbanknamen und die Anmeldeinformationen für die Authentifizierung. Durch die Analyse der Verbindungszeichenfolge können potenzielle Fehlkonfigurationen oder falsche Einstellungen identifiziert werden, die Verbindungsprobleme verursachen können. Beispielsweise kann ein falscher Hostname oder eine falsche Portnummer verhindern, dass der Client die Datenbank-Instance erreicht, während ungültige Anmeldeinformationen für die Authentifizierung zu Authentifizierungsfehlern und Ablehnungen von Verbindungen führen können. Darüber hinaus kann die Verbindungszeichenfolge Probleme im Zusammenhang mit Verbindungspooling, Timeouts oder anderen verbindungsspezifischen Einstellungen aufdecken, die zu Verbindungsproblemen beitragen können. Die Angabe der vollständigen Verbindungszeichenfolge, die von der Clientanwendung verwendet wird, kann dabei helfen, Fehlkonfigurationen auf dem Client zu ermitteln.

Datenbankmetriken

Überwachen Sie Datenbankmetriken wie CPU-Auslastung, Speicherauslastung und Festplatte I/O in Zeiten, in denen Verbindungsprobleme auftreten. Anhand dieser Informationen kann ermittelt werden, ob in der DB-Instance Ressourcenkonflikte oder Leistungsprobleme auftreten.

DB-Engine-Version

Beachten Sie die DB-Engine-Version von Aurora MySQL. AWS veröffentlicht regelmäßig Updates, die bekannte Probleme und Schwachstellen beheben und zu Leistungsverbesserungen führen. Daher wird dringend empfohlen, ein Upgrade auf die neuesten verfügbaren Versionen durchzuführen, da diese Updates häufig Fehlerkorrekturen und Verbesserungen enthalten, die sich speziell auf Konnektivität, Leistung und Stabilität beziehen. Die Bereitstellung der Informationen zur Datenbankversion zusammen mit den anderen gesammelten Details kann dazu beitragen Support , Verbindungsprobleme effektiv zu diagnostizieren und zu lösen.

Netzwerkmetriken

Sammeln Sie Netzwerkmetriken wie Latenz, Paketverlust und Durchsatz, wenn Verbindungsprobleme auftreten. Diese Daten können mit Tools wie `ping` und `traceroute` sowie mit Netzwerküberwachungstools gesammelt werden.

Quellen- und Clientdetails

Ermitteln Sie die IP-Adressen der Anwendungsserver, Load Balancer oder anderer Komponenten, die die Datenbankverbindungen initiieren. Dabei kann es sich um eine einzelne IP-Adresse oder einen Bereich von IP-Adressen (CIDR-Notation) handeln. Wenn es sich bei der Quelle um eine EC2 Amazon-Instance handelt, hilft es auch, den Instance-Typ, die Availability Zone, die Subnetz-ID und die mit der Instance verknüpften Sicherheitsgruppen sowie die Netzwerkschnittstellendetails wie private IP-Adresse und öffentliche IP-Adresse zu überprüfen.

Durch eine gründliche Analyse der gesammelten Daten können Sie Fehlkonfigurationen, Ressourcenbeschränkungen, Netzwerkunterbrechungen oder andere grundlegende Probleme identifizieren, die die zeitweiligen oder persistenten Verbindungsprobleme verursachen. Anhand dieser Informationen können Sie gezielte Maßnahmen ergreifen, z. B. Konfigurationen anpassen, Netzwerkprobleme lösen oder die Handhabung von Verbindungen auf Anwendungsebene verbessern.

Überwachen von Datenbankverbindungen für Aurora MySQL

Sie können auch die folgenden Metriken und Features verwenden, um Verbindungsprobleme zu überwachen und zu beheben.

CloudWatch Metriken

- `CPUUtilization` – Eine hohe CPU-Auslastung auf der DB-Instance kann zu einer langsamen Abfrageausführung führen, was Timeouts oder Ablehnungen von Verbindungen nach sich ziehen kann.
- `DatabaseConnections` – Überwachen Sie die Anzahl der aktiven Verbindungen zur DB-Instance. Eine hohe Anzahl von Verbindungen nahe dem konfigurierten Maximum kann auf potenzielle Verbindungsprobleme oder eine Erschöpfung des Verbindungspools hinweisen.
- `FreeableMemory` – Zu wenig verfügbarer Arbeitsspeicher kann aufgrund von Ressourcenbeschränkungen zu Leistungs- und Verbindungsproblemen führen.

- `NetworkReceiveThroughput` und `NetworkTransmitThroughput` – Ungewöhnliche Spitzen oder Rückgänge beim Netzwerkdurchsatz können auf Verbindungsprobleme oder Netzwerkengpässe hinweisen.

Performance-Insights-Metriken

Analysieren Sie Datenbankmetriken wie die folgenden, um Verbindungsprobleme in Aurora MySQL mithilfe von Performance Insights zu beheben:

- Aborted_clients
- Aborted_connects
- Verbindungen
- max_connections
- Threads_connected
- Threads_created
- Threads_running

Anhand dieser Metriken können Sie Verbindungsengpässe identifizieren, Netzwerk- oder Authentifizierungsprobleme erkennen, das Verbindungspooling optimieren und ein effizientes Thread-Management sicherstellen. Weitere Informationen finden Sie unter Performance-Insights-Zähler für Aurora MySQL.

Performance-Insights-Features

- Datenbanklast – Visualisieren Sie die Datenbanklast im Zeitverlauf und korrelieren Sie sie mit Verbindungsproblemen oder Leistungseinbußen.
- SQL-Statistiken – Analysieren Sie SQL-Statistiken, um ineffiziente Abfragen oder Datenbankvorgänge zu identifizieren, die zu Verbindungsproblemen beitragen können.
- Häufigste Abfragen – Identifizieren und analysieren Sie die ressourcenintensivsten Abfragen, um potenzielle Leistungsengpässe oder Abfragen mit langer Laufzeit zu identifizieren, die Verbindungsprobleme verursachen können.

Durch die Überwachung dieser Metriken und die Nutzung von Performance Insights können Sie Einblick in die Leistung, den Ressourcenverbrauch und mögliche Engpässe der Datenbank-Instance gewinnen, die Verbindungsprobleme verursachen können. Zum Beispiel:

- Ein hoher `DatabaseConnections`-Wert nahe dem Höchstgrenzwert kann auf eine Erschöpfung des Verbindungspools oder eine unsachgemäße Handhabung von Verbindungen hinweisen, was zu Verbindungsproblemen führt.

- Ein hoher `CPUUtilization`-Wert oder ein niedriger `FreeableMemory`-Wert kann auf Ressourcenbeschränkungen hinweisen, die zu einer langsamen Abfrageausführung und zu Timeouts oder Ablehnungen von Verbindungen führen können.
- Durch die Analyse der Top-Abfragen und der SQL-Statistiken können ineffiziente oder ressourcenintensive Abfragen identifiziert werden, die möglicherweise zu Verbindungsproblemen beitragen.

Darüber hinaus können Sie mithilfe der Überwachung von CloudWatch Protokollen und der Einrichtung von Alarmen Verbindungsprobleme proaktiv erkennen und darauf reagieren, bevor sie eskalieren.

Beachten Sie, dass diese Metriken und Tools zwar wertvolle Erkenntnisse liefern können, sie jedoch in Verbindung mit anderen Schritten zur Fehlerbehebung verwendet werden sollten. Indem Sie auch die Netzwerkkonfigurationen, Sicherheitsgruppenregeln und die Verbindungsverwaltung auf Anwendungsebene überprüfen, können Sie Verbindungsprobleme mit DB-Instances von Aurora MySQL umfassend diagnostizieren und lösen.

Weitere Überwachung für Aurora MySQL

CloudWatch Metriken

- `AbortedClients` – Verfolgt die Anzahl der Clientverbindungen, die nicht ordnungsgemäß geschlossen wurden.
- `AuroraSlowConnectionHandleCount` – Verfolgt die Anzahl langsamer Verbindungsverarbeitungsvorgänge und weist auf potenzielle Verbindungsprobleme oder Leistungsengpässe hin.
- `AuroraSlowHandshakeCount` – Misst die Anzahl langsamer Handshake-Vorgänge, was auch ein Hinweis auf Verbindungsprobleme sein kann.
- `ConnectionAttempts` – Misst die Anzahl der Verbindungsversuche zur Aurora MySQL-DB-Instance.

Globale Statusvariablen

`Aurora_external_connection_count` – Zeigt die Anzahl der Datenbankverbindungen mit der DB-Instance an, ausgenommen RDS-Serviceverbindungen, die für Datenbankzustandsprüfungen verwendet werden.

Durch die Überwachung dieser Metriken und globalen Statusvariablen erhalten Sie Einblick in die Verbindungsmuster, Fehler und potenziellen Engpässe, die Verbindungsprobleme mit Ihrer Amazon Aurora MySQL-Instance verursachen können.

Beispielsweise kann eine hohe Anzahl von `AbortedClients` oder `AuroraSlowConnectionHandleCount` auf Verbindungsprobleme hinweisen.

Darüber hinaus können Sie durch die Einrichtung von CloudWatch Alarmen und Benachrichtigungen Verbindungsprobleme proaktiv erkennen und darauf reagieren, bevor sie eskalieren und die Leistung Ihrer Anwendung beeinträchtigen.

Verbindungsfehlercodes für Aurora MySQL

Im Folgenden sind einige häufig auftretende Verbindungsfehler für Datenbanken von Aurora MySQL sowie deren Fehlercodes und Erklärungen aufgeführt.

Fehlercode 1040: Zu viele Verbindungen

Dieser Fehler tritt auf, wenn der Client versucht, mehr Verbindungen herzustellen, als der Datenbankserver maximal zulässt. Dafür gibt es die folgenden möglichen Ursachen:

- Fehlkonfiguration des Verbindungspoolings – Stellen Sie bei Verwendung eines Verbindungspooling-Mechanismus sicher, dass die maximale Poolgröße nicht zu hoch eingestellt ist und dass die Verbindungen wieder ordnungsgemäß zum Pool freigegeben werden.
- Konfiguration der Datenbank-Instance – Überprüfen Sie die Einstellung für die maximal zulässige Anzahl von Verbindungen für die Datenbank-Instance und passen Sie sie gegebenenfalls an, indem Sie den Parameter `max_connections` festlegen.
- Hohe Parallelität – Wenn mehrere Clients oder Anwendungen gleichzeitig eine Verbindung zur Datenbank herstellen, kann das Limit für die maximal zulässige Anzahl von Verbindungen erreicht werden.

Fehlercode 1045: Zugriff verweigert für Benutzer '...' @ '...' (mit Passwort: JA/NEIN)

Dieser Fehler weist auf einen Authentifizierungsfehler beim Versuch hin, eine Verbindung zur Datenbank herzustellen. Dafür gibt es die folgenden möglichen Ursachen:

- Kompatibilität des Authentifizierungs-Plugins – Überprüfen Sie, ob das vom Client verwendete Authentifizierungs-Plugin mit dem Authentifizierungsmechanismus des Datenbankservers kompatibel ist.

- Falscher Benutzername oder falsches Passwort – Stellen Sie sicher, dass der richtige Benutzername und das richtige Passwort in der Verbindungszeichenfolge oder im Authentifizierungsmechanismus verwendet werden.
- Benutzerberechtigungen – Stellen Sie sicher, dass der Benutzer über die erforderlichen Berechtigungen zum Herstellen einer Verbindung mit der Datenbank-Instance über den angegebenen Host oder das angegebene Netzwerk verfügt.

Fehlercode 1049: Unbekannte Datenbank '...'

Dieser Fehler gibt an, dass der Client versucht, eine Verbindung zu einer Datenbank herzustellen, die auf dem Server nicht vorhanden ist. Dafür gibt es die folgenden möglichen Ursachen:

- Datenbank nicht erstellt – Stellen Sie sicher, dass die angegebene Datenbank auf dem Datenbankserver erstellt wurde.
- Falscher Datenbankname – Überprüfen Sie den in der Verbindungszeichenfolge oder Abfrage verwendeten Datenbanknamen noch einmal auf Richtigkeit.
- Benutzerberechtigungen – Stellen Sie sicher, dass der Benutzer über die erforderlichen Berechtigungen für den Zugriff auf die angegebene Datenbank verfügt.

Fehlercode 1153: Paket mit mehr als 'max_allowed_packet' Bytes empfangen

Dieser Fehler tritt auf, wenn der Client versucht, Daten zu senden oder zu empfangen, die die vom Datenbankserver zugelassene maximale Paketgröße überschreiten. Dafür gibt es die folgenden möglichen Ursachen:

- Große Abfragen oder Ergebnismengen – Wenn Abfragen ausgeführt werden, die große Datenmengen beinhalten, kann das Paketgrößenlimit überschritten werden.
- Falsch konfigurierte Einstellungen für die Paketgröße – Überprüfen Sie die Einstellung `max_allowed_packet` auf dem Datenbankserver und passen Sie sie gegebenenfalls an.
- Probleme mit der Netzwerkkonfiguration – Stellen Sie sicher, dass die Netzwerkkonfiguration (z. B. die MTU-Größe) die erforderlichen Paketgrößen zulässt.

Fehlercode 1226: Benutzer '...' hat die Ressource 'max_user_connections' überschritten (aktueller Wert: ...)

Dieser Fehler weist darauf hin, dass der Benutzer die vom Datenbankserver zugelassene maximale Anzahl gleichzeitiger Verbindungen überschritten hat. Dafür gibt es die folgenden möglichen Ursachen:

- Fehlkonfiguration des Verbindungspoolings – Stellen Sie bei Verwendung eines Mechanismus für das Verbindungspooling sicher, dass die maximale Poolgröße für das Verbindungslimit des Benutzers nicht zu hoch festgelegt ist.

- Konfiguration der Datenbank-Instance – Überprüfen Sie die Einstellung `max_user_connections` für die Datenbank-Instance und passen Sie sie gegebenenfalls an.
- Hohe Parallelität – Wenn mehrere Clients oder Anwendungen gleichzeitig über denselben Benutzer eine Verbindung zur Datenbank herstellen, kann das benutzerspezifische Verbindungslimit erreicht werden.

Fehlercode 2003: Es kann keine Verbindung zum MySQL-Server auf '...' hergestellt werden (10061)

Dieser Fehler tritt normalerweise auf, wenn der Client keine TCP/IP Verbindung zum Datenbankserver herstellen kann. Er kann durch verschiedene Probleme verursacht werden, z. B. durch die folgenden:

- Status der Datenbank-Instance – Stellen Sie sicher, dass sich die Datenbank-Instance im Status `available`befindet und keine Wartungs- oder Sicherungsvorgänge durchgeführt werden.
- Firewall-Regeln – Überprüfen Sie, ob Firewalls (Betriebssystem, Netzwerk oder Sicherheitsgruppe) die Verbindung auf dem angegebenen Port blockieren (in der Regel 3306 für MySQL).
- Falscher Hostname oder falscher Endpunkt – Stellen Sie sicher, dass der in der Verbindungszeichenfolge verwendete Hostname oder Endpunkt korrekt ist und mit der Datenbank-Instance übereinstimmt.
- Probleme mit der Netzwerkkonnektivität – Stellen Sie sicher, dass der Client-Computer die Datenbank-Instance über das Netzwerk erreichen kann. Suchen Sie nach Netzwerkausfällen, Routing-Problemen oder VPC- bzw. Subnetz-Fehlkonfigurationen.

Fehlercode 2005: Unbekannter MySQL-Serverhost '...' (11001)

Dieser Fehler tritt auf, wenn der Client den Hostnamen oder den Endpunkt des Datenbankservers nicht in eine IP-Adresse auflösen kann. Dafür gibt es die folgenden möglichen Ursachen:

- Probleme mit der DNS-Auflösung – Stellen Sie sicher, dass der Client-Computer den Hostnamen mithilfe von DNS korrekt auflösen kann. Überprüfen Sie die DNS-Einstellungen und den DNS-Cache und versuchen Sie, die IP-Adresse anstelle des Hostnamens zu verwenden.
- Falscher Hostname oder falscher Endpunkt – Überprüfen Sie den in der Verbindungszeichenfolge verwendeten Hostnamen oder Endpunkt noch einmal auf Richtigkeit.
- Probleme mit der Netzwerkkonfiguration – Stellen Sie sicher, dass die Netzwerkkonfiguration des Clients (z. B. VPC, Subnetz und Routing-Tabellen) die DNS-Auflösung und Konnektivität zur Datenbank-Instance zulässt.

Fehlercode 2026: SSL-Verbindungsfehler

Dieser Fehler tritt auf, wenn während des Verbindungsversuchs ein Problem mit der SSL/TLS Konfiguration oder der Zertifikatsvalidierung auftritt. Dafür gibt es die folgenden möglichen Ursachen:

- Ablauf des Zertifikats — Prüfen Sie, ob das vom Server verwendete SSL/TLS Zertifikat abgelaufen ist und erneuert werden muss.
- Probleme bei der Zertifikatsvalidierung — Stellen Sie sicher, dass der Client das SSL/TLS Serverzertifikat korrekt validieren kann und ob das Zertifikat vertrauenswürdig ist.
- Probleme mit der Netzwerkkonfiguration — Stellen Sie sicher, dass die Netzwerkkonfiguration SSL/TLS Verbindungen zulässt und den SSL/TLS Handshake-Prozess nicht blockiert oder beeinträchtigt.
- SSL/TLS configuration mismatch – Make sure that the SSL/TLSDie Einstellungen (z. B. Cipher Suites und Protokollversionen) auf dem Client und dem Server sind kompatibel.

Wenn Sie die detaillierten Erklärungen und möglichen Ursachen für jeden Fehlercode verstehen, können Sie Verbindungsprobleme bei der Arbeit mit Datenbanken von Aurora MySQL besser beheben und lösen.

Empfehlungen zur Parameteroptimierung für Aurora MySQL

Maximale Anzahl der Verbindungen

Durch die Anpassung dieser Parameter können Verbindungsprobleme vermieden werden, die durch das Erreichen des maximal zulässigen Verbindungslimits verursacht werden. Stellen Sie sicher, dass diese Werte entsprechend den Parallelitätsanforderungen und Ressourcenbeschränkungen Ihrer Anwendung festgelegt sind.

- `max_connections` – Dieser Parameter gibt die maximale Anzahl an gleichzeitigen Verbindungen zur DB-Instance an.
- `max_user_connections` – Dieser Parameter kann bei der Benutzererstellung und -änderung angegeben werden und legt die maximale Anzahl gleichzeitiger Verbindungen fest, die für ein bestimmtes Benutzerkonto zulässig sind.

Größe des Netzwerkpuffers

Eine Erhöhung dieser Werte kann die Netzwerkleistung verbessern, insbesondere bei Workloads mit großen Datenübertragungen oder Ergebnismengen. Seien Sie jedoch vorsichtig, da größere Puffergrößen mehr Speicher verbrauchen können.

- `net_buffer_length` – Dieser Parameter legt die Anfangsgröße für Client-Verbindungen und Ergebnispuffer fest und sorgt so für ein ausgewogenes Verhältnis zwischen Speichernutzung und Abfrageleistung.
- `max_allowed_packet` – Dieser Parameter gibt die maximale Größe eines einzelnen Netzwerkpakets an, das von der DB-Instance gesendet oder empfangen werden kann.

Netzwerkkomprimierung (clientseitig)

Durch die Aktivierung der Netzwerkkomprimierung kann die Nutzung der Netzwerkbandbreite reduziert werden, sie kann jedoch den client- als auch den serverseitigen CPU-Overhead erhöhen.

- `compress`— Dieser Parameter aktiviert oder deaktiviert die Netzwerkkomprimierung für die Kommunikation. client/server
- `compress_protocol` – Dieser Parameter gibt das Komprimierungsprotokoll an, das für die Netzwerkkommunikation verwendet werden soll.

Optimierung der Netzwerkleistung

Die Anpassung dieser Timeouts kann dazu beitragen, inaktive Verbindungen zu verwalten und eine Erschöpfung der Ressourcen zu verhindern. Gehen Sie mit Bedacht vor, da niedrige Werte zu vorzeitigen Verbindungsabbrüchen führen können.

- `interactive_timeout` – Dieser Parameter gibt an, wie viele Sekunden der Server auf Aktivitäten in einer interaktiven Verbindung wartet, bevor er sie schließt.
- `wait_timeout` – Dieser Parameter bestimmt die Anzahl der Sekunden, die der Server auf Aktivität für eine nicht interaktive Verbindung wartet, bevor er sie schließt.

Einstellungen für Netzwerk-Timeouts

Durch die Anpassung dieser Timeouts können Probleme im Zusammenhang mit langsamen oder nicht reagierenden Verbindungen behoben werden. Achten Sie jedoch darauf, sie nicht zu niedrig einzustellen, da dies zu vorzeitigen Verbindungsausfällen führen kann.

- `net_read_timeout` – Dieser Parameter gibt an, wie viele Sekunden auf weitere Daten von einer Verbindung gewartet werden soll, bevor der Lesevorgang beendet wird.
- `net_write_timeout` – Dieser Parameter bestimmt, wie viele Sekunden gewartet werden soll, bis ein Block in eine Verbindung geschrieben wird, bevor der Schreibvorgang beendet wird.

Beispiele für die Behebung von Datenbankverbindungsproblemen für Aurora MySQL

Die folgenden Beispiele zeigen, wie Datenbankverbindungsprobleme für Aurora MySQL identifiziert und behoben werden können.

Beispiel 1: Fehlerbehebung bei fehlgeschlagenen Verbindungsversuchen

Verbindungsversuche können aus verschiedenen Gründen fehlschlagen, darunter Authentifizierungsfehler, SSL/TLS Handshake-Fehler, erreichtes `max_connections` Limit und Ressourcenbeschränkungen für die DB-Instance.

Sie können die Anzahl der fehlgeschlagenen Verbindungen entweder in Performance Insights oder mit dem folgenden Befehl verfolgen.

```
mysql> show global status like 'aborted_connects';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| Aborted_connects | 7     |
+------------------+-------+
1 row in set (0.00 sec)
```

Wenn sich die Anzahl der `Aborted_connects` im Laufe der Zeit erhöht, kann es bei der Anwendung zu zeitweiligen Verbindungsproblemen kommen.

Sie können Advanced Auditing von Aurora verwenden, um die hergestellten und getrennten Client-Verbindungen zu protokollieren. Legen Sie hierzu die folgenden Parameter in der DB-Cluster-Parametergruppe fest:

- `server_audit_logging` = `1`
- `server_audit_events` = `CONNECT`

Im Folgenden finden Sie einen Auszug aus den Prüfprotokollen für eine fehlgeschlagene Anmeldung.

```
1728498527380921,auora-mysql-node1,user_1,172.31.49.222,147189,0,FAILED_CONNECT,,,1045
1728498527380940,auora-mysql-node1,user_1,172.31.49.222,147189,0,DISCONNECT,,,0
```

Wobei Folgendes gilt:

- `1728498527380921` – Der Epochenzeitstempel für den Zeitpunkt der fehlgeschlagene Anmeldung

- `aurora-mysql-node1` – Die Instance-ID des Knotens des Aurora MySQL-Clusters, auf dem die Verbindung fehlgeschlagen ist
- `user_1` – Der Name des Datenbankbenutzers, für den die Anmeldung fehlgeschlagen ist
- `172.31.49.222` – Die private IP-Adresse des Clients, über den die Verbindung hergestellt wurde
- `147189` – Die Verbindungs-ID der fehlgeschlagenen Anmeldung
- `FAILED_CONNECT` – Zeigt an, dass die Verbindung fehlgeschlagen ist
- `1045` – Der Rückgabecode Ein Wert ungleich Null weist auf einen Fehler hin. In diesem Fall entspricht `1045` der Zugriffsverweigerung.

Weitere Informationen finden Sie unter Server error codes und Client error codes in der MySQL-Dokumentation.

Sie können die Aurora MySQL-Fehlerprotokolle auch auf verwandte Fehlermeldungen untersuchen, zum Beispiel:

```
2024-10-09T19:26:59.310443Z 220 [Note] [MY-010926] [Server] Access denied for user
 'user_1'@'172.31.49.222' (using password: YES) (sql_authentication.cc:1502)
```

Beispiel 2: Behebung ungewöhnlicher Trennungen von Client-Verbindungen

Sie können die Anzahl ungewöhnlicher Client-Verbindungsabbrüche entweder in Performance Insights oder mit dem folgenden Befehl verfolgen.

```
mysql> show global status like 'aborted_clients';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| Aborted_clients | 9     |
+-----------------+-------+
1 row in set (0.01 sec)
```

Wenn die Anzahl der `Aborted_clients` im Laufe der Zeit zunimmt, schließt die Anwendung die Verbindungen zur Datenbank nicht ordnungsgemäß. Wenn Verbindungen nicht ordnungsgemäß geschlossen werden, kann dies zu Ressourcenlecks und potenziellen Leistungsproblemen führen. Wenn Verbindungen unnötigerweise geöffnet bleiben, können Systemressourcen wie Speicher und Dateideskriptoren verbraucht werden, was letztendlich dazu führen kann, dass die Anwendung oder der Server nicht mehr reagiert oder neu gestartet wird.

Mit der folgenden Abfrage können Sie Konten identifizieren, die Verbindungen nicht ordnungsgemäß schließen. Sie ruft den Namen des Benutzerkontos, den Host, über den der Benutzer eine Verbindung herstellt, die Anzahl der nicht geschlossenen Verbindungen und den Prozentsatz der nicht geschlossenen Verbindungen ab.

```
SELECT
    ess.user,
    ess.host,
    (a.total_connections - a.current_connections) - ess.count_star AS not_closed,
    (((a.total_connections - a.current_connections) - ess.count_star) * 100) /
 (a.total_connections - a.current_connections) AS pct_not_closed
FROM
    performance_schema.events_statements_summary_by_account_by_event_name AS ess
    JOIN performance_schema.accounts AS a ON (ess.user = a.user AND ess.host = a.host)
WHERE
    ess.event_name = 'statement/com/quit'
    AND (a.total_connections - a.current_connections) > ess.count_star;

+----------+---------------+------------+-----------------+
| user     | host          | not_closed | pct_not_closed |
+----------+---------------+------------+-----------------+
| user1    | 172.31.49.222 |          1 |        33.3333 |
| user1    | 172.31.93.250 |       1024 |        12.1021 |
| user2    | 172.31.93.250 |         10 |        12.8551 |
+----------+---------------+------------+-----------------+
3 rows in set (0.00 sec)
```

Nachdem Sie die Benutzerkonten und Hosts identifiziert haben, von denen aus die Verbindungen nicht geschlossen werden, können Sie den Code überprüfen, der die Verbindungen nicht ordnungsgemäß schließt.

Verwenden Sie beispielsweise mit dem MySQL-Konnektor in Python die Methode `close()` des Verbindungsobjekts, um Verbindungen zu schließen. Hier finden Sie eine Beispielfunktion, die eine Verbindung zu einer Datenbank herstellt, eine Abfrage durchführt und die Verbindung schließt:

```
import mysql.connector

def execute_query(query):
    # Establish a connection to the database
    connection = mysql.connector.connect(
        host="your_host",
        user="your_username",
```

```
        password="your_password",
        database="your_database"
    )

    try:
        # Create a cursor object
        cursor = connection.cursor()

        # Execute the query
        cursor.execute(query)

        # Fetch and process the results
        results = cursor.fetchall()
        for row in results:
            print(row)

    finally:
        # Close the cursor and connection
        cursor.close()
        connection.close()
```

In diesem Beispiel wird die Methode `connection.close()` im Block `finally` aufgerufen, um sicherzustellen, dass die Verbindung geschlossen wird, unabhängig davon, ob eine Ausnahme auftritt oder nicht.

Beispiel 3: Problembehandlung bei fehlgeschlagenen IAM-Verbindungsversuchen

Die Konnektivität mit AWS Identity and Access Management (IAM) -Benutzern kann aus verschiedenen Gründen fehlschlagen, z. B.:

- Falsche IAM-Richtlinienkonfiguration
- Abgelaufene Sicherheitsnachweise
- Probleme mit der Netzwerkkonnektivität
- Die Datenbankberechtigungen stimmen nicht überein

Um diese Authentifizierungsfehler zu beheben, aktivieren `iam-db-auth-error` Sie die Protokollexportfunktion in Ihrer Amazon Relational Database Service (RDS) - oder Aurora-Datenbank. Auf diese Weise können Sie detaillierte Authentifizierungsfehlermeldungen in der CloudWatch Protokollgruppe für Ihren Amazon RDS- oder Amazon Aurora Aurora-Cluster anzeigen.

Nach der Aktivierung können Sie diese Protokolle überprüfen, um die spezifische Ursache Ihrer IAM-Authentifizierungsfehler zu ermitteln und zu beheben.

Zum Beispiel:

```
2025-09-22T12:02:30,806 [ERROR] Failed to authorize the connection request for
 user 'user_1' due to an internal IAM DB Auth error. (Status Code: 500, Error Code:
 InternalError)
```

und

```
2025-09-22T12:02:51,954 [ERROR] Failed to authenticate the connection request for user
 'user_2' because the provided token is malformed or otherwise invalid. (Status Code:
 400, Error Code: InvalidToken)
```

Anleitungen zur Fehlerbehebung finden Sie im Aurora-Leitfaden zur Fehlerbehebung für die IAM-DB-Authentifizierung.

Beheben von Problemen mit der Abfrageleistung bei Aurora-MySQL-Datenbanken

MySQL ermöglicht eine Abfrageoptimiererkontrolle über Systemvariablen, die sich auf die Auswertung von Abfrageplänen, umschaltbare Optimierungen, Optimierer- und Indexhinweise sowie das Optimiererkostenmodell auswirken. Diese Datenpunkte können nicht nur beim Vergleich verschiedener MySQL-Umgebungen hilfreich sein, sondern auch, um frühere Abfrageausführungspläne mit aktuellen Ausführungsplänen zu vergleichen und die Gesamtausführung einer MySQL-Abfrage zu jedem Zeitpunkt zu verstehen.

Die Abfrageleistung hängt von vielen Faktoren ab. Dazu gehören der Ausführungsplan, das Schema und die Größe der Tabelle, die Statistiken, die Ressourcen, die Indizes und die Parameterkonfiguration. Die Abfrageoptimierung erfordert die Identifizierung von Engpässen und die Optimierung des Ausführungspfads.

- Suchen Sie den Ausführungsplan für die Abfrage und überprüfen Sie, ob die Abfrage geeignete Indizes verwendet. Sie können Ihre Abfrage optimieren, indem Sie `EXPLAIN` verwenden und die Details der einzelnen Pläne überprüfen.
- Aurora-MySQL-Version 3 (kompatibel mit MySQL 8.0 Community Edition) verwendet eine `EXPLAIN ANALYZE`-Anweisung. Die `EXPLAIN ANALYZE`-Anweisung ist ein Profiling-Tool, das anzeigt, wo MySQL Zeit für Ihre Anfrage aufwendet und warum. Mit `EXPLAIN ANALYZE` plant Aurora MySQL die Abfrage, bereitet sie vor und führt sie aus, während Zeilen gezählt werden und die an verschiedenen Punkten des Ausführungsplans aufgewendete Zeit gemessen wird. Wenn die Abfrage abgeschlossen ist, druckt `EXPLAIN ANALYZE` den Plan und die Messwerte anstelle des Abfrageergebnisses.
- Halten Sie Ihre Schemastatistiken mithilfe der `ANALYZE`-Anweisung auf dem neuesten Stand. Der Abfrageoptimierer wählt aufgrund veralteter Statistiken manchmal schlechte Ausführungspläne aus. Dies kann aufgrund ungenauer Kardinalitätsschätzungen von Tabellen und Indizes zu einer schlechten Abfrageleistung führen. Die `last_update`-Spalte der Tabelle innodb_table_stats zeigt an, wann Ihre Schemastatistiken zuletzt aktualisiert wurden. Dies ist ein guter „Alterungs“-Indikator.
- Weitere Probleme können auftreten, beispielsweise eine ungleiche Datenverteilung, die bei der Tabellenkardinalität nicht berücksichtigt wird. Weitere Informationen finden Sie unter Schätzen der ANALYZE-TABLE-Komplexität für InnoDB-Tabellen und Histogrammstatistiken in MySQL in der MySQL-Dokumentation.

Verstehen des Zeitaufwands bei Abfragen

Es gibt folgende Möglichkeiten, den Zeitaufwand für Abfragen zu ermitteln:

- Profilerstellung
- Leistungsschema
- Abfragenoptimierer

Profilerstellung

Die Profilerstellung ist standardmäßig deaktiviert. Aktivieren Sie die Profilerstellung, führen Sie dann die langsame Abfrage aus und überprüfen Sie das zugehörige Profil.

```
SET profiling = 1;
Run your query.
SHOW PROFILE;
```

1. Identifizieren Sie die Phase, die die meiste Zeit beansprucht. Gemäß den allgemeinen Thread-Zuständen aus der MySQL-Dokumentation ist das Lesen und Verarbeiten von Zeilen für eine `SELECT`-Anweisung häufig der am längsten anhaltende Zustand während der Lebensdauer einer bestimmten Abfrage. Sie können die `EXPLAIN`-Anweisung verwenden, um zu verstehen, wie MySQL diese Abfrage ausführt.
2. Sehen Sie sich das Slow-Query-Protokoll an, um `rows_examined` und `rows_sent` zu bewerten und sicherzustellen, dass der Workload in jeder Umgebung ähnlich ist. Weitere Informationen finden Sie unter Protokollieren bei Aurora-MySQL-Datenbanken.
3. Führen Sie den folgenden Befehl für Tabellen aus, die Teil der identifizierten Abfrage sind:

   ```
   SHOW TABLE STATUS\G;
   ```

4. Erfassen Sie die folgenden Ausgaben vor und nach der Ausführung der Abfrage in jeder Umgebung:

   ```
   SHOW GLOBAL STATUS;
   ```

5. Führen Sie die folgenden Befehle in jeder Umgebung aus, um festzustellen, ob andere Abfragen/Sitzungen die Leistung dieser Beispielabfrage beeinflussen.

   ```
   SHOW FULL PROCESSLIST;
   ```

```
SHOW ENGINE INNODB STATUS\G;
```

Wenn die Ressourcen auf dem Server ausgelastet sind, wirkt sich dies manchmal auf alle anderen Vorgänge aus, auch auf Abfragen. Sie können auch regelmäßig Informationen erfassen, wenn Abfragen ausgeführt werden, oder einen `cron`-Job einrichten, um Informationen in sinnvollen Intervallen zu erfassen.

Leistungsschema

Das Leistungsschema bietet nützliche Informationen über die Leistung der Serverlaufzeit und hat gleichzeitig nur minimale Auswirkungen auf diese Leistung. Dies unterscheidet sich vom `information_schema`, welches Schemainformationen über die DB-Instance bereitstellt. Weitere Informationen finden Sie unter Übersicht über das Leistungsschema für Performance Insights auf Aurora MySQL.

Ablaufverfolgung des Abfrageoptimierers

Um zu verstehen, warum ein bestimmter Abfrageplan für die Ausführung ausgewählt wurde, können Sie `optimizer_trace` einrichten, um auf den MySQL-Abfrageoptimierer zuzugreifen.

Führen Sie eine Optimierer-Ablaufverfolgung aus, um ausführliche Informationen zu allen Pfaden zu erhalten, die dem Optimierer zur Verfügung stehen, und zu dessen Wahl.

```
SET SESSION OPTIMIZER_TRACE="enabled=on";
SET optimizer_trace_offset=-5, optimizer_trace_limit=5;

-- Run your query.
SELECT * FROM table WHERE x = 1 AND y = 'A';

-- After the query completes:
SELECT * FROM information_schema.OPTIMIZER_TRACE;
SET SESSION OPTIMIZER_TRACE="enabled=off";
```

Überprüfen der Einstellungen des Abfrageoptimierers

Aurora-MySQL-Version 3 (kompatibel mit MySQL 8.0 Community Edition) weist im Vergleich zu Aurora-MySQL-Version 2 (kompatibel mit MySQL 5.7 Community Edition) viele Optimierer-bezogene Änderungen auf. Wenn Sie einige benutzerdefinierte Werte für `optimizer_switch` festgelegt haben, empfehlen wir Ihnen, die Unterschiede in den Standardeinstellungen zu überprüfen und

`optimizer_switch`-Werte festzulegen, die für Ihren Workload am besten geeignet sind. Wir empfehlen Ihnen außerdem, die für Aurora-MySQL-Version 3 verfügbaren Optionen zu testen, um die Leistung Ihrer Abfragen zu überprüfen.

> **Note**
>
> Aurora-MySQL-Version 3 verwendet den Community-Standardwert 20 für den innodb_stats_persistent_sample_pages-Parameter.

Mit dem folgenden Befehl können Sie die `optimizer_switch`-Werte anzeigen:

```
SELECT @@optimizer_switch\G;
```

Die folgende Tabelle zeigt die standardmäßigen `optimizer_switch`-Werte für die Versionen 2 und 3 von Aurora.

Einstellung	Aurora-MySQL-Version 2	Aurora-MySQL-Version 3
batched_key_access	aus	aus
block_nested_loop	on	on
condition_fanout_filter	on	on
derived_condition_pushdown	–	on
derived_merge	on	on
duplicateweedout	on	on
engine_condition_pushdown	on	on
firstmatch	on	on
hash_join	aus	on
hash_join_cost_based	on	–
hypergraph_optimizer	–	aus

Einstellung	Aurora-MySQL-Version 2	Aurora-MySQL-Version 3
index_condition_pushdown	on	on
index_merge	on	on
index_merge_intersection	on	on
index_merge_sort_union	on	on
index_merge_union	on	on
loosescan	on	on
materialization	on	on
mrr	on	on
mrr_cost_based	on	on
prefer_ordering_index	on	on
semijoin	on	on
skip_scan	–	on
subquery_materialization_cost_based	on	on
subquery_to_derived	–	aus
use_index_extensions	on	on
use_invisible_indexes	–	aus

Weitere Informationen finden Sie unter Umschaltbare Optimierungen (MySQL 5.7) und Umschaltbare Optimierungen (MySQL 8.0) in der MySQL-Dokumentation.

Amazon Aurora MySQL-Referenz

Diese Referenz enthält Informationen über Aurora MySQL-Parameter, Statusvariablen und allgemeine SQL-Erweiterungen oder Unterschiede zur MySQL-Datenbank-Engine der Community.

Themen

- Aurora MySQL Konfigurationsparameter
- Globale Statusvariablen von Aurora MySQL
- Aurora-MySQL-Warteereignisse
- Aurora-MySQL-Thread-Zustände
- Aurora MySQL-Isolierungsstufen
- Aurora-MySQL-Hinweise
- Referenz für gespeicherte Aurora-MySQL-Prozeduren
- Aurora-MySQL-spezifische information_schema-Tabellen

Aurora MySQL Konfigurationsparameter

Sie können Ihr Amazon-Aurora-MySQL-DB-Cluster auf dieselbe Art und Weise verwalten, wie Sie Ihre anderen Amazon-RDS-DB-Instances verwalten, und zwar indem Sie die Parameter in einer DB-Parametergruppe verwenden. Amazon Aurora unterscheidet sich von anderen DB-Engines dadurch, dass Sie über einen DB-Cluster verfügen, der mehrere DB-Instances enthält. Folglich gelten einige der Parameter, mit denen Sie Ihren Aurora MySQL-DB-Cluster verwalten, für den gesamten Cluster. Andere Parameter gelten nur für eine bestimmte DB-Instance im DB-Cluster.

Um Parameter auf Cluster-Ebene zu verwalten, verwenden Sie DB-Cluster-Parametergruppen. Um Parameter auf Instance-Ebene zu verwalten, verwenden Sie DB-Parametergruppen. Jede DB-Instance in einem Aurora MySQL-DB-Cluster ist mit der MySQL-Datenbank-Engine kompatibel. Sie wenden jedoch einige der Parameter der MySQL-Datenbank-Engine auf Cluster-Ebene an und verwalten diese Parameter mit DB-Cluster-Parametergruppen. Sie können keine Parameter auf Cluster-Ebene in der DB-Parametergruppe für eine Instance in einem Aurora-DB-Cluster finden. Eine Liste der Parameter auf Cluster-Ebene erscheint weiter unten in diesem Thema.

Sie können die Parameter sowohl auf der Cluster- als auch auf der Instance-Ebene mithilfe der AWS-Managementkonsole, AWS CLI oder Amazon-RDS-API verwalten. Sie verwenden separate Befehle für die Verwaltung von Parametern auf Cluster- und Instance-Ebene. Beispielsweise können Sie mit

dem CLI-Befehl modify-db-cluster-parameter-group Parameter auf Cluster-Ebene in einer DB-Cluster-Parametergruppe verwalten. Sie können den CLI-Befehl modify-db-parameter-group verwenden, um Parameter auf Instance-Ebene in einer DB-Parametergruppe für eine DB-Instance in einem DB-Cluster zu verwalten.

Sie können Parameter auf Cluster- und Instance-Ebene in der Konsole oder mithilfe der CLI oder der RDS-API anzeigen. Beispielsweise können Sie mit den AWS CLI-Befehl describe-db-cluster-parameters verwenden, um Parameter auf Cluster-Ebene in einer DB-Cluster-Parametergruppe anzuzeigen. Sie können den CLI-Befehl describe-db-parameters verwenden, um Parameter auf Instance-Ebene in einer DB-Parametergruppe für eine DB-Instance in einem DB-Cluster anzuzeigen.

> **Note**
>
> Jede Standard-Parametergruppe enthält die Standardwerte für alle Parameter in der Parametergruppe. Wenn der Parameter „Engine-Standard“ für diesen Wert hat, finden Sie in der versionsspezifischen MySQL- oder PostgreSQL-Dokumentation den tatsächlichen Standardwert.
> Sofern nicht anders angegeben, gelten die in den folgenden Tabellen aufgeführten Parameter für die Aurora-MySQL-Versionen 2 und 3.

Weitere Informationen zu DB-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora. Regeln und Einschränkungen für Aurora Serverless v1-Cluster finden Sie unter Parametergruppen für Aurora Serverless v1.

Themen

- Parameter auf Cluster-Ebene
- Parameter auf Instance-Ebene
- MySQL-Parameter, die nicht auf Aurora MySQL zutreffen

Parameter auf Cluster-Ebene

In der folgenden Tabelle werden alle Parameter aufgeführt, die für das gesamte Aurora MySQL-DB-Cluster gelten.

Parametername	Anpassbar	Hinweise
`aurora_binlog_read_buffer_size`	Ja	Beeinflusst nur Cluster, die Binärprotokoll-Replikation (Binlog) verwenden. Hinweise zur Binärprotokoll-Replikation finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation). Aus Aurora-MySQL-Version 3 entfernt.
`aurora_binlog_replication_max_yield_seconds`	Ja	Beeinflusst nur Cluster, die Binärprotokoll-Replikation (Binlog) verwenden. Hinweise zur Binärprotokoll-Replikation finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).
`aurora_binlog_replication_sec_index_parallel_workers`	Ja	Legt die Gesamtzahl der verfügbaren parallelen Threads fest, um sekundäre Indexänderungen anzuwenden, wenn Transaktionen für große Tabellen mit mehr als einem sekundären Index repliziert werden. Der Parameter ist standardmäßig auf `0` (deaktiviert) eingestellt. Dieser Parameter ist in Aurora-MySQL-Version 3.06 und höher verfügbar. Weitere Informationen finden Sie unter Optimieren einer binären Protokoll replikation für Aurora MySQL.
`aurora_binlog_use_large_read_buffer`	Ja	Beeinflusst nur Cluster, die Binärprotokoll-Replikation (Binlog) verwenden. Hinweise zur Binärprotokoll-Replikation

Parametername	Anpassbar	Hinweise
		finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation). Aus Aurora-MySQL-Version 3 entfernt.
`aurora_disable_hash_join`	Ja	Setzen Sie diesen Parameter auf `ON`, um die Hash-Join-Optimierung in Aurora MySQL Version 2.09 oder höher zu deaktivieren. Für Version 3 wird dies nicht unterstützt. Weitere Informati onen finden Sie unter Parallel Query für Amazon Aurora MySQL.
`aurora_enable_replica_log_compression`	Ja	Weitere Informationen finden Sie unter Performance-Überlegungen zur Amazon Aurora MySQL-Replikation. Gilt nicht für Cluster, die Teil einer globalen Aurora-Datenbank sind. Aus Aurora-MySQL-Version 3 entfernt.
`aurora_enable_repl_bin_log_filtering`	Ja	Weitere Informationen finden Sie unter Performance-Überlegungen zur Amazon Aurora MySQL-Replikation. Gilt nicht für Cluster, die Teil einer globalen Aurora-Datenbank sind. Aus Aurora-MySQL-Version 3 entfernt.
`aurora_enable_staggered_replica_restart`	Ja	Diese Einstellung ist in Aurora MySQL Versione 3 verfügbar, wird aber nicht verwendet.
`aurora_enable_zdr`	Ja	Diese Einstellung ist standardmäßig in Aurora MySQL 2.10 und höher aktiviert.

Parametername	Anpassbar	Hinweise
`aurora_in_memory_relaylog`	Ja	Legt den In-Memory-Relay-Protokollmodus fest. Sie können diese Funktion für Binärprotokollreplikate verwenden, um die Leistung der Binärprotokollreplikation zu verbessern. Zum Ausschalten dieser Funktion setzen Sie den Parameter auf OFF. Zum Einschalten dieser Funktion setzen Sie den Parameter auf ON.
`aurora_enhanced_binlog`	Ja	Legen Sie den Wert dieses Parameters auf 1 fest, um das erweiterte Binärprotokoll in Aurora MySQL Version 3.03.1 und höher zu aktivieren. Weitere Informationen finden Sie unter Einrichten eines erweiterten Binärprotokolls für Aurora MySQL.
`aurora_full_double_precision_in_json`	Ja	Stellen Sie den Wert dieses Parameters ein, um das Parsing von Fließkommazahlen in JSON-Dokumenten mit voller Genauigkeit zu ermöglichen.
`aurora_jemalloc_background_thread`	Ja	Verwenden Sie diesen Parameter, um einen Hintergrund-Thread zur Durchführung von Speicherwartungsoperationen zu aktivieren. Die zulässigen Werte sind `0` (deaktiviert) und 1 (aktiviert). Der Standardwert ist `0`. Dieser Parameter gilt für Aurora MySQL Version 3.05 und höher.

Parametername	Anpassbar	Hinweise
`aurora_jemalloc_dirty_decay_ms`	Ja	Verwenden Sie diesen Parameter zur Beibehaltung von freigegebenem Speicher für eine bestimmte Zeit (in Millisekunden). Die Beibehaltung von Speicher ermöglicht eine schnellere Wiederverwendung. Die zulässigen Werte sind `0` bis `18446744073709551615`. Der Standardwert ist `10000` (10 Sekunden). Eine kürzere Verzögerung kann helfen, Speicherprobleme zu vermeiden, führt jedoch zu einer geringeren Leistung. Dieser Parameter gilt für Aurora MySQL Version 3.05 und höher.
`aurora_jemalloc_tcache_enabled`	Ja	Verwenden Sie diesen Parameter, um kleine Speicheranforderungen (bis zu 32 KiB) in einem lokalen Thread-Cache zu bearbeiten und dabei die Speicherarenen zu umgehen. Die zulässigen Werte sind `0` (deaktiviert) und 1 (aktiviert). Der Standardwert ist 1. Dieser Parameter gilt für Aurora MySQL Version 3.05 und höher.
`aurora_load_from_s3_role`	Ja	Weitere Informationen finden Sie unter Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket. Derzeit nicht in Aurora-MySQL-Version 3 verfügbar. Verwenden Sie `aws_default_s3_role`.

Parametername	Anpassbar	Hinweise
`aurora_mask_password_hashes_type`	Ja	Diese Einstellung ist in Aurora MySQL 2.11 und höher standardmäßig aktiviert. Verwenden Sie diese Einstellung, um Passwort-Hashes von Aurora MySQL in den allgemeinen Protokollen, den langsamen Abfrageprotokollen und den Audit-Protokollen zu maskieren . Die zulässigen Werte sind `0` und 1 (Standardeinstellung). Wenn diese Einstellung auf 1 festgelegt ist, werden Passwörter als `<secret>` protokolliert. Wenn diese Einstellung auf `0` festgeleg t ist, werden Passwörter als Hash-Werte (#) protokolliert.
`aurora_select_into_s3_role`	Ja	Weitere Informationen finden Sie unter Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket. Derzeit nicht in Aurora-MySQL-Version 3 verfügbar. Verwenden Sie `aws_default_s3_role` .

Parametername	Anpassbar	Hinweise
`authentication_kerberos_caseins_cmp`	Ja	Steuert den Benutzernamenvergleich ohne Berücksichtigung der Groß-/Kleinschreibung für das Plug-in `authentication_kerberos` . Setzen Sie den Parameter auf `true`, um den Vergleich ohne Berücksichtigung der Groß-/Kleinschreibung zu aktivieren. Standardmäßig wird der Vergleich unter Berücksichtigung der Groß-/Kleinschreibung verwendet (`false`). Weitere Informationen finden Sie unter Verwenden der Kerberos-Authentifizierung für Aurora MySQL. Dieser Parameter ist in Aurora MySQL Version 3.03 und höher verfügbar.
`auto_increment_increment`	Ja	Keine
`auto_increment_offset`	Ja	Keine
`aws_default_lambda_role`	Ja	Weitere Informationen finden Sie unter Aufrufen einer Lambda-Funktion aus einem Amazon Aurora MySQL-DB-Cluster.

Parametername	Anpassbar	Hinweise
`aws_default_s3_role`	Ja	Wird beim Aufrufen der Anweisung `LOAD DATA FROM S3`, `LOAD XML FROM S3` oder `SELECT INTO OUTFILE S3` aus Ihrem DB-Cluster verwendet. Bei Aurora MySQL Version 2 wird die in diesem Parameter festgelegte IAM-Rolle verwendet, wenn keine IAM-Rolle für `aurora_load_from_s3_role` oder `aurora_select_into_s3_role` für die entsprechende Anweisung festgelegt ist. Bei Aurora-MySQL-Version 3 wird die für diesen Parameter festgelegte IAM-Rolle immer verwendet. Weitere Informationen finden Sie unter Zuweisen einer IAM-Rolle zu einem Amazon-Aurora-MySQL-DB-Cluster.
`binlog_backup`	Ja	Legen Sie den Wert dieses Parameters auf 0 fest, um das erweiterte Binärprotokoll in Aurora MySQL Version 3.03.1 und höher zu aktivieren. Sie können diesen Parameter nur deaktivieren, wenn Sie das erweiterte Binärprotokoll verwenden. Weitere Informationen finden Sie unter Einrichten eines erweiterten Binärprotokolls für Aurora MySQL.

Parametername	Anpassbar	Hinweise
binlog_checksum	Ja	Die AWS CLI und RDS-API melden einen Wert von None, wenn dieser Parameter nicht festgelegt ist. In diesem Fall verwendet Aurora MySQL den Standardwert für die Engine, d.h. CRC32. Dies ist anders als die explizite Einstellung von NONE, die die Prüfsumme ausschaltet.
binlog-do-db	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
binlog_format	Ja	Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).
binlog_group_commit_sync_delay	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
binlog_group_commit_sync_no_delay_count	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
binlog-ignore-db	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.

Parametername	Anpassbar	Hinweise
binlog_replication_globaldb	Ja	Legen Sie den Wert dieses Parameters auf 0 fest, um das erweiterte Binärprotokoll in Aurora MySQL Version 3.03.1 und höher zu aktivieren. Sie können diesen Parameter nur deaktivieren, wenn Sie das erweiterte Binärprotokoll verwenden. Weitere Informationen finden Sie unter Einrichten eines erweiterten Binärprotokolls für Aurora MySQL.
binlog_row_image	Nein	Keine
binlog_row_metadata	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
binlog_row_value_options	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
binlog_rows_query_log_events	Ja	Keine
binlog_transaction_compression	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
binlog_transaction_compression_level_zstd	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.

Parametername	Anpassbar	Hinweise
binlog_transaction_dependency_history_size	Ja	Dieser Parameter legt eine Obergrenze für die Anzahl der Zeilen-Hashes fest, die im Speicher abgelegt und für die Suche nach der Transaktion, die eine bestimmte Zeile zuletzt geändert hat, verwendet werden. Sobald diese Anzahl von Hashes erreicht wurde, wird der Verlauf gelöscht. Dieser Parameter gilt für Aurora-MySQL-Version 2.12 und höher sowie Version 3.
binlog_transaction_dependency_tracking	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
character-set-client-handshake	Ja	Keine
character_set_client	Ja	Keine
character_set_connection	Ja	Keine
character_set_database	Ja	Der von der Standarddatenbank verwendete Zeichensatz. Der Standardwert ist utf8mb4.
character_set_filesystem	Ja	Keine
character_set_results	Ja	Keine
character_set_server	Ja	Keine
collation_connection	Ja	Keine
collation_server	Ja	Keine
completion_type	Ja	Keine

Parametername	Anpassbar	Hinweise
`default_storage_engine`	Nein	Aurora MySQL-Cluster verwenden für all Ihre Daten die InnoDB-Speicher-Engine.
`enforce_gtid_consistency`	Manchmal	Veränderbar in Aurora MySQL-Version 2 und höher.
`event_scheduler`	Ja	Zeigt den Status des Ereignisplaners an. Modifizierbar nur auf Cluster-Ebene in Aurora-MySQL-Version 3.
`gtid-mode`	Manchmal	Veränderbar in Aurora MySQL-Version 2 und höher.
`information_schema_stats_expiry`	Ja	Die Anzahl der Sekunden, nach denen der MySQL-Datenbankserver Daten von der Speicher-Engine abruft und die Daten im Cache ersetzt. Die zulässigen Werte sind `0` bis `31536000`. Dieser Parameter gilt für Aurora MySQL Version 3.

Parametername	Anpassbar	Hinweise
`init_connect`	Ja	Der Befehl, der vom Server für jeden Client ausgeführt wird, der eine Verbindung herstellt. Verwenden Sie doppelte Anführungszeichen („) für Einstellungen, um Verbindungsfehler zu vermeiden, zum Beispiel: `SET optimizer_switch="hash_join =off"` Bei Aurora MySQL Version 3 gilt dieser Parameter nicht für Benutzer, die über die `CONNECTION_ADMIN` -Berechtigung verfügen. Dies schließt den Aurora-Hauptbenutzer ein. Weitere Informationen finden Sie unter Rollenbasiertes Berechtigungsmodell.
`innodb_adaptive_hash_index`	Ja	Sie können diesen Parameter auf DB-Cluster-Ebene in den Aurora-MySLQ-Versionen 2 und 3 ändern. Der Adaptive-Hash-Index wird auf Reader-DB-Instances nicht unterstützt.

Parametername	Anpassbar	Hinweise
innodb_aurora_instant_alter_column_allowed	Ja	Steuert, ob der INSTANT-Algorithmus für ALTER COLUMN-Operationen auf globaler Ebene verwendet werden kann. Die zulässigen Werte sind folgende: • 0 – Der INSTANT-Algorithmus ist für ALTER COLUMN-Operationen (OFF) nicht zulässig. Kehrt zu anderen Algorithmen zurück. • 1 – Der INSTANT-Algorithmus ist für ALTER COLUMN-Operationen (ON) zulässig. Dies ist der Standardwert. Weitere Informationen finden Sie unter Column Operations in der MySQL-Dokumentation. Dieser Parameter gilt für Aurora MySQL Version 3.05 und höher.
innodb_autoinc_lock_mode	Ja	Keine
innodb_checksums	Nein	Aus Aurora-MySQL-Version 3 entfernt.
innodb_cmp_per_index_enabled	Ja	Keine
innodb_commit_concurrency	Ja	Keine
innodb_data_home_dir	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.

Parametername	Anpassbar	Hinweise
`innodb_deadlock_detect`	Ja	Diese Option wird verwendet, um die Deadlock-Erkennung in Aurora-MySQL-Version 2.11 und höher sowie Version 3 zu deaktivieren. Auf Systemen mit hoher Parallelität kann die Deadlock-Erkennung zu einer Verlangsamung führen, wenn zahlreiche Threads auf dieselbe Sperre warten. Weitere Informationen zu diesem Parameter finden Sie in der MySQL-Dokumentation.
`innodb_default_row_format`	Ja	Dieser Parameter definiert das Standardzeilenformat für InnoDB-Tabellen (einschließlich benutzerdefinierter temporärer InnoDB-Tabellen). Er gilt für Aurora MySQL Version 2 und 3. Mögliche Werte sind `DYNAMIC`, `COMPACT` oder `REDUNDANT`. .
`innodb_file_per_table`	Ja	Dieser Parameter wirkt sich auf die Organisation des Tabellenspeichers aus. Weitere Informationen finden Sie unter Speicherskalierung.

Parametername	Anpassbar	Hinweise
innodb_flush_log_at_trx_commit	Ja	Es wird dringend empfohlen, den Standardwert 1 zu verwenden. Bevor Sie diesen Parameter in Aurora-MySQL-Version 3 in einen anderen Wert als 1 ändern können, müssen Sie den Wert von innodb_trx_commit_allow_data_loss in 1 ändern. Weitere Informationen finden Sie unter Konfigurieren, wie oft der Protokollpuffer geleert wird.
innodb_ft_max_token_size	Ja	Keine
innodb_ft_min_token_size	Ja	Keine
innodb_ft_num_word_optimize	Ja	Keine
innodb_ft_sort_pll_degree	Ja	Keine
innodb_online_alter_log_max_size	Ja	Keine
innodb_optimize_fulltext_only	Ja	Keine
innodb_page_size	Nein	Keine
innodb_print_all_deadlocks	Ja	Wenn dieser Parameter aktiviert ist, werden Informationen über alle InnoDB-Deadlocks im Fehlerprotokoll von Aurora MySQL aufgezeichnet. Weitere Informationen finden Sie unter Minimieren und Beheben von Aurora-MySQL-Deadlocks.

Parametername	Anpassbar	Hinweise
innodb_purge_batch_size	Ja	Keine
innodb_purge_threads	Ja	Keine
innodb_rollback_on_timeout	Ja	Keine
innodb_rollback_segments	Ja	Keine
innodb_spin_wait_delay	Ja	Keine
innodb_strict_mode	Ja	Keine
innodb_support_xa	Ja	Aus Aurora-MySQL-Version 3 entfernt.
innodb_sync_array_size	Ja	Keine
innodb_sync_spin_loops	Ja	Keine
innodb_stats_include_delete_marked	Ja	Wenn dieser Parameter aktiviert ist, bezieht InnoDB zum Löschen markierte Datensätze in die Berechnung der persistenten Optimierungsstatistiken ein. Dieser Parameter gilt für Aurora-My SQL-Version 2.12 und höher sowie Version 3.
innodb_table_locks	Ja	Keine

Parametername	Anpassbar	Hinweise
innodb_trx_commit_allow_data_loss	Ja	In Aurora-MySQL-Version 3 legen Sie den Wert dieses Parameters auf 1 fest, sodass Sie den Wert von innodb_flush_log_at_trx_commit ändern können. Der Standardwert von innodb_trx_commit_allow_data_loss ist 0. Weitere Informationen finden Sie unter Konfigurieren, wie oft der Protokollpuffer geleert wird.
innodb_undo_directory	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
internal_tmp_disk_storage_engine	Ja	Steuert, welche In-Memory-Speicher-Engine für interne temporäre Tabellen verwendet wird. Zulässige Werte sind INNODB und MYISAM. Dieser Parameter gilt für Aurora MySQL Version 2.
internal_tmp_mem_storage_engine	Ja	Steuert, welche In-Memory-Speicher-Engine für interne temporäre Tabellen verwendet wird. Zulässige Werte sind MEMORY und TempTable . Dieser Parameter gilt für Aurora MySQL Version 3.
key_buffer_size	Ja	Schlüssel-Cache für MyISAM-Tabellen. Weitere Informationen finden Sie unter keycache->cache_lock mutex.

Parametername	Anpassbar	Hinweise
`lc_time_names`	Ja	Keine
`log_error_suppression_list`	Ja	Gibt eine Liste von Fehlercodes an, die nicht im MySQL-Fehlerprotokoll protokolliert werden. Auf diese Weise können Sie bestimmte unkritische Fehlerbedingungen ignorieren, um Ihre Fehlerprotokolle sauber zu halten. Weitere Informationen finden Sie unter log_error_suppression_list in der MySQL-Dokumentation. Dieser Parameter gilt für Aurora-My SQL-Version 3.03 und höher.
`low_priority_updates`	Ja	`INSERT`-, `UPDATE`-, `DELETE`- und `LOCK TABLE WRITE`-Operationen warten, bis kein ausstehender `SELECT`-Vorgang mehr vorhanden ist. Dieser Parameter wirkt sich nur auf Speicher-Engines aus, die ausschließlich Sperren auf Tabellenebene verwenden (MyISAM, MEMORY, MERGE). Dieser Parameter gilt für Aurora MySQL Version 3.

Parametername	Anpassbar	Hinweise
`lower_case_table_names`	Ja (Aurora-MySQL-Version 2) Nur zur Clusterer stellung (Aurora-MySQL-Version 3)	Stellen Sie in Aurora-MySQL-Version 2.10 und höheren 2.x-Versionen sicher, dass Sie alle Reader-Instances neu starten, nachdem Sie diese Einstellung geändert und die Writer-Instance neu gestartet haben. Details hierzu finden Sie unter Neustarten eines Aurora-Clusters mit Leseverfügbarkeit. In Aurora-MySQL-Version 3 wird der Wert dieses Parameters zum Zeitpunkt der Erstellung des Clusters dauerhaft festgelegt. Wenn Sie für diese Option einen nicht standardmäßigen Wert verwenden, richten Sie Ihre benutzerdefinierte Parametergruppe Aurora-MySQL-Version 3 vor dem Upgrade ein, und geben Sie die Parametergruppe während des Snapshot-Wiederherstellungsvorgangs an, der den Cluster der Version 3 erstellt. Mit einer globalen Aurora-Datenbank, die auf Aurora MySQL basiert, können Sie kein direktes Upgrade von Aurora MySQL Version 2 auf Version 3 durchführen, wenn der `lower_case_table_names` -Parameter aktiviert ist. Weitere Informationen zu den möglichen Verfahren finden Sie unter Hauptversions-Upgrades.
`master-info-repository`	Ja	Aus Aurora-MySQL-Version 3 entfernt.

Parametername	Anpassbar	Hinweise
`master_verify_checksum`	Ja	Aurora-MySQL-Version 2. Verwenden Sie `source_verify_checksum` in Aurora-MySQL-Version 3.
`max_delayed_threads`	Ja	Legt die maximale Anzahl von Threads für die Verarbeitung von `INSERT DELAYED`-Anweisungen fest. Dieser Parameter gilt für Aurora MySQL Version 3.
`max_error_count`	Ja	Die maximale Anzahl von Fehler-, Warn- und Hinweismeldungen, die für die Anzeige gespeichert werden sollen. Dieser Parameter gilt für Aurora MySQL Version 3.
`max_execution_time`	Ja	Das Timeout für die Ausführung von `SELECT`-Anweisungen in Milliseku nden. Der Wert kann zwischen `0` und `18446744073709551615` liegen. Wenn auf `0` eingestellt, gibt es kein Timeout. Weitere Informationen finden Sie unter max_execution_time in der MySQL-Dokumentation.
`min_examined_row_limit`	Ja	Verwenden Sie diesen Parameter, um zu verhindern, dass Abfragen, die weniger als die angegebene Anzahl von Zeilen untersuchen, protokolliert werden.
`partial_revokes`	Nein	Dieser Parameter gilt für Aurora MySQL Version 3.

Parametername	Anpassbar	Hinweise
`preload_buffer_size`	Ja	Die Größe des Puffers, der beim Vorabladen von Indizes zugewiesen wird. Dieser Parameter gilt für Aurora MySQL Version 3.
`query_cache_type`	Ja	Aus Aurora-MySQL-Version 3 entfernt.

Parametername	Anpassbar	Hinweise
`read_only`	Ja	Wenn dieser Parameter aktiviert ist, erlaubt der Server nur Aktualisierungen, die von Replikat-Threads durchgeführt werden. Für Aurora-MySQL-Version 2 sind die folgenden Werte gültig: • `0` – `OFF` • `1` – `ON` • `{TrueIfReplica}` – `ON` für Lesereplikate. Dies ist der Standardwert. • `{TrueIfClusterReplica}` – `ON` für Replikat-Cluster wie regionsübergreifende Lesereplikate, sekundäre Cluster in einer globalen Aurora-Datenbank und Blau/Grün-Bereitstellungen Für Aurora-MySQL-Version 3 sind die folgenden Werte gültig: • `0` – `OFF`. Dies ist der Standardwert. • `1` – `ON` • `{TrueIfClusterReplica}` – `ON` für Replikat-Cluster wie regionsübergreifende Lesereplikate, sekundäre Cluster in einer globalen Aurora-Datenbank und Blau/Grün-Bereitstellungen Bei Aurora MySQL Version 3 gilt dieser Parameter nicht für Benutzer,

Parametername	Anpassbar	Hinweise
		die über die `CONNECTION_ADMIN` -Berechtigung verfügen. Dies schließt den Aurora-Hauptbenutzer ein. Weitere Informationen finden Sie unter Rollenbasiertes Berechtigungsmodell.
`relay-log-space-limit`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`replica_parallel_type`	Ja	Dieser Parameter ermöglicht die parallele Ausführung aller Threads mit ausstehendem Commit, die sich bereits in der Vorbereitungsphase befinden, auf dem Replikat, ohne die Konsistenz zu verletzen. Er gilt für Aurora MySQL Version 3. In Aurora MySQL Version 3.03.* und niedriger ist der Standardwert DATABASE. In Aurora MySQL Version 3.04.* und höher ist der Standardwert LOGICAL_CLOCK.
`replica_preserve_commit_order`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`replica_transaction_retries`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.

Parametername	Anpassbar	Hinweise
`replica_type_conversions`	Ja	Dieser Parameter bestimmt die Typkonvertierungen, die für Replikate verwendet werden. Die zulässigen Werte sind `ALL_LOSSY` , `ALL_NON_LOSSY` , `ALL_SIGNED` und `ALL_UNSIGNED` . Weitere Informationen finden Sie in der MySQL-Dokumentation unter Replication with Differing Table Definitions on Source and Replica. Dieser Parameter gilt für Aurora MySQL Version 3.
`replicate-do-db`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`replicate-do-table`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`replicate-ignore-db`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`replicate-ignore-table`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`replicate-wild-do-table`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`replicate-wild-ignore-table`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`require_secure_transport`	Ja	Dieser Parameter gilt für Aurora MySQL Version 2 und 3. Weitere Informationen finden Sie unter TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

Parametername	Anpassbar	Hinweise
`rpl_read_size`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`server_audit_cw_upload`	Nein	Dieser Parameter wurde in Aurora MySQL als veraltet gekennzeichnet. Verwenden Sie `server_audit_logs_upload` . Weitere Informationen finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.
`server_audit_events`	Ja	Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.
`server_audit_excl_users`	Ja	Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.
`server_audit_incl_users`	Ja	Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.
`server_audit_logging`	Ja	Anweisungen zum Hochladen von Protokollen in Amazon CloudWatch Logs finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.

Parametername	Anpassbar	Hinweise
server_audit_logs_upload	Ja	Sie können Prüfprotokolle in CloudWatch Logs veröffentlichen, indem Sie das erweiterte Auditing aktivieren und diesen Parameter auf 1 setzen. Der Standardwert für den Parameter server_audit_logs_upload ist 0. Weitere Informationen finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.
server_id	Nein	Keine
skip-character-set-client-handshake	Ja	Keine
skip_name_resolve	Nein	Keine
slave-skip-errors	Ja	Gilt nur für Cluster der Aurora MySQL Version 2 mit MySQL-5.7-Kompatibilität.
source_verify_checksum	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
sync_frm	Ja	Aus Aurora-MySQL-Version 3 entfernt.
thread_cache_size	Ja	Die Anzahl der Threads, die zwischeng espeichert werden sollen. Dieser Parameter gilt für Aurora MySQL Version 2 und 3.

Parametername	Anpassbar	Hinweise
`time_zone`	Ja	Standardmäßig ist die Zeitzone für einen Aurora-DB-Cluster auf die Universal Time Coordinated (UTC) eingestellt. Sie können die Zeitzone für Instances in Ihrem DB-Cluster stattdessen auf die lokale Zeitzone Ihrer Anwendung festlegen. Weitere Informationen finden Sie unter Lokale Zeitzone für Amazon Aurora-DB-Cluster.
`tls_version`	Ja	Weitere Informationen finden Sie unter TLS-Versionen für Aurora MySQL.

Parameter auf Instance-Ebene

In der folgenden Tabelle werden alle Parameter aufgeführt, die für eine bestimmte DB-Instance in einem Aurora MySQL-DB-Cluster gelten.

Parametername	Anpassbar	Hinweise
`activate_all_roles_on_login`	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
`allow-suspicious-udfs`	Nein	Keine
`aurora_disable_hash_join`	Ja	Setzen Sie diesen Parameter auf `ON`, um die Hash-Join-Optimierung in Aurora MySQL Version 2.09 oder höher zu deaktivieren. Für Version 3 wird dies nicht unterstützt. Weitere Informationen finden Sie unter Parallel Query für Amazon Aurora MySQL.

Parametername	Anpassbar	Hinweise
`aurora_lab_mode`	Ja	Weitere Informationen finden Sie unter Amazon Aurora MySQL-Labor-Modus. Aus Aurora-MySQL-Version 3 entfernt.
`aurora_oom_response`	Ja	Dieser Parameter wird für die Aurora-MySQL-Versionen 2 und 3 unterstützt. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.
`aurora_parallel_query`	Ja	Legen Sie dies auf `ON` fest, um parallele Abfragen in den Aurora-MySQL-Versionen 2.09 oder höher zu unterstützen. Der alte Parameter `aurora_pq` wird in diesen Versionen nicht verwendet. Weitere Informationen finden Sie unter Parallel Query für Amazon Aurora MySQL.
`aurora_pq`	Ja	Setzen Sie dies auf `OFF`, um parallele Abfragen für bestimmte DB-Instances in Aurora-MySQL-Versionen vor 2.09 zu deaktivieren. In den Versionen 2.09 oder höher aktivieren und deaktivieren Sie parallele Abfragen stattdessen mit `aurora_parallel_query`. Weitere Informationen finden Sie unter Parallel Query für Amazon Aurora MySQL.

Parametername	Anpassbar	Hinweise
`aurora_read_replica_read_committed`	Ja	Aktiviert die Isolationsstufe `READ COMMITTED` für Aurora Replicas und ändert das Isolationsverhalten zur Reduzierung der Bereinigungsverzögerung bei lang andauernden Abfragen. Aktivieren Sie diese Einstellung nur, wenn Sie die Verhaltensänderungen und deren Auswirkungen auf Ihre Abfrageergebnisse verstehen. Beispielsweise gehört dazu, dass diese Einstellung eine weniger strikte Isolation als der MySQL-Standard verwendet. Bei Aktivierung kann es sein, dass lang andauernde Abfragen mehr als eine Kopie derselben Zeile sehen, da Aurora die Tabellendaten umorganisiert, während die Abfrage läuft. Weitere Informationen finden Sie unter Aurora MySQL-Isolierungsstufen.
`aurora_tmptable_enable_per_table_limit`	Ja	Bestimmt, ob der der Parameter `tmp_table_size` auch die maximale Größe temporärer Tabellen im Arbeitsspeicher steuert, die von der `TempTable` -Speicher-Engine in Aurora-MySQL-Version 3.04 und höher erstellt werden. Weitere Informationen finden Sie unter Begrenzung der Größe interner temporärer Tabellen im Arbeitsspeicher.

Parametername	Anpassbar	Hinweise
aurora_use_vector_instructions	Ja	Wenn dieser Parameter aktiviert ist, verwendet Aurora MySQL optimierte Anweisungen zur Vektorverarbeitung, die von modernen CPUs bereitgestellt werden, um die Leistung bei I/O-intensiven Workloads zu verbessern. Diese Einstellung ist in Aurora MySQL Version 3.05 und höher standardmäßig aktiviert.
autocommit	Ja	Keine
automatic_sp_privileges	Ja	Keine
back_log	Ja	Keine
basedir	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
binlog_cache_size	Ja	Keine
binlog_max_flush_queue_time	Ja	Keine
binlog_order_commits	Ja	Keine
binlog_stmt_cache_size	Ja	Keine
binlog_transaction_compression	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
binlog_transaction_compression_level_zstd	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
bulk_insert_buffer_size	Ja	Keine

Parametername	Anpassbar	Hinweise
concurrent_insert	Ja	Keine
connect_timeout	Ja	Keine
core-file	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
datadir	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
default_authentication_plugin	Nein	Dieser Parameter gilt für Aurora MySQL Version 3.
default_time_zone	Nein	Keine
default_tmp_storage_engine	Ja	Die Standard-Speicher-Engine für temporäre Tabellen.
default_week_format	Ja	Keine
delay_key_write	Ja	Keine
delayed_insert_limit	Ja	Keine
delayed_insert_timeout	Ja	Keine
delayed_queue_size	Ja	Keine
div_precision_increment	Ja	Keine
end_markers_in_json	Ja	Keine
eq_range_index_dive_limit	Ja	Keine

Parametername	Anpassbar	Hinweise
event_scheduler	Manchmal	Zeigt den Status des Ereignisplaners an. Modifizierbar nur auf Cluster-Ebene in Aurora-MySQL-Version 3.
explicit_defaults_for_timestamp	Ja	Keine
flush	Nein	Keine
flush_time	Ja	Keine
ft_boolean_syntax	Nein	Keine
ft_max_word_len	Ja	Keine
ft_min_word_len	Ja	Keine
ft_query_expansion_limit	Ja	Keine
ft_stopword_file	Ja	Keine
general_log	Ja	Anweisungen zum Hochladen von Protokollen in CloudWatch Logs finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.
general_log_file	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
group_concat_max_len	Ja	Keine
host_cache_size	Ja	Keine

Parametername	Anpassbar	Hinweise
`init_connect`	Ja	Der Befehl, der vom Server für jeden Client ausgeführt wird, der eine Verbindung herstellt. Verwenden Sie doppelte Anführungszeichen („) für Einstellungen, um Verbindungsfehler zu vermeiden, zum Beispiel: `SET optimizer_switch="hash_join =off"` Bei Aurora MySQL Version 3 gilt dieser Parameter nicht für Benutzer, die über die `CONNECTION_ADMIN` -Berechti gung verfügen. Weitere Informati onen finden Sie unter Rollenbasiertes Berechtigungsmodell.
`innodb_adaptive_hash_index`	Ja	Sie können diesen Parameter auf DB-Instance-Ebene in Aurora MySLQ Version 2 ändern. In Aurora MySQL Version 3 kann er nur auf DB-Cluster-Ebene geändert werden. Der Adaptive-Hash-Index wird auf Reader-DB-Instances nicht unterstützt.
`innodb_adaptive_max_sleep_d elay`	Ja	Das Ändern dieses Parameters hat keine Auswirkung, da `innodb_th read_concurrency` für Aurora immer 0 ist.

Parametername	Anpassbar	Hinweise
innodb_aurora_max_partitions_for_range	Ja	In einigen Fällen, in denen persistente Statistiken nicht verfügbar sind, können Sie diesen Parameter verwenden, um die Leistung von Schätzungen der Zeilenanzahl in partitionierten Tabellen zu verbessern. Sie können ihn auf einen Wert zwischen 0 und 8192 festlegen, wobei der Wert die Anzahl der Partitionen bestimmt, die bei der Schätzung der Zeilenanzahl überprüft werden sollen. Der Standardwert ist 0. Dieser entspricht der Schätzung, dass alle Partitionen verwendet werden, was mit dem Standardverhalten von MySQL übereinstimmt. Dieser Parameter ist für Aurora MySQL Version 3.03.1 verfügbar.
innodb_autoextend_increment	Ja	Keine
innodb_buffer_pool_dump_at_shutdown	Nein	Keine
innodb_buffer_pool_dump_now	Nein	Keine
innodb_buffer_pool_filename	Nein	Keine
innodb_buffer_pool_load_abort	Nein	Keine

Parametername	Anpassbar	Hinweise
innodb_buffer_pool_load_at_startup	Nein	Keine
innodb_buffer_pool_load_now	Nein	Keine
innodb_buffer_pool_size	Ja	Der Standardwert wird durch eine Formel dargestellt. Mehr über die Berechnung des DBInstanceClassMemory -Werts in der Formel erfahren Sie unter DB-Parameter-Formel-Variablen.
innodb_change_buffer_max_size	Nein	Aurora MySQL verwendet den InnoDB-Änderungspuffer überhaupt nicht.
innodb_compression_failure_threshold_pct	Ja	Keine
innodb_compression_level	Ja	Keine
innodb_compression_pad_pct_max	Ja	Keine
innodb_concurrency_tickets	Ja	Das Ändern dieses Parameters hat keine Auswirkung, da innodb_thread_concurrency für Aurora immer 0 ist.

Parametername	Anpassbar	Hinweise
innodb_deadlock_detect	Ja	Diese Option wird verwendet, um die Deadlock-Erkennung in Aurora-MySQL-Version 2.11 und höher sowie Version 3 zu deaktivieren. Auf Systemen mit hoher Parallelität kann die Deadlock-Erkennung zu einer Verlangsamung führen, wenn zahlreiche Threads auf dieselbe Sperre warten. Weitere Informationen zu diesem Parameter finden Sie in der MySQL-Dokumentation.
innodb_file_format	Ja	Aus Aurora-MySQL-Version 3 entfernt.
innodb_flushing_avg_loops	Nein	Keine
innodb_force_load_corrupted	Nein	Keine
innodb_ft_aux_table	Ja	Keine
innodb_ft_cache_size	Ja	Keine
innodb_ft_enable_stopword	Ja	Keine
innodb_ft_server_stopword_table	Ja	Keine
innodb_ft_user_stopword_table	Ja	Keine
innodb_large_prefix	Ja	Aus Aurora-MySQL-Version 3 entfernt.
innodb_lock_wait_timeout	Ja	Keine
innodb_log_compressed_pages	Nein	Keine

Parametername	Anpassbar	Hinweise
innodb_lru_scan_depth	Ja	Keine
innodb_max_purge_lag	Ja	Keine
innodb_max_purge_lag_delay	Ja	Keine
innodb_monitor_disable	Ja	Keine
innodb_monitor_enable	Ja	Keine
innodb_monitor_reset	Ja	Keine
innodb_monitor_reset_all	Ja	Keine
innodb_old_blocks_pct	Ja	Keine
innodb_old_blocks_time	Ja	Keine
innodb_open_files	Ja	Keine
innodb_print_all_deadlocks	Ja	Wenn dieser Parameter aktiviert ist, werden Informationen über alle InnoDB-Deadlocks im Fehlerprotokoll von Aurora MySQL aufgezeichnet. Weitere Informationen finden Sie unter Minimieren und Beheben von Aurora-MySQL-Deadlocks.
innodb_random_read_ahead	Ja	Keine
innodb_read_ahead_threshold	Ja	Keine
innodb_read_io_threads	Nein	Keine

Parametername	Anpassbar	Hinweise
innodb_read_only	Nein	Aurora MySQL verwaltet den Schreibschutz- und Lesen/Schreiben-Status von DB-Instances basierend auf dem Typ des Clusters. Ein bereitgestellter Cluster hat beispielsweise eine Lesen/Schreiben-DB-Instance (die primäre Instance). Alle anderen Instances im Cluster sind schreibgeschützt (die Aurora-Replicas).
innodb_replication_delay	Ja	Keine
innodb_sort_buffer_size	Ja	Keine
innodb_stats_auto_recalc	Ja	Keine
innodb_stats_method	Ja	Keine
innodb_stats_on_metadata	Ja	Keine
innodb_stats_persistent	Ja	Keine
innodb_stats_persistent_sample_pages	Ja	Keine
innodb_stats_transient_sample_pages	Ja	Keine
innodb_thread_concurrency	Nein	Keine
innodb_thread_sleep_delay	Ja	Das Ändern dieses Parameters hat keine Auswirkung, da innodb_thread_concurrency für Aurora immer 0 ist.

Parametername	Anpassbar	Hinweise
interactive_timeout	Ja	Aurora wertet den Mindestwert von interactive_timeout und wait_timeout aus. Dieses Minimum wird dann als Timeout verwendet, um alle inaktiven Sitzungen, sowohl interaktive als auch nicht interaktive, zu beenden.
internal_tmp_disk_storage_engine	Ja	Steuert, welche In-Memory-Speicher-Engine für interne temporäre Tabellen verwendet wird. Zulässige Werte sind INNODB und MYISAM. Dieser Parameter gilt für Aurora MySQL Version 2.
internal_tmp_mem_storage_engine	Manchmal	Steuert, welche In-Memory-Speicher-Engine für interne temporäre Tabellen verwendet wird. Zulässige Werte für Schreiber-DB-Instances sind MEMORY undTempTable . Für Leser-DB-Instances ist dieser Parameter auf TempTable gesetzt und kann nicht geändert werden. Dieser Parameter gilt für Aurora MySQL Version 3.
join_buffer_size	Ja	Keine
keep_files_on_create	Ja	Keine
key_buffer_size	Ja	Schlüssel-Cache für MyISAM-Tabellen. Weitere Informationen finden Sie unter keycache->cache_lock mutex.

Parametername	Anpassbar	Hinweise
key_cache_age_threshold	Ja	Keine
key_cache_block_size	Ja	Keine
key_cache_division_limit	Ja	Keine
local_infile	Ja	Keine
lock_wait_timeout	Ja	Keine
log-bin	Nein	Wenn die Einstellung binlog_format auf STATEMENT , MIXED oder ROW gesetzt wird, wird log-bin automatisch auf ON festgelegt. Wenn die Einstellung binlog_format auf OFF gesetzt wird, wird log-bin automatisch auf OFF gesetzt. Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).
log_bin_trust_function_creators	Ja	Keine
log_bin_use_v1_row_events	Ja	Aus Aurora-MySQL-Version 3 entfernt.
log_error	Nein	Keine

Parametername	Anpassbar	Hinweise
log_error_suppression_list	Ja	Gibt eine Liste von Fehlercodes an, die nicht im MySQL-Fehlerprotokoll protokolliert werden. Auf diese Weise können Sie bestimmte unkritische Fehlerbedingungen ignorieren, um Ihre Fehlerprotokolle sauber zu halten. Weitere Informationen finden Sie unter log_error_suppression_list in der MySQL-Dokumentation. Dieser Parameter gilt für Aurora-MySQL-Version 3.03 und höher.
log_output	Ja	Keine
log_queries_not_using_indexes	Ja	Keine
log_slave_updates	Nein	Aurora-MySQL-Version 2. Verwenden Sie log_replica_updates in Aurora-MySQL-Version 3.
log_replica_updates	Nein	Aurora-MySQL-Version 3
log_throttle_queries_not_using_indexes	Ja	Keine
log_warnings	Ja	Aus Aurora-MySQL-Version 3 entfernt.
long_query_time	Ja	Keine

Parametername	Anpassbar	Hinweise
low_priority_updates	Ja	INSERT-, UPDATE-, DELETE- und LOCK TABLE WRITE-Operationen warten, bis kein ausstehender SELECT-Vorgang mehr vorhanden ist. Dieser Parameter wirkt sich nur auf Speicher-Engines aus, die ausschließlich Sperren auf Tabellenebene verwenden (MyISAM, MEMORY, MERGE). Dieser Parameter gilt für Aurora MySQL Version 3.
max_allowed_packet	Ja	Keine
max_binlog_cache_size	Ja	Keine
max_binlog_size	Nein	Keine
max_binlog_stmt_cache_size	Ja	Keine
max_connect_errors	Ja	Keine
max_connections	Ja	Der Standardwert wird durch eine Formel dargestellt. Mehr über die Berechnung des DBInstanceClassMemory -Werts in der Formel erfahren Sie unter DB-Parameter-Formel-Variablen. Informationen zu den Standardwerten in Abhängigkeit von der Instance-Klasse finden Sie unter Maximale Verbindungen zu einer Aurora MySQL-DB-Instance.

Parametername	Anpassbar	Hinweise
`max_delayed_threads`	Ja	Legt die maximale Anzahl von Threads für die Verarbeitung von `INSERT DELAYED`-Anweisungen fest. Dieser Parameter gilt für Aurora MySQL Version 3.
`max_error_count`	Ja	Die maximale Anzahl von Fehler-, Warn- und Hinweismeldungen, die für die Anzeige gespeichert werden sollen. Dieser Parameter gilt für Aurora MySQL Version 3.
`max_execution_time`	Ja	Das Timeout für die Ausführung von `SELECT`-Anweisungen in Milliseku nden. Der Wert kann zwischen `0` und `18446744073709551615` liegen. Wenn auf `0` eingestellt, gibt es kein Timeout. Weitere Informationen finden Sie unter max_execution_time in der MySQL-Dokumentation.
`max_heap_table_size`	Ja	Keine
`max_insert_delayed_threads`	Ja	Keine
`max_join_size`	Ja	Keine
`max_length_for_sort_data`	Ja	Aus Aurora-MySQL-Version 3 entfernt.
`max_prepared_stmt_count`	Ja	Keine
`max_seeks_for_key`	Ja	Keine
`max_sort_length`	Ja	Keine

Parametername	Anpassbar	Hinweise
max_sp_recursion_depth	Ja	Keine
max_tmp_tables	Ja	Aus Aurora-MySQL-Version 3 entfernt.
max_user_connections	Ja	Keine
max_write_lock_count	Ja	Keine
metadata_locks_cache_size	Ja	Aus Aurora-MySQL-Version 3 entfernt.
min_examined_row_limit	Ja	Verwenden Sie diesen Parameter, um zu verhindern, dass Abfragen, die weniger als die angegebene Anzahl von Zeilen untersuchen, protokolliert werden. Dieser Parameter gilt für Aurora MySQL Version 3.
myisam_data_pointer_size	Ja	Keine
myisam_max_sort_file_size	Ja	Keine
myisam_mmap_size	Ja	Keine
myisam_sort_buffer_size	Ja	Keine
myisam_stats_method	Ja	Keine
myisam_use_mmap	Ja	Keine
net_buffer_length	Ja	Keine
net_read_timeout	Ja	Keine
net_retry_count	Ja	Keine
net_write_timeout	Ja	Keine

Parametername	Anpassbar	Hinweise
old-style-user-limits	Ja	Keine
old_passwords	Ja	Aus Aurora-MySQL-Version 3 entfernt.
optimizer_prune_level	Ja	Keine
optimizer_search_depth	Ja	Keine
optimizer_switch	Ja	Weitere Informationen zu Aurora MySQL-Features, die diesen Schalter verwenden, finden Sie unter Bewährte Methoden mit Amazon Aurora MySQL.
optimizer_trace	Ja	Keine
optimizer_trace_features	Ja	Keine
optimizer_trace_limit	Ja	Keine
optimizer_trace_max_mem_size	Ja	Keine
optimizer_trace_offset	Ja	Keine
performance-schema-consumer-events-waits-current	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance-schema-consumer-events-waits-current . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
`performance-schema-instrument`	Ja	Wenn die Spalte Quelle für den Parameter `performance_schema` auf `Modified` gesetzt ist, verwendet das Leistungsschema den Parameter `performance-schema-instrument` . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
`performance_schema`	Ja	Wenn die Spalte Quelle auf `Modified` gesetzt ist, verwaltet Performance Insights das Leistungsschema. Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
`performance_schema_accounts_size`	Ja	Wenn die Spalte Quelle für den Parameter `performance_schema` auf `Modified` gesetzt ist, verwendet das Leistungsschema den Parameter `performance_schema_accounts_size` . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_consumer_global_instrumentation	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_global_instrumentation . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_consumer_thread_instrumentation	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_thread_instrumentation . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_consumer_events_stages_current	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_stages_current . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_consumer_events_stages_history	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_stages_history . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_consumer_events_stages_history_long	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_stages_history_long . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_consumer_events_statements_current	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_statements_current . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_consumer_events_statements_history	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_statements_history . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_consumer_events_statements_history_long	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_statements_history_long . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_consumer_events_waits_history	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_waits_history . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_consumer_events_waits_history_long	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_events_waits_history_long . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_consumer_statements_digest	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_consumer_statements_digest . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_digests_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_digests_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_events_stages_history_long_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_stages_history_long_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_events_stages_history_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_stages_history_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_events_statements_history_long_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_statements_history_long_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_events_statements_history_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_statements_history_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_events_transactions_history_long_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_transactions_history_long_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_events_transactions_history_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_transactions_history_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_events_waits_history_long_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_waits_history_long_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_events_waits_history_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_events_waits_history_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_hosts_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_hosts_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_cond_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_cond_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_cond_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_cond_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_digest_length	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_digest_length . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
`performance_schema_max_file_classes`	Ja	Wenn die Spalte Quelle für den Parameter `performance_schema` auf `Modified` gesetzt ist, verwendet das Leistungsschema den Parameter `performance_schema_max_file_classes` . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
`performance_schema_max_file_handles`	Ja	Wenn die Spalte Quelle für den Parameter `performance_schema` auf `Modified` gesetzt ist, verwendet das Leistungsschema den Parameter `performance_schema_max_file_handles` . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
`performance_schema_max_file_instances`	Ja	Wenn die Spalte Quelle für den Parameter `performance_schema` auf `Modified` gesetzt ist, verwendet das Leistungsschema den Parameter `performance_schema_max_file_instances` . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_index_stat	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_index_stat . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_memory_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_memory_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_metadata_locks	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_metadata_locks . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_mutex_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_mutex_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_mutex_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_mutex_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_prepared_statements_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_prepared_statements_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_program_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_program_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_rwlock_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_rwlock_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_rwlock_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_rwlock_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_socket_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_socket_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_socket_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_socket_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_sql_text_length	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_sql_text_length . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_stage_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_stage_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_statement_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_statement_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_statement_stack	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_statement_stack . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_table_handles	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_table_handles . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_table_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_table_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_table_lock_stat	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_table_lock_stat . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_max_thread_classes	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_thread_classes . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_max_thread_instances	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_max_thread_instances . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_session_connect_attrs_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_session_connect_attrs_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_setup_actors_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_setup_actors_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
performance_schema_setup_objects_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_setup_objects_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_show_processlist	Ja	Dieser Parameter bestimmt, welche SHOW PROCESSLIST -Implementierung verwendet werden soll: • Die Standardimplementierung iteriert innerhalb des Thread-Managers über aktive Threads hinweg und hält dabei einen globalen Mutex bereit. Dies kann zu einer langsamen Leistung führen, insbesondere bei stark ausgelasteten Systemen. • Die alternative SHOW PROCESSLIST -Implementierung basiert auf der Tabelle processlist des Leistungsschemas. Diese Implementierung fragt aktive Threaddaten aus dem Leistungsschema und nicht aus dem Thread-Manager ab und benötigt keinen Mutex. Dieser Parameter gilt für Aurora-MySQL-Version 2.12 und höher sowie Version 3. Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_show_processlist . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.

Parametername	Anpassbar	Hinweise
performance_schema_users_size	Ja	Wenn die Spalte Quelle für den Parameter performance_schema auf Modified gesetzt ist, verwendet das Leistungsschema den Parameter performance_schema_users_size . Weitere Informationen zum Aktivieren des Leistungsschemas finden Sie unter Feststellen, ob Performance Insights das Leistungsschema verwaltet.
pid_file	Nein	Keine
plugin_dir	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
port	Nein	Aurora MySQL verwaltet die Verbindungseigenschaften und erzwingt konsistente Einstellungen für alle DB-Instances in einem Cluster.
preload_buffer_size	Ja	Die Größe des Puffers, der beim Vorabladen von Indizes zugewiesen wird. Dieser Parameter gilt für Aurora MySQL Version 3.
profiling_history_size	Ja	Keine
query_alloc_block_size	Ja	Keine
query_cache_limit	Ja	Aus Aurora-MySQL-Version 3 entfernt.
query_cache_min_res_unit	Ja	Aus Aurora-MySQL-Version 3 entfernt.

Parametername	Anpassbar	Hinweise
query_cache_size	Ja	Der Standardwert wird durch eine Formel dargestellt. Mehr über die Berechnung des DBInstanceClassMemory -Werts in der Formel erfahren Sie unter DB-Parameter-Formel-Variablen. Aus Aurora-MySQL-Version 3 entfernt.
query_cache_type	Ja	Aus Aurora-MySQL-Version 3 entfernt.
query_cache_wlock_invalidate	Ja	Aus Aurora-MySQL-Version 3 entfernt.
query_prealloc_size	Ja	Keine
range_alloc_block_size	Ja	Keine
read_buffer_size	Ja	Keine

Parametername	Anpassbar	Hinweise
`read_only`	Ja	Wenn dieser Parameter aktiviert ist, erlaubt der Server nur Aktualisierungen, die von Replikat-Threads durchgeführt werden. Für Aurora-MySQL-Version 2 sind die folgenden Werte gültig: • `0` – `OFF` • `1` – `ON` • `{TrueIfReplica}` – `ON` für Lesereplikate. Dies ist der Standardwert. • `{TrueIfClusterReplica}` – `ON` für Instances in Replikat-Clustern wie regionsübergreifende Lesereplikate, sekundäre Cluster in einer globalen Aurora-Datenbank und Blau/Grün-Bereitstellungen Wir empfehlen, die DB-Cluster-Parametergruppe in Aurora MySQL Version 2 zu verwenden, um sicherzustellen, dass der `read_only` -Parameter beim Failover auf neue Writer-Instances angewendet wird. **Note** Reader-Instances sind immer schreibgeschützt, da Aurora MySQL `innodb_read_only` für alle Reader auf 1 festlegt.

Parametername	Anpassbar	Hinweise
		Daher ist `read_only` auf Reader-Instances redundant. Auf Instance-Ebene aus Aurora MySQL Version 3 entfernt.
`read_rnd_buffer_size`	Ja	Keine
`relay-log`	Nein	Keine
`relay_log_info_repository`	Ja	Aus Aurora-MySQL-Version 3 entfernt.
`relay_log_recovery`	Nein	Keine
`replica_checkpoint_group`	Ja	Aurora-MySQL-Version 3
`replica_checkpoint_period`	Ja	Aurora-MySQL-Version 3
`replica_parallel_workers`	Ja	Aurora-MySQL-Version 3
`replica_pending_jobs_size_max`	Ja	Aurora-MySQL-Version 3
`replica_skip_errors`	Ja	Aurora-MySQL-Version 3
`replica_sql_verify_checksum`	Ja	Aurora-MySQL-Version 3
`safe-user-create`	Ja	Keine
`secure_auth`	Ja	Dieser Parameter ist in Aurora-MySQL-Version 2 immer aktiviert. Der Versuch, ihn zu deaktivieren, generiert einen Fehler. Aus Aurora-MySQL-Version 3 entfernt.

Parametername	Anpassbar	Hinweise
secure_file_priv	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
show_create_table_verbosity	Ja	Wenn diese Variable aktiviert wird, zeigt SHOW_CREATE_TABLE das ROW_FORMAT unabhängig davon an, ob es das Standardformat ist. Dieser Parameter gilt für Aurora-My SQL-Version 2.12 und höher sowie Version 3.
skip-slave-start	Nein	Keine
skip_external_locking	Nein	Keine
skip_show_database	Ja	Keine
slave_checkpoint_group	Ja	Aurora-MySQL-Version 2. Verwenden Sie replica_checkpoint_group in Aurora-MySQL-Version 3.
slave_checkpoint_period	Ja	Aurora-MySQL-Version 2. Verwenden Sie replica_checkpoint_period in Aurora-MySQL-Version 3.
slave_parallel_workers	Ja	Aurora-MySQL-Version 2. Verwenden Sie replica_parallel_workers in Aurora-MySQL-Version 3.
slave_pending_jobs_size_max	Ja	Aurora-MySQL-Version 2. Verwenden Sie replica_pending_jo bs_size_max in Aurora-MySQL-Version 3.

Parametername	Anpassbar	Hinweise
slave_sql_verify_checksum	Ja	Aurora-MySQL-Version 2. Verwenden Sie replica_sql_verify _checksum in Aurora-MySQL-Version 3.
slow_launch_time	Ja	Keine
slow_query_log	Ja	Anweisungen zum Hochladen von Protokollen in CloudWatch Logs finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.
slow_query_log_file	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.
socket	Nein	Keine
sort_buffer_size	Ja	Keine
sql_mode	Ja	Keine
sql_select_limit	Ja	Keine
stored_program_cache	Ja	Keine
sync_binlog	Nein	Keine
sync_master_info	Ja	Keine
sync_source_info	Ja	Dieser Parameter gilt für Aurora MySQL Version 3.
sync_relay_log	Ja	Aus Aurora-MySQL-Version 3 entfernt.
sync_relay_log_info	Ja	Keine

Parametername	Anpassbar	Hinweise
`sysdate-is-now`	Ja	Keine
`table_cache_element_entry_ttl`	Nein	Keine
`table_definition_cache`	Ja	Der Standardwert wird durch eine Formel dargestellt. Mehr über die Berechnung des `DBInstanceClassMemory` -Werts in der Formel erfahren Sie unter DB-Parameter-Formel-Variablen.
`table_open_cache`	Ja	Der Standardwert wird durch eine Formel dargestellt. Mehr über die Berechnung des `DBInstanceClassMemory` -Werts in der Formel erfahren Sie unter DB-Parameter-Formel-Variablen.
`table_open_cache_instances`	Ja	Keine
`temp-pool`	Ja	Aus Aurora-MySQL-Version 3 entfernt.
`temptable_max_mmap`	Ja	Dieser Parameter gilt für Aurora-MySQL-Version 3. Details hierzu finden Sie unter Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3.
`temptable_max_ram`	Ja	Dieser Parameter gilt für Aurora-MySQL-Version 3. Details hierzu finden Sie unter Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3.

Parametername	Anpassbar	Hinweise
temptable_use_mmap	Ja	Dieser Parameter gilt für Aurora-MySQL-Version 3. Details hierzu finden Sie unter Neues temporäres Tabellenverhalten in Aurora-MySQL-Version 3.
thread_cache_size	Ja	Die Anzahl der Threads, die zwischengespeichert werden sollen. Dieser Parameter gilt für Aurora MySQL Version 2 und 3.
thread_handling	Nein	Keine
thread_stack	Ja	Keine
timed_mutexes	Ja	Keine
tmp_table_size	Ja	Definiert die maximale Größe temporärer Tabellen, die von der MEMORY-Speicher-Engine in Aurora-MySQL-Version 3 erstellt werden. Definiert in Aurora-MySQL-Version 3.04 und höher die maximale Größe temporärer Tabellen im Arbeitsspeicher, die von der TempTable -Speicher-Engine erstellt werden, wenn aurora_tmptable_enable_per_table_limit auf ON festgelegt ist. Weitere Informationen finden Sie unter Begrenzung der Größe interner temporärer Tabellen im Arbeitsspeicher.
tmpdir	Nein	Aurora MySQL verwendet verwaltete Instances, bei denen Sie nicht direkt auf das Archivsystem zugreifen.

Parametername	Anpassbar	Hinweise
transaction_alloc_block_size	Ja	Keine
transaction_isolation	Ja	Dieser Parameter gilt für Aurora MySQL Version 3. Er ersetzt tx_isolation .
transaction_prealloc_size	Ja	Keine
tx_isolation	Ja	Aus Aurora-MySQL-Version 3 entfernt. Es wird durch transaction_isolation ersetzt.
updatable_views_with_limit	Ja	Keine
validate-password	Nein	Keine
validate_password_dictionary_file	Nein	Keine
validate_password_length	Nein	Keine
validate_password_mixed_case_count	Nein	Keine
validate_password_number_count	Nein	Keine
validate_password_policy	Nein	Keine
validate_password_special_char_count	Nein	Keine

Parametername	Anpassbar	Hinweise
`wait_timeout`	Ja	Aurora wertet den Mindestwert von `interactive_timeout` und `wait_timeout` aus. Dieses Minimum wird dann als Timeout verwendet, um alle inaktiven Sitzungen, sowohl interaktive als auch nicht interaktive, zu beenden.

MySQL-Parameter, die nicht auf Aurora MySQL zutreffen

Aufgrund von Architekturunterschieden zwischen Aurora MySQL und MySQL gelten einige MySQL-Parameter nicht für Aurora MySQL.

Die folgenden MySQL-Parameter gelten nicht für Aurora MySQL. Diese Liste ist nicht umfassend.

- `activate_all_roles_on_login` – Dieser Parameter gilt nicht für Aurora-MySQL-Version 2. Er ist in Aurora-MySQL-Version 3 verfügbar.
- `big_tables`
- `bind_address`
- `character_sets_dir`
- `innodb_adaptive_flushing`
- `innodb_adaptive_flushing_lwm`
- `innodb_buffer_pool_chunk_size`
- `innodb_buffer_pool_instances`
- `innodb_change_buffering`
- `innodb_checksum_algorithm`
- `innodb_data_file_path`
- `innodb_dedicated_server`
- `innodb_doublewrite`
- `innodb_flush_log_at_timeout`: Dieser Parameter gilt nicht für Aurora MySQL. Weitere Informationen finden Sie unter Konfigurieren, wie oft der Protokollpuffer geleert wird.
- `innodb_flush_method`

- innodb_flush_neighbors
- innodb_io_capacity
- innodb_io_capacity_max
- innodb_log_buffer_size
- innodb_log_file_size
- innodb_log_files_in_group
- innodb_log_spin_cpu_abs_lwm
- innodb_log_spin_cpu_pct_hwm
- innodb_log_writer_threads
- innodb_max_dirty_pages_pct
- innodb_numa_interleave
- innodb_page_size
- innodb_redo_log_capacity
- innodb_redo_log_encrypt
- innodb_undo_log_encrypt
- innodb_undo_log_truncate
- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_use_native_aio
- innodb_write_io_threads

Globale Statusvariablen von Aurora MySQL

Aurora MySQL enthält Statusvariablen aus Community-MySQL und Variablen, die es nur bei Aurora gibt. Sie können diese Variablen untersuchen, um zu erfahren, was in der Datenbank-Engine passiert. Weitere Informationen zu den Statusvariablen in Community-MySQL finden Sie unter Serverstatusvariablen in der Community-MySQL-8.0-Dokumentation.

Sie können die aktuellen Werte für die globalen Statusvariablen von Aurora MySQL mithilfe einer Anweisung wie den folgenden ermitteln:

```
show global status like '%aurora%';
```

> **Note**
>
> Globale Statusvariablen werden gelöscht, wenn die DB-Engine neu gestartet wird.

In der folgenden Tabelle werden die globalen Statusvariablen beschrieben, die Aurora MySQL verwendet.

Name	Beschreibung
`AuroraDb_commits`	Die Gesamtzahl der Commits seit dem letzten Neustart.
`AuroraDb_commit_latency`	Die aggregierte Commit-Latenz seit dem letzten Neustart.
`AuroraDb_ddl_stmt_duration`	Die aggregierte DDL-Latenz seit dem letzten Neustart.
`AuroraDb_select_stmt_duration`	Die aggregierte `SELECT`-Anweisungslatenz seit dem letzten Neustart.
`AuroraDb_insert_stmt_duration`	Die aggregierte `INSERT`-Anweisungslatenz seit dem letzten Neustart.
`AuroraDb_update_stmt_duration`	Die aggregierte `UPDATE`-Anweisungslatenz seit dem letzten Neustart.
`AuroraDb_delete_stmt_duration`	Die aggregierte `DELETE`-Anweisungslatenz seit dem letzten Neustart.
`Aurora_binlog_io_cache_allocated`	Die Anzahl der Byte, die dem Binlog-I/O-Cache zugewiesen wurden.
`Aurora_binlog_io_cache_read_requests`	Die Anzahl der Leseanfragen an den Binlog-I/OCache.
`Aurora_binlog_io_cache_reads`	Die Anzahl der Leseanfragen, die der Binlog-I/O-Cache verarbeitet hat.

Name	Beschreibung
`Aurora_enhanced_binlog`	Gibt an, ob das erweiterte Binärprotokoll für diese DB-Instance aktiviert oder deaktiviert ist. Weitere Informationen finden Sie unter Einrichten eines erweiterten Binärprotokolls für Aurora MySQL.
`Aurora_external_connection_count`	Die Anzahl der Datenbankverbindungen mit der DB-Instance, ausgenommen RDS-Servi ceverbindungen, die für Datenbankzustandsp rüfungen verwendet werden.
`Aurora_fast_insert_cache_hits`	Ein Zähler, der erhöht wird, wenn der zwischengespeicherte Cursor erfolgreich abgerufen und bestätigt wurde. Weitere Informationen zum Cache für schnelles Einfügen finden Sie unter Amazon Aurora MySQL-Leistungserweiterungen.
`Aurora_fast_insert_cache_misses`	Ein Zähler, der erhöht wird, wenn der zwischengespeicherte Cursor nicht mehr gültig ist und Aurora einen normalen Index-Durchlauf durchführt. Weitere Informationen zum Cache für schnelles Einfügen finden Sie unter Amazon Aurora MySQL-Leistungserweiterungen.
`Aurora_fts_cache_memory_used`	Die Speichermenge in Byte, die das InnoDB-Vo lltextsuchsystem verwendet. Diese Variable gilt für Aurora-MySQL-Version 3.07 und höher.
`Aurora_fwd_master_dml_stmt_count`	Die Gesamtzahl der DML-Anweisungen, die an diese Schreiber-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 2.

Name	Beschreibung
`Aurora_fwd_master_dml_stmt_duration`	Die Gesamtdauer der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 2.
`Aurora_fwd_master_errors_rpc_timeout`	Gibt an, wie oft eine weitergeleitete Verbindung nicht auf dem Writer hergestellt werden konnte.
`Aurora_fwd_master_errors_session_limit`	Die Anzahl der weitergeleiteten Abfragen, die aufgrund von `session full` auf dem Writer abgelehnt werden.
`Aurora_fwd_master_errors_session_timeout`	Die Angabe, wie oft eine Weiterleitungssitzung aufgrund eines Timeouts auf dem Writer beendet wird.
`Aurora_fwd_master_open_sessions`	Die Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance. Diese Variable gilt für Aurora-MySQL-Version 2.
`Aurora_fwd_master_select_stmt_count`	Die Gesamtzahl der `SELECT`-Anweisungen, die an diese Schreiber-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 2.
`Aurora_fwd_master_select_stmt_duration`	Die Gesamtdauer der `SELECT`-Anweisungen, die an diese Schreiber-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 2.
`Aurora_fwd_writer_dml_stmt_count`	Die Gesamtzahl der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 3.

Name	Beschreibung
`Aurora_fwd_writer_dml_stmt_duration`	Die Gesamtdauer der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 3.
`Aurora_fwd_writer_errors_rpc_timeout`	Gibt an, wie oft eine weitergeleitete Verbindung nicht auf dem Writer hergestellt werden konnte.
`Aurora_fwd_writer_errors_session_limit`	Die Anzahl der weitergeleiteten Abfragen, die aufgrund von `session full` auf dem Writer abgelehnt werden.
`Aurora_fwd_writer_errors_session_timeout`	Die Angabe, wie oft eine Weiterleitungssitzung aufgrund eines Timeouts auf dem Writer beendet wird.
`Aurora_fwd_writer_open_sessions`	Die Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance. Diese Variable gilt für Aurora-MySQL-Version 3.
`Aurora_fwd_writer_select_stmt_count`	Die Gesamtzahl der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 3.
`Aurora_fwd_writer_select_stmt_duration`	Die Gesamtdauer der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Diese Variable gilt für Aurora-MySQL-Version 3.
`Aurora_lockmgr_buffer_pool_memory_used`	Die Menge an Pufferpoolspeicher in Byte, die der Aurora MySQL Lock Manager verwendet.
`Aurora_lockmgr_memory_used`	Die Speichermenge in Byte, die der Aurora MySQL Lock Manager verwendet.

Name	Beschreibung
`Aurora_ml_actual_request_cnt`	Die aggregierte Zahl der Anfragen, die Aurora MySQL an die Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg stellt. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.
`Aurora_ml_actual_response_cnt`	Die aggregierte Zahl der Antworten, die Aurora MySQL von den Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg erhält. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.
`Aurora_ml_cache_hit_cnt`	Die aggregierte Zahl der internen Cache-Treffer, die Aurora MySQL von den Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg erhält. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.
`Aurora_ml_logical_request_cnt`	Die Anzahl der logischen Anfragen, die die DB-Instance seit dem letzten Status-Reset zum Senden an die Aurora-Machine-Learning-Services ausgewertet hat. Je nachdem, ob Stapelverarbeitung verwendet wurde, kann dieser Wert höher sein als `Aurora_ml_actual_request_cnt`. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.

Name	Beschreibung
`Aurora_ml_logical_response_cnt`	Die aggregierte Zahl der Antworten, die Aurora MySQL von den Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg erhält. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.
`Aurora_ml_retry_request_cnt`	Die Anzahl der wiederholten Anfragen, die die DB-Instance seit dem letzten Status-Reset an die Aurora-Machine-Learning-Services gesendet hat. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.
`Aurora_ml_single_request_cnt`	Die aggregierte Zahl der Machine-Learning-Funktionen von Aurora, die im Nicht-Batch-Modus über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg evaluiert werden. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL.
`aurora_oom_avoidance_recovery_state`	Gibt an, ob sich die Wiederherstellung nach Vermeidung eines Speichermangels von Aurora für diese DB-Instance im Status `INACTIVE` oder `ACTIVE` befindet. Diese Variable gilt für Aurora-MySQL-Version 3.06.0 und höher.

Name	Beschreibung
`aurora_oom_reserved_mem_enter_kb`	Stellt den Schwellenwert für den Eintritt in den `RESERVED`-Status im Mechanismus zur Behandlung eines Speichermangels von Aurora dar. Wenn der verfügbare Speicher auf dem Server unter diesen Schwellenwert fällt, ändert sich der `aurora_oom_status` in `RESERVED`, was darauf hinweist, dass sich der Server einem kritischen Speicherverbrauchsniveau nähert. Diese Variable gilt für Aurora-MySQL-Version 3.06.0 und höher.
`aurora_oom_reserved_mem_exit_kb`	Stellt den Schwellenwert für den Austritt aus dem `RESERVED`-Status im Mechanismus zur Behandlung eines Speichermangels von Aurora dar. Wenn der verfügbare Speicher auf dem Server über diesen Schwellenwert steigt, kehrt der `aurora_oom_status` zu `NORMAL` zurück, was bedeutet, dass der Server in einen stabileren Zustand mit ausreichenden Speicherressourcen zurückgekehrt ist. Diese Variable gilt für Aurora-MySQL-Version 3.06.0 und höher.

Name	Beschreibung
`aurora_oom_status`	Stellt den aktuellen Status eines Speicherm angels dieser DB-Instance dar. Wenn der Wert `NORMAL` lautet, bedeutet dies, dass ausreiche nd Speicherressourcen vorhanden sind. Ändert sich der Wert in `RESERVED`, bedeutet dies, dass der Server nur wenig verfügbaren Arbeitsspeicher hat. Die Aktionen werden auf der Grundlage der `aurora_oom_respons e` -Parameterkonfiguration ausgeführt. Weitere Informationen finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken. Diese Variable gilt für Aurora-MySQL-Versi on 3.06.0 und höher.
`Aurora_pq_bytes_returned`	Die Anzahl der Bytes, die in Zusammenh ang mit Tupel-Datenstrukturen während der Parallelabfragen an den Hauptknoten übertrage n wurden. Für den Abgleich mit durch 16 348 teile `Aurora_pq_pages_pushed_down` .
`Aurora_pq_max_concurrent_re quests`	Die Höchstanzahl der Parallel Query-Sit zungen, die auf dieser Aurora-DB-Instance gleichzeitig ausgeführt werden können. Diese Zahl ist festgelegt und hängt von der AWS-DB-Instance-Klasse ab.
`Aurora_pq_pages_pushed_down`	Die Anzahl der Datenseiten (jeweils fix 16 KiB groß), auf denen Parallel Query eine netzwerkg ebundene Übertragung an den Hauptknoten verhindert hat.

Name	Beschreibung
`Aurora_pq_request_attempted`	Die Anzahl der angeforderten Parallel Query-Sitzungen. Hinter diesem Wert können pro Abfrage mehrere Sitzungen stehen. Ausschlag gebend sind SQL-Konstrukte wie Unterabfr agen und Join-Abfragen.
`Aurora_pq_request_executed`	Die Anzahl der erfolgreich ausgeführten Parallel Query-Sitzungen.
`Aurora_pq_request_failed`	Die Anzahl der Parallel Query-Sitzungen, die dem Client einen Fehler zurückgaben. Es kann vorkommen, dass eine angeforderte Parallelabfrage fehlschlägt – z. B. wegen eines Problems auf der Speicherschicht. In diesen Fällen wird der fehlgeschlagene Abfrageteil wiederholt. Dafür kommt der nicht-parallele Abfragemechanismus zum Einsatz. Wenn auch die wiederholte Abfrage fehlschlägt, wird dem Client ein Fehler zurückgegeben. Die Zähleranz eige erhöht sich.
`Aurora_pq_request_in_progress`	Die Anzahl der derzeit in Ausführung befindlic hen Parallel Query-Sitzungen. Die Zahl bezieht sich auf die angeschlossene Aurora-DB-Instance, mit der Sie verbunden sind, nicht auf das gesamte Aurora-DB-Cluster. Sie können prüfen, ob eine DB-Instance die Obergrenze für gleichzeitige Abfragen annähernd erreicht hat. Gleichen Sie dazu diesen Wert mit a `Aurora_pq_max_concurrent_re quests` .

Name	Beschreibung
`Aurora_pq_request_not_chosen`	Die Anzahl der Situationen, in denen Parallel Query nicht für die Abarbeitung einer Abfrage genutzt wurde. Dieser Wert setzt sich aus den Beiträgen mehrerer Zähler mit höherer Granularität zusammen. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chosen_below_min_rows`	Die Anzahl der Situationen, in denen die Anzahl der Tabellenzeilen der Grund für den Nichteinsatz von Parallel Query war. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chosen_column_bit`	Die Anzahl der Parallelabfrageanforderungen, die aufgrund eines nicht unterstützten Datentyps in der Liste der projizierten Spalten den nicht-parallelen Abfrageverarbeitungspfad nutzen.
`Aurora_pq_request_not_chosen_column_geometry`	Die Anzahl der Parallelabfrageanforderungen, die den nicht-parallelen Abfrageverarbeitungspfad nutzen, da die Tabelle Spalten mit dem Datentyp `GEOMETRY` enthält. Informationen zu Aurora-MySQL-Versionen, die diese Einschränkung aufheben, finden Sie unter Upgrade paralleler Abfrage-Cluster auf Aurora-MySQL-Version 3.

Name	Beschreibung
`Aurora_pq_request_not_chosen_column_lob`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle Spalten mit einem `LOB`-Datentyp enthält oder `VARCHAR`-Spalten, die aufgrund der deklarierten Länge extern gespeichert werden. Informationen zu Aurora-MySQL-Versionen, die diese Einschränkung aufheben, finden Sie unter Upgrade paralleler Abfrage-Cluster auf Aurora-MySQL-Version 3.
`Aurora_pq_request_not_chosen_column_virtual`	Die Anzahl der Parallelabfrageanforderungen, die den nicht-parallelen Abfrageverarbeitungspfad nutzen, da die Tabelle eine virtuelle Spalte enthält.
`Aurora_pq_request_not_chosen_custom_charset`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle Spalten mit einem benutzerdefinierten Zeichensatz enthält.
`Aurora_pq_request_not_chosen_fast_ddl`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle zurzeit durch eine schnelle `ALTER`-DDL-Anweisung geändert wird.
`Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool`	Die Anzahl der Situationen, in denen Parallel Query nicht genutzt wurde, obwohl weniger als 95 Prozent der Tabellendaten im Bufferpool waren. Grund: Es gab zu wenig nicht gepufferte Tabellendaten. Eine Parallelabfrage lohnte sich deshalb nicht.

Name	Beschreibung
`Aurora_pq_request_not_chosen_full_text_index`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle Volltextindizes enthält.
`Aurora_pq_request_not_chosen_high_buffer_pool_pct`	Die Anzahl der Situationen, in denen Parallel Query nicht genutzt wurde, weil ein hoher Anteil der Tabellendaten (derzeit >95 Prozent) bereits im Bufferpool war. In diesen Fällen entscheidet der Optimierer, dass das Lesen der Daten aus dem Bufferpool effizienter ist. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chosen_index_hint`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage einen Indexhinweis enthält.
`Aurora_pq_request_not_chosen_innodb_table_format`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Tabelle ein nicht unterstütztes InnoDB-Zeilenformat verwendet. Die parallele Aurora-Abfrage gilt nur für die `COMPACT`-, `REDUNDANT` - und `DYNAMIC`-Zeilenformate.
`Aurora_pq_request_not_chosen_long_trx`	Die Anzahl der Parallelabfrageanforderungen, die den nicht-parallelen Abfrageverarbeitungspfad nutzten, weil die Abfrage in einer lang laufenden Transaktion gestartet wurde. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.

Name	Beschreibung
`Aurora_pq_request_not_chosen_no_where_clause`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage keine `WHERE`-Klausel enthält.
`Aurora_pq_request_not_chosen_range_scan`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage einen Bereichsscan für einen Index verwendet.
`Aurora_pq_request_not_chosen_row_length_too_long`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Gesamtlänge aller Spalten zu groß ist.
`Aurora_pq_request_not_chosen_small_table`	Die Anzahl der Situationen, in denen die Gesamtgröße der Tabelle (Zeilenanzahl und durchschnittliche Zeilenlänge) der Grund für den Nichteinsatz von Parallel Query war. Die Zähleranzeige kann mit einer `EXPLAIN`-Anweisung erhöht werden, selbst wenn die Abfrage nicht tatsächlich ausgeführt wird.
`Aurora_pq_request_not_chosen_temporary_table`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage auf temporäre Tabellen verweist, die die nicht unterstützten Tabellentypen `MyISAM` oder `memory` verwenden.

Name	Beschreibung
`Aurora_pq_request_not_chosen_tx_isolation`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage eine nicht unterstützte Transaktionsisolationsstufe verwendet. Bei DB-Leser-Instances wird die parallele Abfrage nur auf die Isolationsstufen `REPEATABLE READ` und `READ COMMITTED` angewendet.
`Aurora_pq_request_not_chosen_update_delete_stmts`	Die Anzahl der parallelen Abfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da die Abfrage Teil einer `UPDATE`- oder `DELETE`-Anweisung ist.
`Aurora_pq_request_not_chosen_unsupported_access`	Die Anzahl der Parallelabfrageanforderungen, die den nicht-parallelen Abfrageverarbeitungspfad nutzen, weil die `WHERE`-Klausel nicht den Kriterien einer Parallelabfrage gerecht wird. Dieses Ergebnis kann eintreten, wenn für die Abfrage kein datenintensiver Scan erforderlich ist oder wenn es sich bei der Abfrage um eine `DELETE`- oder `UPDATE`-Anweisung handelt.
`Aurora_pq_request_not_chosen_unsupported_storage_type`	Die Anzahl der Parallelabfrageanforderungen, die den nicht parallelen Abfrageverarbeitungspfad verwenden, da der DB-Cluster von Aurora MySQL keine unterstützte Aurora-Cluster-Speicherkonfiguration verwendet. Weitere Informationen finden Sie unter Einschränkungen. Dieser Parameter gilt für Aurora-MySQL-Version 3.04 und höher.

Name	Beschreibung
`Aurora_pq_request_throttled`	Die Anzahl der Situationen, in denen Parallel Query nicht genutzt wurde, weil die Höchstmenge gleichzeitiger Parallelabfragen bereits auf einer bestimmten Aurora-DB-Instance ausgeführt wird.
`Aurora_repl_bytes_received`	Anzahl der Byte, die seit dem letzten Neustart auf eine Reader-Datenbank-Instance von Aurora MySQL repliziert wurden. Weitere Informationen finden Sie unter Replikation mit Amazon Aurora My SQL.
`Aurora_reserved_mem_exceeded_incidents`	Gibt an, wie oft die Engine seit dem letzten Neustart die Grenzen des reservierten Speichers überschritten hat. Wenn `aurora_oom_response` konfiguriert ist, definiert dieser Schwellenwert, wann Aktivitäten zur Vermeidung von Speichermangel ausgelöst werden. Weitere Informationen zur Antwort wegen Speichermangel von Aurora MySQL finden Sie unter Beheben von Speicherproblemen bei Aurora-MySQL-Datenbanken.
`aurora_temptable_max_ram_allocation`	Die maximale Speichermenge in Byte, die zu einem beliebigen Zeitpunkt von internen temporären Tabellen seit dem letzten Neustart verwendet wird.
`aurora_temptable_ram_allocation`	Die aktuelle Speichermenge in Byte, die von internen temporären Tabellen verwendet wird.
`Aurora_in_memory_relaylog_status`	Der aktuelle Status der In-Memory-Relay-Protokollfunktion. Der Wert kann ENABLED oder DISABLED sein.

Name	Beschreibung
`Aurora_in_memory_relaylog_disabled_reason`	Zeigt den Grund für den aktuellen Status der In-Memory-Relay-Protokollfunktion an. Wenn die Funktion deaktiviert ist, wird eine Meldung mit einer Erklärung angezeigt, warum die Funktion deaktiviert ist.
`Aurora_in_memory_relaylog_fallback_count`	Zeigt die Gesamtzahl der Fallbacks der In-Memory-Relay-Protokollfunktion in den persistenten Relay-Protokollmodus (veraltet) an. Ein Fallback kann entweder durch ein einzelnes Ereignis verursacht werden, das größer als die Cachegröße (derzeit 128 MB) ist, oder wenn eine Transaktionswiederholung das Wiederholungslimit für Replikattransaktionen replica_transaction_retries überschreitet.
`Aurora_in_memory_relaylog_recovery_count`	Zeigt die Gesamtzahl der In-Memory-Relay-Protokollwiederherstellungen an, die automatisch durchgeführt wurden. Diese Anzahl umfasst die Gesamtzahl der Fallbacks sowie die Anzahl der automatischen Modus-Rückschaltungen in den In-Memory-Relay-Protokollmodus nach den temporären Fallbacks.
`Aurora_thread_pool_thread_count`	Die aktuelle Anzahl von Threads im Aurora-Threadpool. Weitere Informationen zum Threadpool in Aurora MySQL finden Sie unter Thread-Pool.

Name	Beschreibung
`Aurora_tmz_version`	Bezeichnet die aktuelle Version der Zeitzoneninformationen, die vom DB-Cluster verwendet werden. Die Werte folgen dem Format der Internet Assigned Numbers Authority (IANA): YYYY`suffix` , zum Beispiel `2022a` und `2023c`. Dieser Parameter gilt für Aurora-MySQL-Version 2.12 und höher sowie Version 3.04 und höher.
`Aurora_zdr_oom_threshold`	Stellt den Speicherschwellenwert in Kilobyte (KB) für eine Aurora-DB-Instance dar, um einen Neustart ohne Ausfallzeit (Zero Downtime Restart, ZDR) zur Bewältigung potenzieller Speicherprobleme einzuleiten.
`server_aurora_das_running`	Gibt an, ob Database Activity Streams (DAS) auf dieser DB-Instance aktiviert oder deaktiviert sind. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.

MySQL-Statusvariablen, die nicht für Aurora MySQL gelten

Aufgrund der architektonischen Unterschiede zwischen Aurora MySQL und MySQL gelten einige MySQL-Statusvariablen nicht für Aurora MySQL.

Die folgenden MySQL-Statusvariablen gelten nicht für Aurora MySQL. Diese Liste ist nicht umfassend.

- `innodb_buffer_pool_bytes_dirty`
- `innodb_buffer_pool_pages_dirty`
- `innodb_buffer_pool_pages_flushed`

Aurora-MySQL-Version 3 entfernt die folgenden Statusvariablen in Aurora-MySQL-Version 2:

- AuroraDb_lockmgr_bitmaps0_in_use
- AuroraDb_lockmgr_bitmaps1_in_use
- AuroraDb_lockmgr_bitmaps_mem_used
- AuroraDb_thread_deadlocks
- available_alter_table_log_entries
- Aurora_lockmgr_memory_used
- Aurora_missing_history_on_replica_incidents
- Aurora_new_lock_manager_lock_release_cnt
- Aurora_new_lock_manager_lock_release_total_duration_micro
- Aurora_new_lock_manager_lock_timeout_cnt
- Aurora_total_op_memory
- Aurora_total_op_temp_space
- Aurora_used_alter_table_log_entries
- Aurora_using_new_lock_manager
- Aurora_volume_bytes_allocated
- Aurora_volume_bytes_left_extent
- Aurora_volume_bytes_left_total
- Com_alter_db_upgrade
- Compression
- External_threads_connected
- Innodb_available_undo_logs
- Last_query_cost
- Last_query_partial_plans
- Slave_heartbeat_period
- Slave_last_heartbeat
- Slave_received_heartbeats
- Slave_retried_transactions
- Slave_running

- `Time_since_zero_connections`

Diese MySQL-Statusvariablen sind in Aurora-MySQL-Version 1 2 verfügbar, in Aurora-MySQL-Version 3 jedoch nicht:

- `Innodb_redo_log_enabled`
- `Innodb_undo_tablespaces_total`
- `Innodb_undo_tablespaces_implicit`
- `Innodb_undo_tablespaces_explicit`
- `Innodb_undo_tablespaces_active`

Aurora-MySQL-Warteereignisse

Es folgen verschiedene typische Warteereignisse für Aurora MySQL.

> **Note**
>
> Informationen zur Optimierung der Leistung von Aurora MySQL mithilfe von Warteereignissen finden Sie unter Optimieren von Aurora MySQL mit Warteereignissen.
> Informationen zu den in MySQL-Warteereignissen geltenden Namenskonventionen finden Sie unter Performance Schema Instrument Naming Conventions in der MySQL-Dokumentation.

cpu

Die Anzahl der aktiven Verbindungen, die betriebsbereit sind, ist durchweg höher als die Anzahl von CPUs v. Weitere Informationen finden Sie untercpu.

io/aurora_redo_log_flush

Eine Sitzung enthält Daten für den Aurora-Speicher. Typischerweise bezieht sich dieses Warteereignis auf eine Schreib-I/O-Operation in Aurora MySQL. Weitere Informationen finden Sie unter io/aurora_redo_log_flush.

io/aurora_respond_to_client

Die Abfrageverarbeitung ist abgeschlossen und die Ergebnisse werden für die folgenden Aurora-MySQL-Version an den Anwendungs-Client zurückgegeben: 2.10.2 und höhere 2.10-

Versionen, 2.09.3 und höhere 2.09-Versionen, 2.07.7 und höhere 2.07-Versionen. Vergleichen Sie die Netzwerkbandbreite der DB-Instance-Klasse mit der Größe der zurückgegebenen Ergebnismenge. Überprüfen Sie auch clientseitige Reaktionszeiten. Wenn der Client nicht reagiert und die TCP-Pakete nicht verarbeiten kann, können Paketabfälle und TCP-Neuübertragungen auftreten. Diese Situation wirkt sich negativ auf die Netzwerkbandbreite aus. In Versionen vor 2.10.2, 2.09.3 und 2.07.7 enthält das Warteereignis fälschlicherweise die Leerlaufzeit. Um zu erfahren, wie Sie Ihre Datenbank optimieren, wenn diese Wartezeit im Vordergrund steht, lesen Sie io/aurora_respond_to_client.

io/file/csv/data

Threads schreiben in Tabellen im CSV-Format (Comma-Separated Value). Prüfen Sie die Auslastung der CSV-Tabellen. Eine typische Ursache für dieses Ereignis ist die Einstellung `log_output` auf einer Tabelle.

io/file/sql/binlog

Ein Thread wartet auf eine binäre Protokoll-Datei (binlog), die auf den Datenträger geschrieben wird.

io/redo_log_flush

Eine Sitzung enthält Daten für den Aurora-Speicher. In der Regel bezieht sich dieses Warteereignis auf einen I/O Schreibvorgang in Aurora MySQL. Weitere Informationen finden Sie unter io/redo_log_flush.

io/socket/sql/client_Verbindung

Das Programm `mysqld` ist damit beschäftigt, Threads zu erstellen, um eingehende neue Client-Verbindungen zu verarbeiten. Weitere Informationen finden Sie unter io/socket/sql/client_connection.

io/table/sql/handler

Der Motor wartet auf Zugang zu einer Tabelle. Dieses Ereignis tritt unabhängig davon auf, ob die Daten im Pufferpool zwischengespeichert oder auf der Festplatte zugegriffen werden. Weitere Informationen finden Sie unter io/table/sql/handler.

lock/table/sql/handler

Dieses Warteereignis ist ein Warteereignis-Handler für Tabellensperren. Weitere Informationen zu Atom- und Molekülereignissen im Leistungsschema finden Sie unter Leistungsschema-Atom- und Molekülereignisse in der MySQL-Dokumentation.

synch/cond/innodb/row_lock_wait

Mehrere DML-Anweisungen (DML = Data Manipulation Language) greifen gleichzeitig auf dieselben Datenbankzeilen zu. Weitere Informationen finden Sie unter synch/cond/innodb/row_lock_wait.

synch/cond/innodb/row_lock_wait_cond

Mehrere DML-Anweisungen greifen gleichzeitig auf dieselben Datenbankzeilen zu. Weitere Informationen finden Sie unter synch/cond/innodb/row_lock_wait_cond.

synch/cond/sql/MDL_context: :cond_wait_status

Threads warten auf eine Tabellen-Metadatensperre. Die Engine verwendet diese Art von Sperre, um den gleichzeitigen Zugriff auf ein Datenbankschema zu verwalten und die Datenkonsistenz sicherzustellen. Weitere Informationen finden Sie unter Optimizing Locking Operations in der MySQL-Dokumentation. Um zu erfahren, wie Sie Ihre Datenbank optimieren, wenn dieses Ereignis im Vordergrund steht, lesen Sie synch/cond/sql/MDL_context::COND_wait_status.

synch/cond/sql/MYSQL_BIN_LOG: :cond_DONE

Sie haben die binäre Protokollierung aktiviert. Es kann einen hohen Commit-Durchsatz, eine große Anzahl von Transaktionen oder Replikate geben, die Binlogs lesen. Erwägen Sie, mehrzeilige Anweisungen zu verwenden oder Anweisungen in einer Transaktion zu bündeln. Verwenden Sie in Aurora globale Datenbanken anstelle der Binärprotokollreplikation oder verwenden Sie die Parameter `aurora_binlog_*`.

synch/mutex/innodb/aurora_lock_thread_slot_futex

Mehrere DML-Anweisungen greifen gleichzeitig auf dieselben Datenbankzeilen zu. Weitere Informationen finden Sie unter synch/mutex/innodb/aurora_lock_thread_slot_futex.

synch/mutex/innodb/buf_pool_mutex

Der Puffer-Pool ist nicht groß genug, um den Arbeitsdatensatz zu speichern. Oder die Workload greift auf Seiten aus einer bestimmten Tabelle zu, was zu Konflikten im Pufferpool führt. Weitere Informationen finden Sie unter synch/mutex/innodb/buf_pool_mutex.

synch/mutex/innodb/fil_system_mutex

Der Prozess wartet auf den Zugriff auf den Tablespace-Speicher-Cache. Weitere Informationen finden Sie unter synch/mutex/innodb/fil_system_mutex.

synch/mutex/innodb/trx_sys_mutex

Operationen sind das Überprüfen, Aktualisieren, Löschen oder Hinzufügen von Transaktionen IDs in InnoDB auf konsistente oder kontrollierte Weise. Diese Vorgänge erfordern einen `trx_sys`-Mutex-Aufruf, der von der Leistungs-Schema-Instrumentierung verfolgt wird. Zu den Vorgängen gehören die Verwaltung des Transaktionssystems beim Starten oder Herunterfahren der Datenbank, Rollbacks, Rückgängig-Bereinigungen, Zeilen-Lesezugriff und Pufferpool-Lasten. Eine hohe Datenbanklast bei einer großen Anzahl von Transaktionen führt zum häufigen Auftreten dieses Wait-Ereignisses. Weitere Informationen finden Sie unter synch/mutex/innodb/trx_sys_mutex.

synch/mutex/mysys/KEY_CACHE: :cache_lock

Der `keycache->cache_lock`-Mutex steuert den Zugriff auf den Schlüsselcache für MyISAM-Tabellen. Aurora MySQL erlaubt zwar keine MyISAM-Tabellen zum Speichern persistenter Daten, diese werden jedoch zum Speichern interner temporärer Tabellen verwendet. Prüfen Sie ggf. die Statuszähler `created_tmp_tables` oder `created_tmp_disk_tables`, da in bestimmten Situationen temporäre Tabellen auf die Festplatte geschrieben werden, wenn sie nicht mehr in den Speicher passen.

synch/mutex/sql/FILE_AS_TABLE: :Lock_Offsets

Die Engine erwirbt diesen Mutex beim Öffnen oder Erstellen eines Tabellen-Metadatenarchivs. Wenn dieses Warteereignis mit übermäßiger Häufigkeit auftritt, ist die Anzahl der erstellten oder geöffneten Tabellen gestiegen.

synch/mutex/sql/FILE_AS_TABLE: :Lock_SHIM_LISTS

Die Engine ruft diesen Mutex ab, während sie Vorgänge wie `reset_size`, `detach_contents` oder `add_contents` an der internen Struktur durchführt, die geöffnete Tabellen verfolgt. Der Mutex synchronisiert den Zugriff auf den Listeninhalt. Wenn dieses Warteereignis mit hoher Frequenz auftritt, deutet dies auf eine plötzliche Änderung des Tabellensatzes hin, auf die zuvor zugegriffen wurde. Die Engine muss auf neue Tabellen zugreifen oder den Kontext, der sich auf zuvor aufgerufene Tabellen bezieht, loslassen.

synch/mutex/sql/LOCK_öffnen

Die Anzahl der Tabellen, die Ihre Sitzungen öffnen, überschreitet die Größe des Tabellendefinitions-Caches oder des Caches zum Öffnen der Tabelle. Erhöhen Sie die Größe dieser Caches. Weitere Informationen finden Sie unter How MySQL Opens and Closes Tables.

synch/mutex/sql/LOCK_Tabellencache

Die Anzahl der Tabellen, die Ihre Sitzungen öffnen, überschreitet die Größe des Tabellendefinitions-Caches oder des Caches zum Öffnen der Tabelle. Erhöhen Sie die Größe dieser Caches. Weitere Informationen finden Sie unter How MySQL Opens and Closes Tables.

synch/mutex/sql/LOG

Bei diesem Warteereignis warten Threads auf eine Protokollsperre. Beispiel: Ein Thread wartet auf eine Sperre, um in die Slow-Query-Protokolldatei zu schreiben.

synch/mutex/sql/MYSQL_BIN_LOG: :Lock_Commit

Bei diesem Warteereignis wartet ein Thread auf eine Sperre, um Daten in das Binärprotokoll schreiben zu können. Konflikte in der Binärprotokollierung können bei Datenbanken mit sehr hoher Änderungsrate auftreten. In Abhängigkeit von der MySQL-Version werden verschiedene Sperren verwendet, um Konsistenz und Haltbarkeit des Binärprotokolls zu schützen. In RDS für MySQL werden Binärprotokolle für die Replikation und für automatische Sicherungen verwendet. In Aurora MySQL werden keine Binärprotokolle für die native Replikation und für Sicherungen benötigt. Sie sind standardmäßig deaktiviert, können aber aktiviert und für die externe Replikation oder die Erfassung von Änderungsdaten genutzt werden. Weitere Informationen finden Sie unter The Binary Log in der MySQL-Dokumentation.

sync/mutex/sql/MYSQL_BIN_LOG: :Lock_Dump_Thread_Metrics_Collection

Wenn die binäre Protokollierung aktiviert ist, erwirbt die Engine diesen Mutex, wenn sie aktive Dump-Thread-Metriken an das Engine-Fehlerprotokoll und in die interne Vorgangs-Mappe druckt.

sync/mutex/sql/MYSQL_BIN_LOG: :Lock_Inactive_BinLogs_Map

Wenn die binäre Protokollierung aktiviert ist, erwirbt die Engine diesen Mutex, wenn sie die Liste der Binlog-Datei hinter der neuesten hinzufügt, sie löscht oder durchsucht.

sync/mutex/sql/MYSQL_BIN_LOG: :IO_Cache sperren

Wenn die binäre Protokollierung aktiviert ist, erwirbt die Engine diesen Mutex während der Aurora-Binlog-IO-Cache-Vorgänge: Zuweisen, Größe ändern, freigeben, schreiben, lesen, löschen und auf Cache-Informationen zugreifen. Wenn dieses Ereignis häufig auftritt, greift die Engine auf den Cache zu, in dem Binlog-Ereignisse gespeichert werden. Um Wartezeiten zu reduzieren, reduzieren Sie Commits. Versuchen Sie, mehrere Anweisungen in einer einzigen Transaktion zu gruppieren.

synch/mutex/sql/MYSQL_BIN_LOG: :LOG SPERREN

Sie haben die binäre Protokollierung aktiviert. Möglicherweise gibt es einen hohen Commit-Durchsatz, viele Transaktionen oder Replikate, die Binlogs lesen. Erwägen Sie, mehrzeilige Anweisungen zu verwenden oder Anweisungen in einer Transaktion zu bündeln. Verwenden Sie in Aurora globale Datenbanken anstelle der Binärprotokollreplikation oder verwenden Sie die Parameter `aurora_binlog_*`.

synch/mutex/sql/SERVER_THREAD: :LOCK_SYNC

Der `SERVER_THREAD::LOCK_sync`-Mutex wird während des Planens, Verarbeitens oder Startens von Threads für Dateischreibvorgänge erfasst. Das übermäßige Auftreten dieses Wait-Ereignisses weist auf eine erhöhte Schreibaktivität in der Datenbank hin.

synch/mutex/sql/TABLESPACES_THREAD: :lock_sync ----sep----:sperren

Die Engine ruft den `TABLESPACES:lock`-Mutex während der folgenden Tablespace-Vorgänge ab: erstellen, löschen, abschneiden und erweitern. Das übermäßige Auftreten dieses Wait-Ereignisses weist auf eine hohe Häufigkeit von Tablespace-Vorgänge hin. Ein Beispiel ist das Laden einer großen Datenmenge in die Datenbank.

synch/rwlock/innodb/dict

Bei diesem Warteereignis warten Threads auf eine rwlock, die für das InnoDB-Datenwörterbuch gesetzt ist.

synch/rwlock/innodb/dict_operation_lock

Bei diesem Warteereignis warten Threads auf Sperren, die für InnoDB-Datenwörterbuch-Operationen gesetzt sind.

synch/rwlock/innodb/dictsys-RW-Sperre

Eine große Anzahl gleichzeitiger Datensteuerungs-Sprachanweisungen (DCLs) im Datendefinitionssprachencode (DDLs) wird gleichzeitig ausgelöst. Reduzieren Sie die Abhängigkeit der Anwendung DDLs während der regulären Anwendungsaktivität.

synch/rwlock/innodb/index_tree_rw_lock

Eine große Anzahl ähnlicher DML-Anweisungen (Data Manipulation Language) greift gleichzeitig auf dasselbe Datenbankobjekt zu. Versuchen Sie es mit mehrreihigen Anweisungen. Verteilen Sie die Workload auch auf verschiedene Datenbankobjekte. Implementieren Sie beispielsweise die Partitionierung.

synch/sxlock/innodb/dict_Betriebssperre

Eine große Anzahl gleichzeitiger Datensteuerungs-Sprachanweisungen (DCLs) im Datendefinitionssprachencode (DDLs) wird gleichzeitig ausgelöst. Reduzieren Sie die Abhängigkeit der Anwendung DDLs während der regulären Anwendungsaktivität.

synch/sxlock/innodb/dict_sys_lock

Eine große Anzahl gleichzeitiger Datensteuerungs-Sprachanweisungen (DCLs) im Datendefinitionssprachencode (DDLs) wird gleichzeitig ausgelöst. Reduzieren Sie die Abhängigkeit der Anwendung DDLs während der regulären Anwendungsaktivität.

synch/sxlock/innodb/hash_table_locks

Die Sitzung konnte keine Seiten im Puffer-Pool finden. Die Engine muss entweder eine Datei lesen oder die zuletzt verwendete Liste (LRU) für den Pufferpool ändern. Erwägen Sie, die Puffer-Cache-Größe zu erhöhen und die Zugriffspfade für die entsprechenden Abfragen

synch/sxlock/innodb/index_tree_rw_lock

Viele ähnliche DML-Anweisungen (DML = Data Manipulation Language) greifen gleichzeitig auf dasselbe Datenbankobjekt zu. Versuchen Sie es mit mehrreihigen Anweisungen. Verteilen Sie die Workload auch auf verschiedene Datenbankobjekte. Implementieren Sie beispielsweise die Partitionierung.

synch/mutex/innodb/temp_poolmanager_mutex

Dieses Wartungsereignis tritt auf, wenn eine Sitzung darauf wartet, einen Mutex für die Verwaltung des Pools temporärer Sitzungs-Tablespaces zu erhalten.

Weitere Informationen zur Fehlerbehebung bei Synchronisierungs-Warteereignissen finden Sie unter Warum zeigt meine MySQL-DB-Instance eine hohe Anzahl aktiver Sitzungen an, die auf SYNCH-Warteereignisse in Performance Insights warten?.

Aurora-MySQL-Thread-Zustände

Im Folgenden werden einige allgemeine Thread-Zustände für Aurora MySQL aufgeführt.

Berechtigungen werden überprüft

Der Thread prüft, ob der Server über die erforderlichen Berechtigungen zum Ausführen der Anweisung verfügt.

Abfrage-Cache auf Abfrage prüfen

Der Server prüft, ob die aktuelle Abfrage im Abfrage-Cache vorhanden ist.

Bereinigen

Dies ist der endgültige Status einer Verbindung, deren Arbeit abgeschlossen ist, aber vom Client nicht geschlossen wurde. Die beste Lösung besteht darin, die Verbindung explizit im Code zu schließen. Oder Sie stellen einen niedrigeren Wert für `wait_timeout` in Ihrer Parametergruppe ein.

Schließen von Tabellen

Der Thread löscht die geänderten Tabellendaten auf die Festplatte und schließt die verwendeten Tabellen. Wenn dies kein schneller Vorgang ist, überprüfen Sie die Metriken für den Verbrauch der Netzwerkbandbreite im Hinblick auf die Netzwerkbandbreite der Instance-Klasse. Überprüfen Sie außerdem, ob die Parameterwerte für die Parameter `table_open_cache` und `table_definition_cache` erlauben, dass genügend Tabellen gleichzeitig geöffnet sind, damit die Engine Tabellen nicht häufig öffnen und schließen muss. Diese Parameter beeinflussen den Speicherverbrauch der Instance.

Konvertieren von HEAP in myISAM

Die Abfrage konvertiert eine temporäre Tabelle von In-Memory in On-Disk. Diese Konvertierung ist notwendig, da die von MySQL in den Zwischenschritten der Abfrageverarbeitung erstellten temporären Tabellen für den Speicher zu groß wurden. Überprüfen Sie die Werte von `tmp_table_size` und `max_heap_table_size`. In späteren Versionen lautet dieser Threadstatusname `converting HEAP to ondisk`.

Konvertieren von HEAP in Ondisk

Der Thread konvertiert eine interne temporäre Tabelle von einer In-Memory-Tabelle in eine On-Disk-Tabelle.

in tmp-Tabelle kopieren

Der Thread verarbeitet eine `ALTER TABLE`-Anweisung. Dieser Status tritt auf, nachdem die Tabelle mit der neuen Struktur erstellt wurde, aber bevor Zeilen in sie kopiert werden. Für einen Thread in diesem Status können Sie das Leistungsschema verwenden, um Informationen über den Fortschritt des Kopiervorgangs zu erhalten.

Sortierindex erstellen

Aurora MySQL führt eine Sortierung durch, weil es keinen vorhandenen Index verwenden kann, um die `ORDER BY`- oder `GROUP BY`-Klausel einer Abfrage zu erfüllen. Weitere Informationen finden Sie unter Erstellen des Sortierindex.

Tabelle wird erstellt

Der Thread erstellt eine permanente oder temporäre Tabelle.

verzögertes Commit ok fertig

Ein asynchrones Commit in Aurora MySQL hat eine Bestätigung erhalten und ist vollständig.

verzögertes Commit ok initiiert

Der Aurora MySQL-Thread hat den asynchronen Commit-Prozess gestartet, wartet aber auf die Bestätigung. Dies ist normalerweise die echte Commit-Zeit einer Transaktion.

verzögert senden ok fertig

Ein Aurora MySQL-Worker-Thread, der an eine Verbindung gebunden ist, kann freigegeben werden, während eine Antwort an den Client gesendet wird. Der Thread kann mit anderen Arbeiten beginnen. Der Zustand `delayed send ok` bedeutet, dass die asynchrone Quittung an den Client abgeschlossen ist.

verzögert senden ok initiiert

Ein Aurora MySQL-Worker-Thread hat asynchron eine Antwort an einen Client gesendet und kann nun für andere Verbindungen arbeiten. Die Transaktion hat einen asynchronen Commit-Prozess gestartet, der noch nicht bestätigt wurde.

executing

Der Thread hat begonnen, eine Anweisung auszuführen.

Freigeben von Gegenständen

Der Thread hat einen Befehl ausgeführt. Ein gewisses Freigeben von Elementen, die in diesem Status durchgeführt wurden, beinhaltet den Abfrage-Cache. Auf diesen Zustand folgt normalerweise ein Aufräumen.

init

Dieser Zustand tritt vor der Initialisierung von `ALTER TABLE`-, `DELETE`-, `INSERT`-, `SELECT`- oder `UPDATE`-Anweisungen auf. Zu den Aktionen in diesem Status gehören das Löschen des Binärprotokolls oder des InnoDB-Protokolls und eine Bereinigung des Abfrage-Caches.

Quelle hat alle Binlogs an das Replikat gesendet; wartet auf weitere Updates

Der Primärknoten hat seinen Teil der Replikation abgeschlossen. Der Thread wartet darauf, dass weitere Abfragen ausgeführt werden, damit er in das Binärprotokoll (Binlog) schreiben kann.

Öffnen von Tabellen

Der Thread versucht eine Tabelle zu öffnen. Dieser Vorgang ist schnell, es sei denn, eine `ALTER TABLE`- oder `LOCK TABLE`-Anweisung muss beendet werden oder sie überschreitet den Wert von `table_open_cache`.

optimieren

Der Server führt erste Optimierungen für eine Abfrage durch.

vorbereiten

Dieser Status tritt während der Abfrageoptimierung auf.

Abfrage Ende

Dieser Status tritt nach der Verarbeitung einer Abfrage jedoch vor dem Status „Freigegebene Elemente“ auf.

Entfernen von Duplikaten

Aurora MySQL konnte einen `DISTINCT`-Vorgang in der frühen Phase einer Abfrage nicht optimieren. Aurora MySQL muss alle duplizierten Zeilen entfernen, bevor das Ergebnis an den Client gesendet wird.

Zeilen nach Update suchen

Der Thread findet alle übereinstimmenden Zeilen, bevor er sie aktualisiert. Diese Phase ist erforderlich, wenn `UPDATE` den Index ändert, den die Engine zum Auffinden der Zeilen verwendet.

Binlog-Ereignis an Slave senden

Der Thread liest ein Ereignis aus dem Binärprotokoll und sendet es an das Replikat.

Zwischengespeichertes Ergebnis an den Client senden

Der Server nimmt das Ergebnis einer Abfrage aus dem Abfrage-Cache und sendet es an den Client.

senden von Daten

Der Thread liest und verarbeitet Zeilen für eine `SELECT`-Anweisung, hat aber noch nicht damit begonnen, Daten an den Client zu senden. Der Prozess identifiziert, welche Seiten die Ergebnisse

enthalten, die erforderlich sind, um die Abfrage zu erfüllen. Weitere Informationen finden Sie unter Senden von Daten.

an den Kunden senden

Der Server schreibt ein Paket an den Client. In früheren MySQL-Versionen wurde dieses Warteereignis mit `writing to net` bezeichnet.

starting

Dies ist die erste Stufe zu Beginn der Ausführung von Anweisungen.

statistics

Der Server berechnet Statistiken, um einen Abfrageausführungsplan zu entwickeln. Wenn sich ein Thread längere Zeit in diesem Zustand befindet, ist der Server wahrscheinlich festplattengebunden, während er andere Arbeiten ausführt.

Speichern von Ergebnis im Abfrage-Cache

Der Server speichert das Ergebnis einer Abfrage im Abfrage-Cache.

Systemsperre

Der Thread hat `mysql_lock_tables` aufgerufen, aber der Threadstatus wurde seit dem Aufruf nicht aktualisiert. Dieser allgemeine Zustand tritt aus vielen Gründen auf.

update

Der Thread bereitet sich darauf vor, die Tabelle zu aktualisieren.

executing

Der Thread sucht nach Zeilen und aktualisiert sie.

Benutzersperre

Der Thread hat einen `GET_LOCK`-Aufruf ausgegeben. Der Thread hat entweder eine Beratungssperre angefordert und wartet darauf oder plant, sie anzufordern.

auf weitere Updates warten

Der Primärknoten hat seinen Teil der Replikation abgeschlossen. Der Thread wartet darauf, dass weitere Abfragen ausgeführt werden, damit er in das Binärprotokoll (Binlog) schreiben kann.

Warten auf Schema-Metadatensperre

Dies ist ein Warten auf eine Metadatensperre.

auf die Sperre der gespeicherten Funktion Metadaten

Dies ist ein Warten auf eine Metadatensperre.

Warten auf Metadatensperre der gespeicherten

Dies ist ein Warten auf eine Metadatensperre.

Warten auf Tabellenintegration

Der Thread führt `FLUSH TABLES` aus und wartet darauf, dass alle Threads ihre Tabellen schließen. Oder der Thread erhielt eine Benachrichtigung, dass sich die zugrunde liegende Struktur für eine Tabelle geändert hat, daher muss er die Tabelle erneut öffnen, um die neue Struktur zu erhalten. Um die Tabelle erneut zu öffnen, muss der Thread warten, bis alle anderen Threads die Tabelle geschlossen haben. Diese Benachrichtigung erfolgt, wenn ein anderer Thread eine der folgenden Anweisungen in der Tabelle verwendet hat: `FLUSH TABLES`, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, oder `OPTIMIZE TABLE`.

auf Tabellen-Levelsperre

Eine Sitzung hält eine Sperre für eine Tabelle, während eine andere Sitzung versucht, dieselbe Sperre für dieselbe Tabelle zu erwerben.

Warten auf Tabellen-Metadatensperre

Aurora MySQL verwendet die Sperre von Metadaten, um den gleichzeitigen Zugriff auf Datenbankobjekte zu verwalten und die Datenkonsistenz sicherzustellen. In diesem Warteereignis hält eine Sitzung eine Metadatensperre für eine Tabelle, während eine andere Sitzung versucht, dieselbe Sperre für dieselbe Tabelle zu erwerben. Wenn das Leistungsschema aktiviert ist, wird dieser Threadstatus als Warteereignis `synch/cond/sql/MDL_context::COND_wait_status` gemeldet.

Schreiben ins Netz

Der Server schreibt ein Paket in das Netzwerk. In späteren MySQL-Versionen wird dieses Warteereignis mit `Sending to client` markiert.

Aurora MySQL-Isolierungsstufen

Im Folgenden erfahren Sie, wie DB-Instances in einem Aurora MySQL-Cluster die Datenbankeigenschaft „Isolation“ implementieren. Dieses Thema erläutert, wie das Standardverhalten von Aurora MySQL einen Ausgleich zwischen strenger Konsistenz und

hoher Leistung sucht. Mithilfe dieser Informationen können Sie entscheiden, wann je nach den Eigenschaften Ihres Workloads die Standardeinstellungen geändert werden sollten.

Verfügbare Isolierungsstufen für Writer-Instances

Sie können die Isolierungsstufen `REPEATABLE READ`, `READ COMMITTED`, `READ UNCOMMITTED` und `SERIALIZABLE` für die primäre Instance eines Aurora MySQL DB-Clusters verwenden. Diese Isolierungsstufen funktionieren in Aurora MySQL genauso wie in RDS für MySQL.

Isolierungsstufe REPEATABLE READ für Reader-Instances

Standardmäßig verwenden als schreibgeschützte Aurora-Replicas konfigurierte Aurora MySQL-DB-Instances die Isolierungsstufe `REPEATABLE READ`. Diese DB-Instances ignorieren alle `SET TRANSACTION ISOLATION LEVEL`-Anweisungen und verwenden weiterhin die Isolierungsstufe `REPEATABLE READ`.

Isolierungsstufe READ COMMITTED für Reader-Instances

Wenn Ihre Anwendung schreibintensive Workloads auf der primären Instance und lang andauernde Abfragen auf den Aurora-Replicas beinhaltet, kann es zu erheblichen Bereinigungsverzögerungen kommen. Eine Bereinigungsverzögerung tritt auf, wenn die interne Garbage Collection von lang andauernden Abfragen blockiert wird. Das Symptom, das Sie sehen, ist ein hoher Wert für `history list length` in der Ausgabe des `SHOW ENGINE INNODB STATUS`-Befehls. Sie können diesen Wert anhand der `RollbackSegmentHistoryListLength`-Metrik in CloudWatch überwachen. Eine erhebliche Bereinigungsverzögerung kann die Effektivität sekundärer Indizes reduzieren und zu einer geringeren allgemeinen Abfrageleistung führen sowie Speicherplatz verschwenden.

Wenn diese Probleme bei Ihnen auftreten, können Sie mit einer Aurora MySQL-Konfigurationseinstellung auf Sitzungsebene, `aurora_read_replica_read_committed`, die Isolierungsstufe `READ COMMITTED` für Aurora-Replicas verwenden. Die Verwendung dieser Einstellung kann Verlangsamungen sowie die Verschwendung von Speicherplatz durch lang andauernde Abfragen reduzieren, während Transaktionen Ihre Tabellen modifizieren.

Wie empfehlen, dass Sie sich mit dem spezifischen Aurora MySQL-Verhalten der `READ COMMITTED`-Isolierung gut vertraut machen, bevor Sie diese Einstellung verwenden. Das Aurora-Replica `READ COMMITTED`-Verhalten entspricht dem ANSI SQL-Standard. Die Isolierung ist jedoch weniger strikt als bei dem typischen MySQL `READ COMMITTED`-Verhalten, mit dem Sie möglicherweise vertraut sind. So kann es etwa zu abweichenden Ergebnissen einer Abfrage unter `READ COMMITTED` auf einem Aurora MySQL-Lesereplikat und für dieselbe Abfrage unter `READ COMMITTED` auf der primären Aurora MySQL-Instance oder auf RDS für MySQL kommen. Sie können die

`aurora_read_replica_read_committed`-Einstellung für Anwendungsfälle wie einen umfassenden Bericht über eine sehr große Datenbank verwenden. Für kurze Abfragen mit kleinen Ergebnissätzen, bei denen es auf Präzision und Wiederholbarkeit ankommt, sollten Sie sie eher nicht verwenden.

Die `READ COMMITTED`-Isolierungsstufe ist nicht für Sitzungen innerhalb eines sekundären Clusters in einer globalen Aurora-Datenbank verfügbar, die die Schreibweiterleitungsfunktion verwenden. Informationen zur Schreibweiterleitung finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

Verwenden von READ COMMITTED für Reader

Um die Isolierungsstufe `READ COMMITTED` für Aurora-Replicas zu aktivieren, setzen Sie die Konfigurationseinstellung `aurora_read_replica_read_committed` auf `ON`. Verwenden Sie diese Einstellung auf Sitzungsebene bei Verbindung zu einer spezifischen Aurora-Replica. Führen Sie dazu die folgenden SQL-Befehle aus:

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

Sie können diese Konfigurationseinstellung temporär verwenden, um interaktive einmalige Abfragen durchzuführen. Sie können auch eine Berichts- oder Datenanalyseanwendung ausführen, die von der Isolierungsstufe `READ COMMITTED` profitiert, wobei die Standards für andere Anwendungen unverändert bleiben.

Wenn die `aurora_read_replica_read_committed`-Einstellung aktiviert ist, geben Sie mit dem `SET TRANSACTION ISOLATION LEVEL`-Befehl die Isolierungsstufe für die jeweiligen Transaktionen an.

```
set transaction isolation level read committed;
```

Unterschiede beim Verhalten von READ COMMITTED auf Aurora-Replikaten

Die `aurora_read_replica_read_committed`-Einstellung stellt die Isolierungsstufe `READ COMMITTED` für eine Aurora-Replica zur Verfügung, mit einem Konsistenzverhalten, das für lang andauernde Transaktionen optimiert ist. Die Isolierungsstufe `READ COMMITTED` auf Aurora-Replicas bietet eine weniger strikte Isolierung als auf primären Aurora-Instances. Aktivieren Sie daher diese Einstellung nur auf Aurora-Replicas, wenn Sie wissen, dass für Ihre Abfragen bestimmte Arten inkonsistenter Ergebnisse akzeptabel sind.

Bei Ihren Abfragen können bestimmte Arten von Anomalien auftreten, wenn die `aurora_read_replica_read_committed`-Einstellung aktiviert ist. Dabei sollten Sie besonders zwei Arten von Anomalien verstehen und in Ihrem Anwendungscode berücksichtigen. Eine nicht-wiederholbarer Lesevorgang tritt auf, wenn eine andere Transaktion ein Commit durchführt, während Ihre Abfrage läuft. Eine lang andauernde Abfrage kann an ihrem Anfang andere Daten sehen als an ihrem Ende. Eine Phantom-Lesevorgang tritt auf, wenn andere Transaktionen dazu führen, dass bestehende Zeilen umorganisiert werden, während Ihre Abfrage läuft, und dass dadurch eine oder mehrere Zeilen von Ihrer Abfrage zweimal gelesen werden.

Phantom-Lesevorgänge können in Ihren Abfragen zu inkonsistenten Zeilenanzahlen führen. Nicht-wiederholbare Lesevorgänge können auch zur Ausgabe unvollständiger oder inkonsistenter Ergebnisse führen. Nehmen Sie beispielsweise an, eine Join-Operation bezieht sich auf Tabellen, die gleichzeitig von SQL-Anweisungen wie `INSERT` oder `DELETE` modifiziert werden. In diesem Fall kann es sein, dass die Join-Abfrage eine Zeile aus einer Tabelle liest, aber nicht die entsprechende Zeile aus einer anderen Tabelle.

Der ANSI SQL-Standard lässt beide Verhaltensweisen für die Isolationsstufe `READ COMMITTED` zu. Diese Verhaltensweisen unterscheiden sich jedoch von der typischen MySQL-Implementierung von `READ COMMITTED`. Prüfen Sie daher vor der Aktivierung der `aurora_read_replica_read_committed`-Einstellung bestehenden SQL-Code darauf, ob er im Rahmen des lockereren Konsistenzmodell so funktioniert wie erwartet.

Zeilenanzahlen und andere Ergebnisse sind auf der Isolationsstufe `READ COMMITTED` bei Aktivierung dieser Einstellung möglicherweise nicht streng konsistent. Sie sollten diese Einstellung daher typischerweise nur aktivieren, wenn Sie analytische Abfragen ausführen, die große Datenmengen aggregieren und für die keine absolute Präzision erforderlich ist. Wenn Sie nicht solche lang andauernden Abfragen zusammen mit schreibintensiven Workloads verwenden, benötigen Sie die `aurora_read_replica_read_committed`-Einstellung wahrscheinlich nicht. Ohne die Kombination aus lang andauernden Abfragen und schreibintensiven Workloads sind Probleme mit der Länge der Verlaufsliste unwahrscheinlich.

Example Abfragen mit Isolationsverhalten für READ COMMITTED auf Aurora-Replicas

Das folgende Beispiel zeigt, wie `READ COMMITTED`-Abfragen auf einer Aurora-Replica möglicherweise nicht wiederholbare Ergebnisse ausgeben, wenn Transaktionen gleichzeitig die zugehörigen Tabellen modifizieren. Die Tabelle `BIG_TABLE` enthält eine Million Zeilen vor Beginn jeder Anfrage. Andere DML- (Data Manipulation Language) Anweisungen fügen Zeilen hinzu, entfernen oder ändern sie.

Die Abfragen auf der primären Aurora-Instance bei Isolierungsstufe `READ COMMITTED` führen zu vorhersehbaren Ergebnissen. Die Overheadlast, die damit verbunden ist, den Consistent-Lesevorgang für die gesamte Dauer einer lang andauernden Abfrage aufrechtzuerhalten, kann später zum umfangreichen Garbage Collection führen.

Die Abfragen auf der Aurora-Replica bei Isolierungsstufe `READ COMMITTED` sind so optimiert, dass dieses Garbage-Collection-Overhead minimiert wird. Dies führt aber andererseits dazu, dass die Ergebnisse abweichen können, wenn die Abfragen auf Zeilen stoßen, die von Transaktionen, die während der Ausführung der Abfrage Commits durchführen, hinzugefügt, entfernt oder umorganisiert werden. Die Abfragen können diese Zeilen berücksichtigen, müssen dies aber nicht tun. Zu Demonstrationszwecken prüfen die Abfragen nur die Anzahl der Zeilen in der Tabelle mithilfe der `COUNT(*)`-Funktion.

Zeit	DML-Anweisung auf der primären Aurora-Instance	Abfrage auf der primären Aurora-Instance mit READ COMMITTED	Abfrage auf einer Aurora-Replica mit READ COMMITTED
T1	`INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;`		
T2		Q1: `SELECT COUNT(*) FROM big_table;`	Q2: `SELECT COUNT(*) FROM big_table;`
T3	`INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;`		
T4		Wenn Q1 jetzt beendet wird, ist das Ergebnis 1 000 000.	Wenn Q2 jetzt beendet wird, ist das Ergebnis 1 000 000 oder 1 000.001.

Zeit	DML-Anweisung auf der primären Aurora-Instance	Abfrage auf der primären Aurora-Instance mit READ COMMITTED	Abfrage auf einer Aurora-Replica mit READ COMMITTED
T5	`DELETE FROM big_table LIMIT 2; COMMIT;`		
T6		Wenn Q1 jetzt beendet wird, ist das Ergebnis 1 000 000.	Wenn Q2 jetzt beendet wird, ist das Ergebnis 1 000 000, 1 000 001, 999 999 oder 999 998.
T7	`UPDATE big_table SET c2 = CONCAT(c2 ,c2,c2); COMMIT;`		
T8		Wenn Q1 jetzt beendet wird, ist das Ergebnis 1 000 000.	Wenn Q2 jetzt beendet wird, ist das Ergebnis 1 000 000, 1 000 001, 999 999 oder möglicherweise eine höhere Zahl.
T9		Q3: `SELECT COUNT(*) FROM big_table;`	Q4: `SELECT COUNT(*) FROM big_table;`
T10		Wenn Q3 jetzt beendet wird, ist das Ergebnis 999.999.	Wenn Q4 jetzt beendet wird, ist das Ergebnis 999.999.

Zeit	DML-Anweisung auf der primären Aurora-Instance	Abfrage auf der primären Aurora-Instance mit READ COMMITTED	Abfrage auf einer Aurora-Replica mit READ COMMITTED
T11		Q5: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`	Q6: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`
T12	`INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;`		
T13		Wenn Q5 jetzt beendet wird, ist das Ergebnis 0.	Wenn Q6 jetzt beendet wird, ist das Ergebnis 0 oder 1.

Wenn die Abfragen schnell abgeschlossen sind, bevor andere Transaktionen DML-Anweisungen und Commits durchführen, sind die Ergebnisse vorhersehbar und zwischen der primären Instance und der Aurora-Replica identisch. Sehen wir uns die Verhaltensunterschiede im Detail an, beginnend mit der ersten Abfrage.

Die Ergebnisse für Q1 sind äußerst vorhersehbar, weil `READ COMMITTED` auf der primären Instance ein starkes Konsistenzmodell ähnlich der Isolierungsstufe `REPEATABLE READ` verwendet.

Die Ergebnisse für Q2 können je nachdem, welche Transaktionen während der Ausführung der Abfrage Commits durchführen, abweichen. Nehmen wir beispielsweise an, dass andere

Transaktionen DML-Anweisungen und Commits durchführen, während die Abfragen laufen. In diesem Fall kann es sein, dass die Abfrage auf der Aurora-Replica mit Isolierungsstufe `READ COMMITTED` die Änderungen berücksichtigt oder nicht. Die Zeilenanzahlen sind dabei nicht in der gleichen Weise vorhersagbar wie bei Isolierungsstufe `REPEATABLE READ`. Sie sind auch nicht so vorhersagbar wie Abfragen mit Isolierungsstufe `READ COMMITTED` auf der primären Instance oder auf einer RDS für MySQL-Instance.

Die `UPDATE`-Anweisung bei T7 ändert die Anzahl der Zeilen in der Tabelle nicht. Durch die Änderung der Länge einer Spalte mit variabler Länge kann diese Anweisung jedoch dazu führen, dass Zeilen intern umorganisiert werden. Eine lang andauernde `READ COMMITTED`-Transaktion kann zunächst die alte Version einer Zeile und später im Rahmen derselben Transaktion eine neue Version derselben Zeile sehen. Die Abfrage kann auch die alte und die neue Version der Zeile überspringen, so dass die Anzahl der Zeilen von der erwarteten abweicht.

Die Ergebnisse von Q5 und Q6 können identisch oder leicht abweichend sein. Abfrage Q6 auf der Aurora-Replica unter `READ COMMITTED` kann, muss aber nicht, die neuen Zeilen sehen, für die während der Dauer der Abfrage ein Commit durchgeführt wird. Möglicherweise sieht sie auch die Zeile aus einer, aber nicht aus der anderen Tabelle. Wenn die Join-Abfrage keine übereinstimmende Zeile in beiden Tabellen findet, gibt sie die Anzahl 0 zurück. Wenn die Abfrage beide neuen Zeilen in `PARENT_TABLE` und `CHILD_TABLE` findet, gibt sie die Anzahl 1 zurück. In einer lang andauernden Anfrage können die Lookups aus den verbundenen Tabellen zu weit auseinanderliegenden Zeitpunkten stattfinden.

> **Note**
>
> Diese Verhaltensunterschiede hängen davon ab, wann für Transaktionen Commits durchgeführt werden, und wann die Abfragen die zugrunde liegenden Tabellenzeilen abfragen. Daher sind solche Unterschiede am wahrscheinlichsten in Berichtsabfragen, die mehrere Minuten oder Stunden dauern, und die auf Aurora-Clustern durchgeführt werden, die gleichzeitig OLTP-Transaktionen ausführen. Dies sind die Arten gemischter Workloads, die am meisten von der Isolierungsstufe `READ COMMITTED` auf Aurora-Replicas profitieren.

Aurora-MySQL-Hinweise

Sie können SQL-Hinweise mit Aurora-MySQL-Abfragen verwenden, um die Leistung zu optimieren. Sie können auch Hinweise verwenden, um zu verhindern, dass Ausführungspläne für wichtige Abfragen aufgrund unvorhersehbarer Bedingungen geändert werden.

> **Tip**
>
> Um zu überprüfen, welche Auswirkungen ein Hinweis auf eine Abfrage hat, überprüfen Sie den von der `EXPLAIN`-Anweisung erzeugten Abfrageplan. Vergleichen Sie die Abfragepläne mit und ohne Hinweis.

In Aurora MySQL Version 3 können Sie alle Hinweise verwenden, die in der MySQL Community Edition 8.0 verfügbar sind. Weitere Informationen zu diesen Hinweisen finden Sie unter Optimierungshinweise im MySQL-Referenzhandbuch.

Die folgenden Hinweise sind in Aurora MySQL Version 2 verfügbar. Diese Hinweise gelten für Abfragen, bei denen die Hash-Join-Funktion in Aurora MySQL Version 2 verwendet wird, insbesondere Abfragen, bei denen die parallele Abfrageoptimierung verwendet wird.

PQ, NO_PQ

Gibt an, ob der Optimierer gezwungen werden soll, Parallelabfragen pro Tabelle oder pro Abfrage zu verwenden.

`PQ` zwingt den Optimierer, Parallelabfragen für bestimmte Tabellen oder die gesamte Abfrage (Block) zu verwenden. `NO_PQ` verhindert, dass der Optimierer Parallelabfragen für bestimmte Tabellen oder die gesamte Abfrage (Block) verwendet.

Dieser Hinweis ist in Aurora MySQL 2.11 und höher verfügbar. In den folgenden Beispielen wird gezeigt, wie Sie diesen Hinweis verwenden können.

> **Note**
>
> Die Angabe eines Tabellennamens zwingt den Optimierer, den `PQ/NO_PQ`-Hinweis nur auf diese ausgewählten Tabellen anzuwenden. Wenn kein Tabellenname angegeben ist, wird der `PQ/NO_PQ`-Hinweis für alle vom Abfrageblock betroffenen Tabellen erzwungen.

```
EXPLAIN SELECT /*+ PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;
```

```
EXPLAIN SELECT /*+ PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;

EXPLAIN SELECT /*+ NO_PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;
```

HASH_JOIN, NO_HASH_JOIN

Aktiviert oder deaktiviert die Funktion des Parallelabfragenoptimierers, auszuwählen, ob die Hash-Join-Optimierungsmethode für eine Abfrage verwendet werden soll. `HASH_JOIN` ermöglicht es dem Optimierer, Hash-Join zu verwenden, wenn dieser Mechanismus effizienter ist. `NO_HASH_JOIN` verhindert, dass der Optimierer Hash-Join für die Abfrage verwendet. Dieser Hinweis ist in Aurora MySQL 2.08 und höher verfügbar. Er hat keine Auswirkungen in Aurora-MySQL-Version 3.

In den folgenden Beispielen wird gezeigt, wie Sie diesen Hinweis verwenden können.

```
EXPLAIN SELECT/*+ HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_PROBING, NO_HASH_JOIN_PROBING

Gibt in einer Hash-Join-Abfrage an, ob die angegebene Tabelle für die Testseite des Joins verwendet werden soll. Statt den gesamten Inhalt der Testtabelle zu lesen, wird über die Abfrage geprüft, ob Spaltenwerte aus der Build-Tabelle in der Testtabelle vorhanden sind. Mit `HASH_JOIN_PROBING` und `HASH_JOIN_BUILDING` können Sie angeben, wie Hash-Join-Abfragen verarbeitet werden, ohne die Tabellen innerhalb des Abfragetextes neu zu ordnen. Dieser Hinweis ist in Aurora MySQL 2.08 und höher verfügbar. Er hat keine Auswirkungen in Aurora-MySQL-Version 3.

In den folgenden Beispielen wird gezeigt, wie Sie diesen Hinweis verwenden können. Die Angabe des HASH_JOIN_PROBING-Hinweises für die Tabelle T2 hat den gleichen Effekt wie die Angabe von NO_HASH_JOIN_PROBING für die Tabelle T1.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_BUILDING, NO_HASH_JOIN_BUILDING

Gibt in einer Hash-Join-Abfrage an, ob die angegebene Tabelle für die Build-Seite des Joins verwendet werden soll oder nicht. Mit der Abfrage werden alle Zeilen aus dieser Tabelle verarbeitet, um die Liste der Spaltenwerte und den Querverweis auf die andere Tabelle zu erstellen. Mit HASH_JOIN_PROBING und HASH_JOIN_BUILDING können Sie angeben, wie Hash-Join-Abfragen verarbeitet werden, ohne die Tabellen innerhalb des Abfragetextes neu zu ordnen. Dieser Hinweis ist in Aurora MySQL 2.08 und höher verfügbar. Er hat keine Auswirkungen in Aurora-MySQL-Version 3.

Im folgenden Beispiel wird gezeigt, wie Sie diesen Hinweis verwenden können. Die Angabe des HASH_JOIN_BUILDING-Hinweises für die Tabelle T2 hat den gleichen Effekt wie die Angabe von NO_HASH_JOIN_BUILDING für die Tabelle T1.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

Gibt an, dass Tabellen in der Abfrage basierend auf der Reihenfolge, in der sie in der Abfrage aufgelistet sind, verknüpft werden. Dieser Hinweis ist besonders nützlich bei Abfragen mit drei oder mehr Tabellen. Er ist als Ersatz für den MySQL–H9inweis gedacht und entspricht dem MySQL-Hinweis JOIN_FIXED_ORDER. Dieser Hinweis ist in Aurora MySQL 2.08 und höher verfügbar.

Im folgenden Beispiel wird gezeigt, wie Sie diesen Hinweis verwenden können.

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER() */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

Gibt die Join-Reihenfolge für die Tabellen in der Abfrage an. Dieser Hinweis ist besonders nützlich bei Abfragen mit drei oder mehr Tabellen. Er entspricht dem MySQL-JOIN_ORDER-Hinweis. Dieser Hinweis ist in Aurora MySQL 2.08 und höher verfügbar.

Im folgenden Beispiel wird gezeigt, wie Sie diesen Hinweis verwenden können.

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

Gibt die Tabellen an, die in der Join-Reihenfolge an erster Stelle stehen sollen. Dieser Hinweis ist besonders nützlich bei Abfragen mit drei oder mehr Tabellen. Er entspricht dem MySQL-JOIN_PREFIX-Hinweis. Dieser Hinweis ist in Aurora MySQL 2.08 und höher verfügbar.

Im folgenden Beispiel wird gezeigt, wie Sie diesen Hinweis verwenden können.

```
EXPLAIN SELECT /*+ JOIN_PREFIX (t4, t2) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

Gibt die Tabellen an, die in der Join-Reihenfolge an letzter Stelle stehen sollen. Dieser Hinweis ist besonders nützlich bei Abfragen mit drei oder mehr Tabellen. Er entspricht dem MySQL-JOIN_SUFFIX-Hinweis. Dieser Hinweis ist in Aurora MySQL 2.08 und höher verfügbar.

Im folgenden Beispiel wird gezeigt, wie Sie diesen Hinweis verwenden können.

```
EXPLAIN SELECT /*+ JOIN_SUFFIX (t1) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

Informationen zur Verwendung von Hash-Join-Abfragen finden Sie unter Optimierung von großen Aurora-MySQL-Join-Abfragen mit Hash-Joins.

Referenz für gespeicherte Aurora-MySQL-Prozeduren

Sie können Ihren Aurora MySQL-DB-Clusters durch Aufrufen integrierter gespeicherter Verfahren verwalten.

Themen

- Erfassen und Verwalten des globalen Statusverlaufs
- Konfigurieren, Starten und Beenden der Binärprotokollreplikation (binlog)
- Beenden einer Sitzung oder Abfrage
- Replizieren von Transaktionen mithilfe von GTIDs
- Rotieren der Abfrageprotokolle
- Festlegen und Anzeigen der Konfiguration des Binärprotokolls

Erfassen und Verwalten des globalen Statusverlaufs

Amazon RDS stellt einen Satz von Verfahren bereit, die Snapshots der Werte dieser Statusvariablen im Zeitverlauf erstellen und diese zusammen mit allen Änderungen seit dem letzten Snapshot in eine Tabelle schreiben. Diese Infrastruktur wird als Global Status History bezeichnet. Weitere Informationen finden Sie unter Verwalten des globalen Statusverlaufs.

Die folgenden gespeicherten Prozeduren verwalten, wie die Global Status History erfasst und verwaltet wird.

Themen

- mysql.rds_collect_global_status_history
- mysql.rds_disable_gsh_collector
- mysql.rds_disable_gsh_rotation
- mysql.rds_enable_gsh_collector
- mysql.rds_enable_gsh_rotation
- mysql.rds_rotate_global_status_history
- mysql.rds_set_gsh_collector
- mysql.rds_set_gsh_rotation

mysql.rds_collect_global_status_history

Generiert einen Snapshot auf Anforderung für den globalen Statusverlauf (Global Status History, GoSH).

Syntax

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

Deaktiviert die periodische Generierung von Snapshots des globalen Statusverlaufs (Global Status History, GoSH).

Syntax

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

Schaltet die Rotation der `mysql.global_status_history`-Tabelle aus.

Syntax

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

Aktiviert den globalen Statusverlauf (Global Status History, GoSH), um Standard-Snapshots in zeitlichen Abständen, die mithilfe von `rds_set_gsh_collector` festgelegt wurden, zu generieren.

Syntax

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

Aktiviert die Rotation der Inhalte der Tabelle `mysql.global_status_history` zu `mysql.global_status_history_old` in zeitlichen Abständen, die durch `rds_set_gsh_rotation` angegeben werden.

Syntax

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

Rotiert die Inhalte der Tabelle `mysql.global_status_history` bei Anforderung zu `mysql.global_status_history_old`.

Syntax

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

Gibt den zeitlichen Abstand für die Generierung von aufeinander folgenden Snapshots durch den globalen Statusverlauf (Global Status History, GoSH) an.

Syntax

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

Parameter

intervalPeriod

Der zeitliche Abstand in Minuten für die periodische Generierung von Snapshots. Der Standardwert ist 5.

mysql.rds_set_gsh_rotation

Gibt den zeitlichen Abstand in Tagen für die periodische Rotation der Tabelle `mysql.global_status_history` an.

Syntax

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

Parameter

intervalPeriod

Der zeitliche Abstand in Tagen für die periodische Tabellenrotation. Der Standardwert ist 7.

Konfigurieren, Starten und Beenden der Binärprotokollreplikation (binlog)

Sie können die folgenden gespeicherten Prozeduren aufrufen, während Sie mit der primären Instance in einem Aurora MySQL-Cluster verbunden sind. Diese Verfahren steuern, wie Transaktionen aus einer externen Datenbank in Aurora MySQL oder aus Aurora MySQL in einer externen Datenbank repliziert werden.

Themen

- mysql.rds_disable_session_binlog (Aurora-MySQL-Version 2)
- mysql.rds_enable_session_binlog (Aurora-MySQL-Version 2)
- mysql.rds_import_binlog_ssl_material
- mysql.rds_next_master_log (Aurora-MySQL-Version 2)
- mysql.rds_next_source_log (Aurora-MySQL-Version 3)
- mysql.rds_remove_binlog_ssl_material
- mysql.rds_reset_external_master (Aurora-MySQL-Version 2)
- mysql.rds_reset_external_source (Aurora-MySQL-Version 3)
- mysql.rds_set_binlog_source_ssl (Aurora-MySQL-Version 3)
- mysql.rds_set_external_master (Aurora-MySQL-Version 2)
- mysql.rds_set_external_source (Aurora-MySQL-Version 3)
- mysql.rds_set_external_master_with_auto_position (Aurora-MySQL-Version 2)
- mysql.rds_set_external_source_with_auto_position (Aurora MySQL Version 3)
- mysql.rds_set_master_auto_position (Aurora-MySQL-Version 2)
- mysql.rds_set_read_only (Aurora-MySQL-Version 3)
- mysql.rds_set_session_binlog_format (Aurora-MySQL-Version 2)
- mysql.rds_set_source_auto_position (Aurora-MySQL-Version 3)
- mysql.rds_skip_repl_error
- mysql.rds_start_replication
- mysql.rds_start_replication_until (Aurora-MySQL-Version 3)
- mysql.rds_stop_replication

mysql.rds_disable_session_binlog (Aurora-MySQL-Version 2)

Deaktiviert die binäre Protokollierung für die aktuelle Sitzung, indem die Variable `sql_log_bin` auf `OFF` festgelegt wird.

Syntax

```
CALL mysql.rds_disable_session_binlog;
```

Parameter

Keine

Nutzungshinweise

Sie rufen dieses gespeicherte Verfahren für einen Aurora MySQL-DB-Cluster auf, während Sie mit der primären Instance verbunden sind.

Für Aurora wird dieses Verfahren für Aurora-MySQL-Version 2.12 und höher und MySQL-5.7-kompatible Versionen unterstützt.

> **Note**
>
> In Aurora-MySQL-Version 3 können Sie den folgenden Befehl verwenden, um die Binärprotokollierung für die aktuelle Sitzung zu deaktivieren, sofern Sie über die entsprechenden `SESSION_VARIABLES_ADMIN`-Berechtigungen verfügen:
>
> ```
> SET SESSION sql_log_bin = OFF;
> ```

mysql.rds_enable_session_binlog (Aurora-MySQL-Version 2)

Aktiviert die binäre Protokollierung für die aktuelle Sitzung, indem die Variable `sql_log_bin` auf `ON` festgelegt wird.

Syntax

```
CALL mysql.rds_enable_session_binlog;
```

Parameter

Keine

Nutzungshinweise

Sie rufen dieses gespeicherte Verfahren für einen Aurora MySQL-DB-Cluster auf, während Sie mit der primären Instance verbunden sind.

Für Aurora wird dieses Verfahren für Aurora-MySQL-Version 2.12 und höher und MySQL-5.7-kompatible Versionen unterstützt.

> **Note**
>
> In Aurora-MySQL-Version 3 können Sie den folgenden Befehl verwenden, um die Binärprotokollierung für die aktuelle Sitzung zu aktivieren, sofern Sie über die entsprechenden `SESSION_VARIABLES_ADMIN`-Berechtigungen verfügen:
>
> ```
> SET SESSION sql_log_bin = ON;
> ```

mysql.rds_import_binlog_ssl_material

Importiert das Zertifizierungsstellenzertifikat, das Clientzertifikat und den Clientschlüssel in eine/einen Aurora-MySQL-DB-Cluster. Die Informationen werden für die SSL-Kommunikation und die verschlüsselte Replikation benötigt.

> **Note**
>
> Derzeit wird dieses Verfahren für folgende Aurora-MySQL 2-Versionen unterstützt: 2.09.2, 2.10.0, 2.10.1 und 2.11.0; sowie Version 3: 3.01.1 und höher.

Syntax

```
CALL mysql.rds_import_binlog_ssl_material (
  ssl_material
);
```

Parameter

ssl_material

JSON-Nutzlast mit dem Inhalt der folgenden PEM-Dateien für einen MySQL-Client:

- „ssl_ca“:““ *Certificate authority certificate*
- „ssl_cert“:““ *Client certificate*
- „ssl_key“:““ *Client key*

Nutzungshinweise

Bereiten Sie die verschlüsselte Replikation vor, bevor Sie diese Schritte durchführen:

- Wenn SSL auf dem externen Server mit der MySQL-Quelldatenbankinstance nicht aktiviert ist und Sie keinen Clientschlüssel und kein Clientzertifikat vorbereitet haben, aktivieren Sie SSL auf dem MySQL-Datenbankserver und generieren Sie den Clientschlüssel und das Clientzertifikat.
- Wenn SSL auf der externen Quelldatenbankinstance aktiviert ist, geben Sie einen Clientschlüssel und ein Clientzertifikat für das Aurora MySQL-DB-Cluster an. Wenn Sie diese Werte nicht haben, erstellen Sie ein einen neuen Schlüssel und ein neues Zertifikat für das Aurora MySQL-DB-Cluster. Sie benötigen zur Signierung des Clientzertifikats den Zertifizierungsstellenschlüssel, den Sie zum Konfigurieren von SSL auf der externen MySQL-Quelldatenbankinstance verwendet haben.

Weitere Informationen finden Sie unter Creating SSL Certificates and Keys Using openssl in der MySQL-Dokumentation.

> **Important**
>
> Führen Sie nach dem Vorbereiten der verschlüsselten Replikation die folgenden Schritte über eine SSL-Verbindung durch. Der Clientschlüssel darf nicht über eine unsichere Verbindung übertragen werden.

Bei diesem Vorgang werden SSL-Informationen aus einer externen MySQL-Datenbank in ein Aurora MySQL-DB-Cluster importiert. Die SSL-Informationen für das Aurora MySQL-DB-Cluster befinden sich in PEM-Dateien. Während der verschlüsselten Replikation dient das Aurora MySQL-DB-Cluster als Client für den MySQL-Datenbankserver. Die Zertifikate und Schlüssel für den Aurora MySQL-Client befinden sich in Dateien im PEM-Format.

Sie können die Informationen aus diesen Dateien in den Parameter `ssl_material` in der richtigen JSON-Nutzlast kopieren. Um die verschlüsselte Replikation zu unterstützen, importieren Sie diese SSL-Informationen in das Aurora MySQL-DB-Cluster.

Die JSON-Nutzlast muss das folgende Format aufweisen.

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
ssl_ca_pem_body_code
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
ssl_cert_pem_body_code
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
ssl_key_pem_body_code
-----END RSA PRIVATE KEY-----\n"}'
```

Beispiele

Im folgenden Beispiel werden SSL-Informationen in eine Aurora MySQL importiert. Der Code in den PEM-Dateien ist in der Regel länger als in diesem Beispiel.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

mysql.rds_next_master_log (Aurora-MySQL-Version 2)

Ändert die Protokollposition der Quelldatenbankinstance in den Anfang des nächsten Binärprotokolls auf der Quelldatenbankinstance. Verwenden Sie dieses Verfahren nur, wenn Sie bei einer Read Replica den I/O Replikationsfehler 1236 erhalten.

Syntax

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

Parameter

curr_master_log

Der Index der aktuellen Master-Protokolldatei. Der Index ist im Dateinamen codiert. Eine aktuelle Datei mit dem Namen `mysql-bin-changelog.012345` hat beispielsweise den Index 12345. Um den Namen der aktuellen Master-Protokolldatei zu ermitteln, führen Sie den Befehl `SHOW REPLICA STATUS` aus. Sie finden den Namen anschließend im Feld `Master_Log_File`.

Nutzungshinweise

Die Prozedur `mysql.rds_next_master_log` muss vom Hauptbenutzer ausgeführt werden.

> **Warning**
>
> Rufen Sie `mysql.rds_next_master_log` nur auf, wenn die Replikation nach einem Failover einer Multi-AZ-DB-Instance, die die Replikationsquelle ist, fehlschlägt und das `Last_IO_Errno` Feld den Fehler 1236 meldet. `SHOW REPLICA STATUS` I/O
> Ein Aufruf von `mysql.rds_next_master_log` kann zu Datenverlust im Lesereplikat führen, falls Transaktionen in der Quell-Instance nicht in das binäre Protokoll auf der Festplatte geschrieben wurden, bevor das Failover-Ereignis auftrat.

Beispiele

Angenommen, die Replikation schlägt auf einer - Aurora-MySQL-Read Replica fehl. Die Ausführung von `SHOW REPLICA STATUS\G` für das Lesereplikat gibt das folgende Ergebnis zurück:

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
```

```
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Den Angeben im Feld `Last_IO_Errno` ist zu entnehmen, dass die Instance eine I/O-Fehlermeldung mit der Nummer 1236 erhalten hat. Dem Feld `Master_Log_File` ist zudem zu entnehmen, dass die betroffene Protokolldatei den Namen `mysql-bin-changelog.012345` aufweist und ihr Index folglich `12345` lautet. Zur Behebung des Fehlers können Sie dann `mysql.rds_next_master_log` mit dem folgenden Parameter aufrufen:

```
CALL mysql.rds_next_master_log(12345);
```

mysql.rds_next_source_log (Aurora-MySQL-Version 3)

Ändert die Protokollposition der Quelldatenbankinstance in den Anfang des nächsten Binärprotokolls auf der Quelldatenbankinstance. Verwenden Sie dieses Verfahren nur, wenn Sie bei einer Read Replica den I/O Replikationsfehler 1236 erhalten.

Syntax

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

Parameter

curr_source_log

Der Index der aktuellen Quell-Protokolldatei. Der Index ist im Dateinamen codiert. Eine aktuelle Datei mit dem Namen `mysql-bin-changelog.012345` hat beispielsweise den Index 12345. Um den Namen der aktuellen Quell-Protokolldatei zu ermitteln, führen Sie den Befehl `SHOW REPLICA STATUS` aus. Sie finden den Namen anschließend im Feld `Source_Log_File`.

Nutzungshinweise

Der Administrator muss das `mysql.rds_next_source_log`-Verfahren ausführen.

> **Warning**
>
> Rufen Sie `mysql.rds_next_source_log` nur auf, wenn die Replikation nach einem Failover einer Multi-AZ-DB-Instance, die die Replikationsquelle ist, fehlschlägt und das `Last_IO_Errno` Feld den Fehler 1236 meldet. `SHOW REPLICA STATUS` I/O
> Ein Aufruf von `mysql.rds_next_source_log` kann zu Datenverlust im Lesereplikat führen, falls Transaktionen in der Quell-Instance nicht in das binäre Protokoll auf der Festplatte geschrieben wurden, bevor das Failover-Ereignis auftrat. Sie können durch Festlegung der Quell-Instance-Parameter `sync_binlog` und `innodb_support_xa` auf 1 das Risiko dafür verringern, obwohl dies zu einer Reduzierung der Leistung führen kann.

Beispiele

Angenommen, die Replikation schlägt auf einer - Aurora-MySQL-Read Replica fehl. Die Ausführung von `SHOW REPLICA STATUS\G` für das Lesereplikat gibt das folgende Ergebnis zurück:

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
```

```
               Until_Condition: None
                Until_Log_File:
                 Until_Log_Pos: 0
            Source_SSL_Allowed: No
            Source_SSL_CA_File:
            Source_SSL_CA_Path:
               Source_SSL_Cert:
             Source_SSL_Cipher:
                Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Den Angeben im Feld `Last_IO_Errno` ist zu entnehmen, dass die Instance eine I/O-Fehlermeldung mit der Nummer 1236 erhalten hat. Dem Feld `Source_Log_File` ist zudem zu entnehmen, dass die betroffene Protokolldatei den Namen `mysql-bin-changelog.012345` aufweist und ihr Index folglich `12345` lautet. Zur Behebung des Fehlers können Sie dann `mysql.rds_next_source_log` mit dem folgenden Parameter aufrufen:

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_remove_binlog_ssl_material

Entfernt das Zertifizierungsstellenzertifikat, das Clientzertifikat und den Clientschlüssel für SSL-Kommunikation und verschlüsselte Replikation. Diese Informationen werden mit importier mysql.rds_import_binlog_ssl_material.

Syntax

```
CALL mysql.rds_remove_binlog_ssl_material;
```

mysql.rds_reset_external_master (Aurora-MySQL-Version 2)

Rekonfiguriert eine Aurora-MySQL-DB-Instance, sodass sie keine Read Replica einer Instance von MySQL außerhalb von Amazon RDS ist.

> **Important**
>
> Um diese Prozedur auszuführen, muss `autocommit` aktiviert sein. Um dies zu aktivieren, setzen Sie den `autocommit`-Parameter auf 1. Weitere Informationen zum Ändern von Parametern finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.

Syntax

```
CALL mysql.rds_reset_external_master;
```

Nutzungshinweise

Die Prozedur `mysql.rds_reset_external_master` muss vom Hauptbenutzer ausgeführt werden. Diese Prozedur muss auf der MySQL-DB-Instance ausgeführt werden, die nicht mehr Lesereplikat einer außerhalb von Amazon RDS ausgeführten MySQL-Instance sein soll.

> **Note**
>
> Wir bieten diese gespeicherten Prozeduren hauptsächlich an, um die Replikation mit MySQL-Instances zu ermöglichen, die außerhalb von Amazon RDS ausgeführt werden. Wir empfehlen die Verwendung von Aurora Replicas, um die Replikation innerhalb eines DB-Clusters von Aurora MySQL zu verwalten, wenn dies möglich ist. Informationen zur Verwaltung der Replikation in DB-Clustern von Aurora MySQL finden Sie unter Verwendung von Aurora-Replicas.

Weitere Informationen zur Verwendung der Replikation für den Import von Daten aus einer außerhalb von Aurora MySQL ausgeführten Instance finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

mysql.rds_reset_external_source (Aurora-MySQL-Version 3)

Rekonfiguriert eine Aurora-MySQL-DB-Instance, sodass sie keine Read Replica einer Instance von MySQL außerhalb von Amazon RDS ist.

> **Important**
>
> Um diese Prozedur auszuführen, muss `autocommit` aktiviert sein. Um dies zu aktivieren, setzen Sie den `autocommit`-Parameter auf 1. Weitere Informationen zum Ändern von Parametern finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.

Syntax

```
CALL mysql.rds_reset_external_source;
```

Nutzungshinweise

Der Administrator muss das `mysql.rds_reset_external_source`-Verfahren ausführen. Diese Prozedur muss auf der MySQL-DB-Instance ausgeführt werden, die nicht mehr Lesereplikat einer außerhalb von Amazon RDS ausgeführten MySQL-Instance sein soll.

> **Note**
>
> Wir bieten diese gespeicherten Prozeduren hauptsächlich an, um die Replikation mit MySQL-Instances zu ermöglichen, die außerhalb von Amazon RDS ausgeführt werden. Wir empfehlen die Verwendung von Aurora Replicas, um die Replikation innerhalb eines DB-Clusters von Aurora MySQL zu verwalten, wenn dies möglich ist. Informationen zur Verwaltung der Replikation in DB-Clustern von Aurora MySQL finden Sie unter Verwendung von Aurora-Replicas.

mysql.rds_set_binlog_source_ssl (Aurora-MySQL-Version 3)

Aktiviert die `SOURCE_SSL`-Verschlüsselung für die Binlog-Replikation. Weitere Informationen finden Sie unter CHANGE REPLICATION SOURCE TO-Anweisung in der MySQL-Dokumentation.

Syntax

```
CALL mysql.rds_set_binlog_source_ssl(mode);
```

Parameter

mode

Ein Wert, der angibt, ob die SOURCE_SSL-Verschlüsselung aktiviert ist:

- 0 – Die SOURCE_SSL-Verschlüsselung ist deaktiviert. Der Standardwert ist 0.
- 1 – Die SOURCE_SSL-Verschlüsselung ist aktiviert. Sie können die Verschlüsselung mit SSL oder TLS konfigurieren.

Nutzungshinweise

Diese Prozedur wird für Aurora-MySQL-Version 3.06 und höher unterstützt.

mysql.rds_set_external_master (Aurora-MySQL-Version 2)

Konfiguriert eine Aurora-MySQL-Instance für die Verwendung als Read Replica einer außerhalb von Amazon RDS ausgeführten MySQL-Instance.

Das `mysql.rds_set_external_master`-Verfahren ist veraltet und wird in einer künftigen Version entfernt. Verwenden Sie stattdessen `mysql.rds_set_external_source`.

> **Important**
>
> Um diese Prozedur auszuführen, muss `autocommit` aktiviert sein. Um dies zu aktivieren, setzen Sie den `autocommit`-Parameter auf 1. Weitere Informationen zum Ändern von Parametern finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.

Syntax

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
```

```
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameter

host_name

Der Hostname bzw. die IP-Adresse der außerhalb von Amazon RDS ausgeführten MySQL-Instance, die als Quelldatenbank-Instance festgelegt werden soll.

host_port

Der Port, der von der außerhalb von Amazon RDS ausgeführten MySQL-Instance verwendet wird, die als Quelldatenbank-Instance konfiguriert werden soll. Wenn Ihre Netzwerkkonfiguration die Replikation über Secure Shell (SSH)-Ports einschließt, welche die Portnummer konvertiert, geben Sie für diesen Parameter die von SSH offengelegte Portnummer an.

replication_user_name

Die ID eines Benutzers mit den Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` auf der MySQL-Instance, die extern zu Amazon RDS ausgeführt wird. Es wird empfohlen, ein Benutzerkonto bereitzustellen, das ausschließlich für die Replikation mit der externen Instance genutzt wird.

replication_user_password

Das zu dem in `replication_user_name` angegebenen User-ID gehörige Passwort.

mysql_binary_log_file_name

Der Name des Binärprotokolls auf der Quelldatenbank-Instance, die die Replikationsinformationen enthält.

mysql_binary_log_file_location

Die Position in der binären Protokolldatei `mysql_binary_log_file_name`, ab der bei der Replikation die Replikationsinformationen gelesen werden.

Sie können den Namen und den Speicherort der Binlog-Datei ermitteln, indem Sie `SHOW MASTER STATUS`auf der Quelldatenbank-Instance starten.

ssl_encryption

Ein Wert, der angibt, ob die SSL-Verschlüsselung (Secure Socket Layer) für die Replikationsverbindung verwendet wird. 1 = SSL-Verschlüsselung, 0 = keine Verschlüsselung. Der Standardwert ist 0.

> **Note**
>
> Die Option `MASTER_SSL_VERIFY_SERVER_CERT` wird nicht unterstützt. Diese Option ist auf 0 gesetzt, was bedeutet, dass die Verbindung verschlüsselt ist, aber die Zertifikate nicht überprüft werden.

Nutzungshinweise

Die Prozedur `mysql.rds_set_external_master` muss vom Hauptbenutzer ausgeführt werden. Diese Prozedur muss auf der MySQL-DB-Instance ausgeführt werden, die als Lesereplikat einer außerhalb von Amazon RDS ausgeführten MySQL-Instance konfiguriert werden soll.

Vor der Ausführung von `mysql.rds_set_external_master` müssen Sie zuerst die außerhalb von Amazon RDS ausgeführte MySQL-Instance für die Verwendung als Quelldatenbank-Instance konfigurieren. Um eine Verbindung zu der außerhalb von Amazon RDS ausgeführten MySQL-Instance herzustellen, müssen Sie Werte für `replication_user_name` und `replication_user_password` bereitstellen, die auf einen Replikationsbenutzer verweisen, der über die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` für die externe MySQL-Instance verfügt.

So konfigurieren Sie eine externe Instance von MySQL als Quelldatenbank-Instance

1. Verbinden Sie sich mithilfe eines MySQL-Clients Ihrer Wahl mit der externen MySQL-Instance und erstellen Sie ein Benutzerkonto, das für die Replikation verwendet werden soll. Im Folgenden wird ein -Beispiel gezeigt.

   MySQL 5.7

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   MySQL 8.0

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
 'password';
```

> **Note**
>
> Geben Sie aus Sicherheitsgründen ein anderes Passwort als hier angegeben an.

2. Erteilen Sie innerhalb der externen MySQL-Instance Ihrem Replikationsbenutzer die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE`. Im folgenden Beispiel werden dem Benutzer 'repl_user' für Ihre Domäne die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` für alle Datenbanken erteilt.

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

Um die verschlüsselte Replikation zu verwenden, konfigurieren Sie die Quelldatenbank-Instance für die Verwendung von SSL-Verbindungen. Importieren Sie außerdem mit der Prozedur mysql.rds_import_binlog_ssl_material das Zertifizierungsstellenzertifikat, das Clientzertifikat und den Clientschlüssel in die DB-Instance bzw. das DB-Cluster.

> **Note**
>
> Wir bieten diese gespeicherten Prozeduren hauptsächlich an, um die Replikation mit MySQL-Instances zu ermöglichen, die außerhalb von Amazon RDS ausgeführt werden. Wir empfehlen die Verwendung von Aurora Replicas, um die Replikation innerhalb eines DB-Clusters von Aurora MySQL zu verwalten, wenn dies möglich ist. Informationen zur Verwaltung der Replikation in DB-Clustern von Aurora MySQL finden Sie unter Verwendung von Aurora-Replicas.

Nachdem Sie `mysql.rds_set_external_master` aufgerufen haben, um eine Amazon-RDS-DB-Instance als Lesereplikat zu konfigurieren, können Sie mysql.rds_start_replication für das Lesereplikat aufrufen, um die Replikation zu starten Zudem haben Sie die Möglichkeit, mit einem Aufruf von mysql.rds_reset_external_master (Aurora-MySQL-Version 2) die Lesereplikat-Konfiguration zu entfernen.

Beim Aufrufen von `mysql.rds_set_external_master` werden von Amazon RDS Uhrzeit, Benutzer und eine Aktion von `set master` in den Tabellen `mysql.rds_history` und `mysql.rds_replication_status` protokolliert.

Beispiele

Bei Ausführung innerhalb einer MySQL-DB-Instance konfiguriert das folgende Beispiel diese DB-Instance für die Verwendung als Lesereplikat einer außerhalb von Amazon RDS ausgeführten MySQL-Instance.

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_source (Aurora-MySQL-Version 3)

Konfiguriert eine Aurora-MySQL-Instance für die Verwendung als Read Replica einer außerhalb von Amazon RDS ausgeführten MySQL-Instance.

> **Important**
>
> Um diese Prozedur auszuführen, muss `autocommit` aktiviert sein. Um dies zu aktivieren, setzen Sie den `autocommit`-Parameter auf 1. Weitere Informationen zum Ändern von Parametern finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.

Syntax

```
CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameter

host_name

Der Hostname bzw. die IP-Adresse der außerhalb von Amazon RDS ausgeführten MySQL-Instance, die als Quelldatenbank-Instance festgelegt werden soll.

host_port

Der Port, der von der außerhalb von Amazon RDS ausgeführten MySQL-Instance verwendet wird, die als Quelldatenbank-Instance konfiguriert werden soll. Wenn Ihre Netzwerkkonfiguration die Replikation über Secure Shell (SSH)-Ports einschließt, welche die Portnummer konvertiert, geben Sie für diesen Parameter die von SSH offengelegte Portnummer an.

replication_user_name

Die ID eines Benutzers mit den Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` auf der MySQL-Instance, die extern zu Amazon RDS ausgeführt wird. Es wird empfohlen, ein Benutzerkonto bereitzustellen, das ausschließlich für die Replikation mit der externen Instance genutzt wird.

replication_user_password

Das zu dem in `replication_user_name` angegebenen User-ID gehörige Passwort.

mysql_binary_log_file_name

Der Name des Binärprotokolls auf der Quelldatenbank-Instance, die die Replikationsinformationen enthält.

mysql_binary_log_file_location

Die Position in der binären Protokolldatei `mysql_binary_log_file_name`, ab der bei der Replikation die Replikationsinformationen gelesen werden.

Sie können den Namen und den Speicherort der Binlog-Datei ermitteln, indem Sie `SHOW MASTER STATUS`auf der Quelldatenbank-Instance starten.

`ssl_encryption`

Ein Wert, der angibt, ob die SSL-Verschlüsselung (Secure Socket Layer) für die Replikationsverbindung verwendet wird. 1 = SSL-Verschlüsselung, 0 = keine Verschlüsselung. Der Standardwert ist 0.

> **Note**
>
> Sie müssen ein benutzerdefiniertes SSL-Zertifikat mit mysql.rds_import_binlog_ssl_material importiert haben, um diese Option zu aktivieren. Wenn Sie kein benutzerdefiniertes SSL-Zertifikat importiert haben, legen Sie diesen Parameter auf 0 fest und verwenden Sie mysql.rds_set_binlog_source_ssl (Aurora-MySQL-Version 3), um SSL für die binäre Protokollreplikation zu aktivieren.
> Die Option `SOURCE_SSL_VERIFY_SERVER_CERT` wird nicht unterstützt. Diese Option ist auf 0 gesetzt, was bedeutet, dass die Verbindung verschlüsselt ist, aber die Zertifikate nicht überprüft werden.

Nutzungshinweise

Der Administrator muss das `mysql.rds_set_external_source`-Verfahren ausführen. Dieses Verfahren muss auf der DB-Instance von Aurora MySQL ausgeführt werden, die als Lesereplikat einer außerhalb von Amazon RDS ausgeführten MySQL-Instance konfiguriert werden soll.

Vor der Ausführung von `mysql.rds_set_external_source` müssen Sie zuerst die außerhalb von Amazon RDS ausgeführte MySQL-Instance für die Verwendung als Quelldatenbank-Instance konfigurieren. Um eine Verbindung zu der außerhalb von Amazon RDS ausgeführten MySQL-Instance herzustellen, müssen Sie Werte für `replication_user_name` und `replication_user_password` bereitstellen, die auf einen Replikationsbenutzer verweisen, der über die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` für die externe MySQL-Instance verfügt.

So konfigurieren Sie eine externe Instance von MySQL als Quelldatenbank-Instance

1. Verbinden Sie sich mithilfe eines MySQL-Clients Ihrer Wahl mit der externen MySQL-Instance und erstellen Sie ein Benutzerkonto, das für die Replikation verwendet werden soll. Im Folgenden wird ein -Beispiel gezeigt.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

> **Note**
>
> Geben Sie aus Sicherheitsgründen ein anderes Passwort als hier angegeben an.

2. Erteilen Sie innerhalb der externen MySQL-Instance Ihrem Replikationsbenutzer die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE`. Im folgenden Beispiel werden dem Benutzer 'repl_user' für Ihre Domäne die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` für alle Datenbanken erteilt.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

Um die verschlüsselte Replikation zu verwenden, konfigurieren Sie die Quelldatenbank-Instance für die Verwendung von SSL-Verbindungen. Importieren Sie außerdem mit der Prozedur mysql.rds_import_binlog_ssl_material das Zertifizierungsstellenzertifikat, das Clientzertifikat und den Clientschlüssel in die DB-Instance bzw. den DB-Cluster.

> **Note**
>
> Wir bieten diese gespeicherten Prozeduren hauptsächlich an, um die Replikation mit MySQL-Instances zu ermöglichen, die außerhalb von Amazon RDS ausgeführt werden. Wir empfehlen die Verwendung von Aurora Replicas, um die Replikation innerhalb eines DB-Clusters von Aurora MySQL zu verwalten, wenn dies möglich ist. Informationen zur Verwaltung der Replikation in DB-Clustern von Aurora MySQL finden Sie unter Verwendung von Aurora-Replicas.

Nachdem Sie `mysql.rds_set_external_source` aufgerufen haben, um eine DB-Instance von Aurora MySQL als Lesereplikat zu konfigurieren, können Sie mysql.rds_start_replication für das Lesereplikat aufrufen, um die Replikation zu starten Zudem haben Sie die Möglichkeit, mit einem Aufruf von mysql.rds_reset_external_source (Aurora-MySQL-Version 3) die Lesereplikat-Konfiguration zu entfernen.

Beim Aufrufen von `mysql.rds_set_external_source` werden von Amazon RDS Uhrzeit, Benutzer und eine Aktion von `set master` in den Tabellen `mysql.rds_history` und `mysql.rds_replication_status` protokolliert.

Beispiele

Bei Ausführung auf einer DB-Instance von Aurora MySQL konfiguriert das folgende Beispiel diese DB-Instance für die Verwendung als Lesereplikat einer außerhalb von Amazon RDS ausgeführten MySQL-Instance.

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_master_with_auto_position (Aurora-MySQL-Version 2)

Konfiguriert eine primäre Instance von Aurora MySQL, um eine eingehende Replikation von einer externen MySQL-Instance zu akzeptieren. Bei diesem Verfahren wird auch die Replikation auf der Grundlage globaler Transaktions-Identifikatoren () konfiguriert. GTIDs

Dieses Verfahren konfiguriert nicht die verzögerte Replikation, da Aurora MySQL die verzögerte Replikation nicht unterstützt.

Syntax

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

Parameter

host_name

Der Hostname bzw. die IP-Adresse der außerhalb von Aurora ausgeführten MySQL-Instance, die als Replikationsquelle festgelegt werden soll.

host_port

Der Port, der von der außerhalb von Aurora ausgeführten MySQL-Instance verwendet wird, die als Replikations-Quelle konfiguriert werden soll. Wenn Ihre Netzwerkkonfiguration die Replikation über Secure Shell (SSH)-Ports einschließt, welche die Portnummer konvertiert, geben Sie für diesen Parameter die von SSH offengelegte Portnummer an.

replication_user_name

Die ID eines Benutzers mit den Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` auf der MySQL-Instance, die extern zu Aurora ausgeführt wird. Es wird empfohlen, ein Benutzerkonto bereitzustellen, das ausschließlich für die Replikation mit der externen Instance genutzt wird.

replication_user_password

Das zu dem in `replication_user_name` angegebenen User-ID gehörige Passwort.

ssl_encryption

Diese Option ist derzeit nicht implementiert. Der Standardwert ist 0.

Nutzungshinweise

Sie rufen dieses gespeicherte Verfahren für einen Aurora MySQL-DB-Cluster auf, während Sie mit der primären Instance verbunden sind.

Die Prozedur `mysql.rds_set_external_master_with_auto_position` muss vom Hauptbenutzer ausgeführt werden. Der Master-Benutzer führt diese Verfahren auf der primären Instance eines Aurora MySQL-DB-Clusters durch, der als Replikationsziel fungiert. Dies kann das Replikationsziel einer externen MySQL-DB-Instance oder eines Aurora MySQL-DB-Clusters sein.

Diese Prozedur wird für Aurora-MySQL-Version 2 unterstützt. Verwenden Sie für Aurora-MySQL-Version 3 stattdessen das Verfahren mysql.rds_set_external_source_with_auto_position (Aurora MySQL Version 3).

Konfigurieren Sie die externe MySQL-DB-Instance vor der Ausführung von `mysql.rds_set_external_master_with_auto_position` als Replikationsquelle. Um sich mit der externen MySQL-Instance zu verbinden, geben Sie Werte für `replication_user_name` und `replication_user_password` an. Diese Werte müssen einen Replikationsbenutzer mit den Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` auf der externen MySQL-Instance angeben.

So konfigurieren Sie eine externe MySQL-Instance als Replikationsquelle

1. Verbinden Sie sich mithilfe eines MySQL-Clients Ihrer Wahl mit der externen MySQL-Instance und erstellen Sie ein Benutzerkonto, das für die Replikation verwendet werden soll. Im Folgenden wird ein Beispiel gezeigt.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. Erteilen Sie Ihrem Replikationsbenutzer auf der externen MySQL-Instance die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE`. Im folgenden Beispiel werden dem Benutzer `REPLICATION CLIENT` für Ihre Domäne die Berechtigungen `REPLICATION SLAVE` und `'repl_user'` für alle Datenbanken erteilt.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

Wenn Sie `mysql.rds_set_external_master_with_auto_position` aufrufen, zeichnet Amazon RDS bestimmte Informationen auf. Diese Informationen umfassen die Zeit, den Benutzer und eine Aktion von `"set master"` in den Tabellen `mysql.rds_history` und `mysql.rds_replication_status`.

Wenn Sie eine bestimmte GTID-basierte Transaktion überspringen möchten, von der Sie wissen, dass sie ein Problem verursacht, können Sie die gespeicherte Prozedur mysql.rds_skip_transaction_with_gtid (Aurora-MySQL-Version 2 und 3) verwenden. Weitere Informationen über das Arbeiten mit der GTID-basierten Replikation finden Sie unter Verwenden der GTID-basierten Replikation.

Beispiele

Wenn die folgende Beispielkonfiguration auf einer primären Instance von Aurora ausgeführt wird, wird der Aurora-Cluster so konfiguriert, dass er als Lesereplikat einer Instance von MySQL dient, die extern von Aurora ausgeführt wird.

```
call mysql.rds_set_external_master_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_external_source_with_auto_position (Aurora MySQL Version 3)

Konfiguriert eine primäre Instance von Aurora MySQL, um eine eingehende Replikation von einer externen MySQL-Instance zu akzeptieren. Bei diesem Verfahren wird auch die Replikation auf der Grundlage globaler Transaktions-Identifikatoren () konfiguriert. GTIDs

Syntax

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

Parameter

host_name

Der Hostname bzw. die IP-Adresse der außerhalb von Aurora ausgeführten MySQL-Instance, die als Replikationsquelle festgelegt werden soll.

host_port

Der Port, der von der außerhalb von Aurora ausgeführten MySQL-Instance verwendet wird, die als Replikations-Quelle konfiguriert werden soll. Wenn Ihre Netzwerkkonfiguration die Replikation über Secure Shell (SSH)-Ports einschließt, welche die Portnummer konvertiert, geben Sie für diesen Parameter die von SSH offengelegte Portnummer an.

replication_user_name

Die ID eines Benutzers mit den Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` auf der MySQL-Instance, die extern zu Aurora ausgeführt wird. Es wird empfohlen, ein Benutzerkonto bereitzustellen, das ausschließlich für die Replikation mit der externen Instance genutzt wird.

replication_user_password

Das zu dem in `replication_user_name` angegebenen User-ID gehörige Passwort.

ssl_encryption

Diese Option ist derzeit nicht implementiert. Der Standardwert ist 0.

> **Note**
>
> Verwenden Sie mysql.rds_set_binlog_source_ssl (Aurora-MySQL-Version 3), um SSL für die Binärprotokollreplikation zu aktivieren.

Nutzungshinweise

Sie rufen dieses gespeicherte Verfahren für einen Aurora MySQL-DB-Cluster auf, während Sie mit der primären Instance verbunden sind.

Der Administrator muss das `mysql.rds_set_external_source_with_auto_position`-Verfahren ausführen. Der Administrator-Benutzer führt diese Verfahren auf der primären Instance eines Aurora MySQL-DB-Clusters durch, der als Replikationsziel fungiert. Dies kann das Replikationsziel einer externen MySQL-DB-Instance oder eines Aurora MySQL-DB-Clusters sein.

Zurzeit wird diese Prozedur für Aurora MySQL Version 3 unterstützt. Dieses Verfahren konfiguriert nicht die verzögerte Replikation, da Aurora MySQL die verzögerte Replikation nicht unterstützt.

Konfigurieren Sie die externe MySQL-DB-Instance vor der Ausführung von `mysql.rds_set_external_source_with_auto_position` als Replikationsquelle. Um sich mit der externen MySQL-Instance zu verbinden, geben Sie Werte für `replication_user_name` und `replication_user_password` an. Diese Werte müssen einen Replikationsbenutzer mit den Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE` auf der externen MySQL-Instance angeben.

So konfigurieren Sie eine externe MySQL-Instance als Replikationsquelle

1. Verbinden Sie sich mithilfe eines MySQL-Clients Ihrer Wahl mit der externen MySQL-Instance und erstellen Sie ein Benutzerkonto, das für die Replikation verwendet werden soll. Im Folgenden wird ein Beispiel gezeigt.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. Erteilen Sie Ihrem Replikationsbenutzer auf der externen MySQL-Instance die Berechtigungen `REPLICATION CLIENT` und `REPLICATION SLAVE`. Im folgenden Beispiel werden dem Benutzer `REPLICATION CLIENT` für Ihre Domäne die Berechtigungen `REPLICATION SLAVE` und `'repl_user'` für alle Datenbanken erteilt.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

Wenn Sie `mysql.rds_set_external_source_with_auto_position` aufrufen, zeichnet Amazon RDS bestimmte Informationen auf. Diese Informationen umfassen die Zeit, den Benutzer und eine Aktion von `"set master"` in den Tabellen `mysql.rds_history` und `mysql.rds_replication_status`.

Wenn Sie eine bestimmte GTID-basierte Transaktion überspringen möchten, von der Sie wissen, dass sie ein Problem verursacht, können Sie die gespeicherte Prozedur mysql.rds_skip_transaction_with_gtid (Aurora-MySQL-Version 2 und 3) verwenden. Weitere Informationen über das Arbeiten mit der GTID-basierten Replikation finden Sie unter Verwenden der GTID-basierten Replikation.

Beispiele

Wenn die folgende Beispielkonfiguration auf einer primären Instance von Aurora ausgeführt wird, wird der Aurora-Cluster so konfiguriert, dass er als Lesereplikat einer Instance von MySQL dient, die extern von Aurora ausgeführt wird.

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_master_auto_position (Aurora-MySQL-Version 2)

Legt fest, dass der Replikationsmodus entweder auf binären Protokolldateipositionen oder auf globalen Transaktions-Identifikatoren (GTIDs) basiert.

Syntax

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

Parameter

auto_position_mode

Ein Wert, der angibt, ob die Replikation auf Basis der Protokolldateiposition oder der GTID verwendet werden soll:

- `0` – Verwendung der auf der Binärprotokolldateiposition basierenden Replikationsmethode. Der Standardwert ist `0`.
- `1` – Verwendung der auf GTID basierenden Replikationsmethode.

Nutzungshinweise

Die Prozedur `mysql.rds_set_master_auto_position` muss vom Hauptbenutzer ausgeführt werden.

Diese Prozedur wird für Aurora-MySQL-Version 2 unterstützt.

mysql.rds_set_read_only (Aurora-MySQL-Version 3)

Aktiviert oder deaktiviert den Modus `read_only` global für die DB-Instance.

Syntax

```
CALL mysql.rds_set_read_only(mode);
```

Parameter

mode

Ein Wert, der angibt, ob der `read_only`-Modus für die DB-Instance global aktiviert oder deaktiviert ist:

- `0` – `OFF`. Der Standardwert ist `0`.
- `1` – `ON`

Nutzungshinweise

Die gespeicherte Prozedur `mysql.rds_set_read_only` ändert nur den Parameter `read_only`. Der Parameter `innodb_read_only` kann auf Reader-DB-Instances nicht geändert werden.

Die Änderung des Parameters `read_only` wird beim Neustart nicht beibehalten. Um dauerhafte Änderungen an `read_only` vorzunehmen, müssen Sie den DB-Cluster-Parameter `read_only` verwenden.

Diese Prozedur wird für Aurora-MySQL-Version 3.06 und höher unterstützt.

mysql.rds_set_session_binlog_format (Aurora-MySQL-Version 2)

Legt das binäre Protokollformat für die aktuelle Sitzung fest.

Syntax

```
CALL mysql.rds_set_session_binlog_format(format);
```

Parameter

format

Ein Wert, der das binäre Protokollformat für die aktuelle Sitzung angibt:

- `STATEMENT` – Die Replikationsquelle schreibt Ereignisse auf der Grundlage von SQL-Anweisungen in das Binärprotokoll.
- `ROW` – Die Replikationsquelle schreibt Ereignisse in das Binärprotokoll, die auf Änderungen an einzelnen Tabellenzeilen hinweisen.
- `MIXED` – Die Protokollierung basiert in der Regel auf SQL-Anweisungen, wechselt jedoch unter bestimmten Bedingungen zu Zeilen. Weitere Informationen finden Sie unter Mixed Binary Logging Format (Gemischtes Binärprotokollformat) in der MySQL-Dokumentation.

Nutzungshinweise

Sie rufen dieses gespeicherte Verfahren für einen Aurora MySQL-DB-Cluster auf, während Sie mit der primären Instance verbunden sind.

Wenn Sie diese gespeicherte Prozedur verwenden möchten, müssen Sie die Binärprotokollierung für die aktuelle Sitzung konfiguriert haben.

Für Aurora wird dieses Verfahren für Aurora-MySQL-Version 2.12 und höher und MySQL-5.7-kompatible Versionen unterstützt.

mysql.rds_set_source_auto_position (Aurora-MySQL-Version 3)

Legt fest, dass der Replikationsmodus entweder auf binären Protokolldateipositionen oder auf globalen Transaktions-Identifikatoren () GTIDs basiert.

Syntax

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

Parameter

auto_position_mode

Ein Wert, der angibt, ob die Replikation auf Basis der Protokolldateiposition oder der GTID verwendet werden soll:

- `0` – Verwendung der auf der Binärprotokolldateiposition basierenden Replikationsmethode. Der Standardwert ist `0`.
- 1 – Verwendung der auf GTID basierenden Replikationsmethode.

Nutzungshinweise

Sie rufen dieses gespeicherte Verfahren für einen Aurora MySQL-DB-Cluster auf, während Sie mit der primären Instance verbunden sind.

Der Administrator muss das `mysql.rds_set_source_auto_position`-Verfahren ausführen.

mysql.rds_skip_repl_error

Ignoriert und löscht einen Replikationsfehler in einem MySQL-DB-Lesereplikat.

Syntax

```
CALL mysql.rds_skip_repl_error;
```

Nutzungshinweise

Der Hauptbenutzer muss die Prozedur `mysql.rds_skip_repl_error` auf einem Lesereplikat ausführen. Weitere Informationen zu dieser Prozedur finden Sie unter Überspringen des aktuellen Replikationsfehlers.

Führen Sie den MySQL-Befehl `SHOW REPLICA STATUS\G` aus, um festzustellen, ob Fehler aufgetreten sind. Wenn ein Replikationsfehler nicht als kritisch eingestuft, ist, können Sie `mysql.rds_skip_repl_error` ausführen, um den Fehler zu überspringen. Wenn mehrere Fehler aufgetreten sind, löscht `mysql.rds_skip_repl_error` den ersten Fehler und weist darauf hin, dass weitere Fehlermeldungen anhängig sind. In diesem Fall können Sie mithilfe von `SHOW REPLICA STATUS\G` die angemessene Vorgehensweise bei der Handhabung des nächsten Fehlers ermitteln. Informationen zu den zurückgegebenen Werten finden Sie unter SHOW REPLICA STATUS-Anweisung in der MySQL-Dokumentation.

Weitere Informationen zur Handhabung von Replikationsfehlern mit Aurora MySQL finden Sie unter Diagnose und Lösung eines Fehlers bei einer MySQL Read Replica.

Fehler „Replication stopped (Replikation gestoppt)“

Wenn Sie die Prozedur `mysql.rds_skip_repl_error` aufrufen, wird möglicherweise eine Fehlermeldung angezeigt, die besagt, dass das Replikat ausgefallen oder deaktiviert ist.

Diese Fehlermeldung wird angezeigt, wenn Sie die Prozedur auf der primären Instance statt auf dem Lesereplikat ausführen. Sie müssen diese Prozedur auf dem Lesereplikat ausführen, damit sie funktioniert.

Diese Fehlermeldung wird möglicherweise auch angezeigt, wenn Sie die Prozedur zwar auf dem Lesereplikat ausführen, die Replikation jedoch nicht neu gestartet werden kann.

Wenn Sie eine größere Anzahl von Fehlern überspringen müssen, kann die Dauer der Replikationsverzögerung den standardmäßigen Aufbewahrungszeitraum für binäre Protokolldateien (binlog) überschreiten. In diesem Fall kann es zu einem schwerwiegenden Fehler kommen, weil Binärprotokolldateien bereinigt werden, bevor ihr Inhalt in das Lesereplikat repliziert wurde. Diese Bereinigung führt zur Beendigung der Replikation, und Sie können den Befehl

`mysql.rds_skip_repl_error` nicht mehr aufrufen, um Replikationsfehler zu überspringen und zu ignorieren.

Sie können dieses Problem verringern, indem Sie die Anzahl der Stunden erhöhen, die die Binärprotokolldateien auf Ihrer Quelldatenbankinstance aufbewahrt werden. Nachdem Sie die Aufbewahrungsdauer für binäre Protokolldateien verlängert haben, können Sie die Replikation neu starten und nach Bedarf den Befehl `mysql.rds_skip_repl_error` aufrufen.

Verwenden Sie zur Festlegung der Aufbewahrungszeit für Binärprotokolldateien die Prozedur mysql.rds_set_configuration und legen Sie einen Konfigurationsparameter für `'binlog retention hours'` zusammen mit der Stundenanzahl für die Aufbewahrung der Binärprotokolldateien im DB-Cluster fest. Beim folgenden Beispiel wird die Aufbewahrungszeit für binäre Protokolle auf 48 Stunden festgelegt.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

Startet die Replikation von einer/einem Aurora-MySQL-DB-Cluster.

> **Note**
>
> Sie können die gespeicherte Prozedur mysql.rds_start_replication_until (Aurora-MySQL-Version 3) oder mysql.rds_start_replication_until_gtid (Aurora-MySQL-Version 3) verwenden, um die Replikation von einer Aurora-MySQL-DB-Instance zu initiieren und die Replikation an der angegebenen Position der Binärprotokolldatei zu stoppen.

Syntax

```
CALL mysql.rds_start_replication;
```

Nutzungshinweise

Die Prozedur `mysql.rds_start_replication` muss vom Hauptbenutzer ausgeführt werden.

Um Daten aus einer außerhalb von Amazon RDS ausgeführten MySQL-Instance zu importieren, rufen Sie `mysql.rds_start_replication` für das Lesereplikat auf, um den Replikationsvorgang

zu starten, nachdem Sie mysql.rds_set_external_master (Aurora-MySQL-Version 2) oder mysql.rds_set_external_source (Aurora-MySQL-Version 3) aufgerufen haben, um die Replikation zu konfigurieren. Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Zum Export von Daten in eine außerhalb von Amazon RDS ausgeführte MySQL-Instance rufen Sie `mysql.rds_start_replication` und `mysql.rds_stop_replication` für das Lesereplikat auf, um Replikationsaktionen wie das Bereinigen von Binärprotokollen zu steuern. Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Darüber hinaus können Sie `mysql.rds_start_replication` für das Lesereplikat aufrufen, um einen zuvor durch einen Aufruf von `mysql.rds_stop_replication` gestoppten Replikationsprozess wieder zu starten. Weitere Informationen finden Sie unter Fehler „Replication stopped (Replikation gestoppt)“.

mysql.rds_start_replication_until (Aurora-MySQL-Version 3)

Initiiert die Replikation von einem Aurora-MySQL-DB-Cluster und stoppt die Replikation an der angegebenen Position in der Binärprotokolldatei.

Syntax

```
CALL mysql.rds_start_replication_until (
replication_log_file
  , replication_stop_point
);
```

Parameter

replication_log_file

Der Name des Binärprotokolls auf der Quelldatenbank-Instance, die die Replikationsinformationen enthält.

replication_stop_point

Die Position im `replication_log_file`-Binärprotokoll, an der die Replikation stoppt.

Nutzungshinweise

Die Prozedur `mysql.rds_start_replication_until` muss vom Hauptbenutzer ausgeführt werden.

Diese Prozedur wird für Aurora-MySQL-Version 3.04 und höher unterstützt.

Die gespeicherte Prozedur `mysql.rds_start_replication_until` wird für die verwaltete Replikation nicht unterstützt, darunter Folgendes:

- Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg
- Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat)

Der für den Parameter `replication_log_file` angegebene Dateiname muss mit dem Binlogdateinamen der Quelldatenbankinstance übereinstimmen.

Wenn der Parameter `replication_stop_pointe`ine Stoppposition angibt, die in der Vergangenheit liegt, wird die Replikation sofort gestoppt.

Beispiele

Das folgende Beispiel initiiert die Replikation und repliziert die Änderungen, bis die Position `120` in der Binärprotokolldatei `mysql-bin-changelog.000777` erreicht wird.

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_stop_replication

Stoppt die Replikation von einer MySQL-DB-Instance.

Syntax

```
CALL mysql.rds_stop_replication;
```

Nutzungshinweise

Die Prozedur `mysql.rds_stop_replication` muss vom Hauptbenutzer ausgeführt werden.

Wenn Sie die Replikation für den Import von Daten aus einer außerhalb von Amazon RDS ausgeführten MySQL-Instance konfigurieren, stoppen Sie mit einem Aufruf von `mysql.rds_stop_replication` für das Lesereplikat den Replikationsvorgang nach Abschluss des Imports. Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Wenn Sie die Replikation für den Export von Daten in eine außerhalb von Amazon RDS ausgeführte MySQL-Instance konfigurieren, rufen Sie `mysql.rds_start_replication` und `mysql.rds_stop_replication` für das Lesereplikat auf, um Replikationsaktionen wie das Bereinigen von Binärprotokollen zu steuern. Weitere Informationen finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation).

Die gespeicherte Prozedur `mysql.rds_stop_replication` wird für die verwaltete Replikation nicht unterstützt, darunter Folgendes:

- Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg
- Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat)

Beenden einer Sitzung oder Abfrage

Die folgenden gespeicherten Prozeduren beenden eine Sitzung oder Abfrage.

Themen

- mysql.rds_kill
- mysql.rds_kill_query

mysql.rds_kill

Beendet eine Verbindung zum MySQL-Server.

Syntax

```
CALL mysql.rds_kill(processID);
```

Parameter

processID

Die ID des Verbindungs-Threads, der beendet werden soll.

Nutzungshinweise

Jede Verbindung zum MySQL-Server wird in einem eigenen Thread ausgeführt. Um eine Verbindung zu beenden, verwenden Sie die Prozedur `mysql.rds_kill` und übergeben ihr als Parameter die Thread-ID der Verbindung. Die Thread-ID erhalten Sie mithilfe des MySQL-Befehls SHOW PROCESSLIST.

Beispiele

Im folgenden Beispiel wird eine Verbindung mit der Thread-ID 4243 beendet:

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

Beendet eine an den MySQL-Server übermittelte Abfrage.

Syntax

```
CALL mysql.rds_kill_query(processID);
```

Parameter

processID

Die Identität des Prozesses oder Threads, der die zu beendende Abfrage ausführt.

Nutzungshinweise

Um eine an den MySQL-Server übermittelte Abfrage zu beenden, verwenden Sie die Prozedur `mysql_rds_kill_query` und übergeben die ID des Threads, der die Abfrage ausführt. Die Prozedur beendet dann die Verbindung.

Die Abfrage-ID erhalten Sie mithilfe der MySQL-Tabelle INFORMATION_SCHEMA PROCESSLIST oder des MySQL-Befehls SHOW PROCESSLIST. Der Wert in der ID-Spalte von `SHOW PROCESSLIST` oder `SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST` ist die *processID*.

Beispiele

Im folgenden Beispiel wird eine Abfrage mit der Thread-ID 230040 beendet:

```
CALL mysql.rds_kill_query(230040);
```

Replizieren von Transaktionen mithilfe von GTIDs

Die folgenden gespeicherten Prozeduren steuern, wie Transaktionen mithilfe globaler Transaktionskennungen (GTIDs) mit RDS für MySQL repliziert werden. Weitere Informationen zur Verwendung der Replikation basierend auf GTIDs mit Aurora MySQL finden Sie unter Verwenden der GTID-basierten Replikation.

Themen

- mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL Version 3)
- mysql.rds_gtid_purged (Aurora-MySQL-Version 3)
- mysql.rds_skip_transaction_with_gtid (Aurora-MySQL-Version 2 und 3)
- mysql.rds_start_replication_until_gtid (Aurora-MySQL-Version 3)

mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL Version 3)

Konfiguriert die `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`-Option der `CHANGE REPLICATION SOURCE TO`-Anweisung. Dadurch weist der Replikationskanal replizierten Transaktionen, die keine haben, eine GTID zu. Auf diese Weise können Sie die Binärprotokollreplikation von einer Quelle aus, die keine GTID-basierte Replikation verwendet, zu einem Replikat, durchführen, das dies tut. Weitere Informationen finden Sie unter CHANGE REPLICATION SOURCE TO Statement und Replikation von einer Quelle ohne GTIDs zu einer Replik mit GTIDs im MySQL-Referenzhandbuch.

Syntax

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

Parameter

`gtid_option`

Zeichenfolgenwert Die erlaubten Werte sind `OFF`, `LOCAL`, oder eine angegebene UUID.

Nutzungshinweise

Dieses Vorgehen hat die gleiche Wirkung wie das Absetzen der Anweisung `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = gtid_option` in Community MySQL.

GTID muss auf ON gesetzt werden, damit *gtid_option* auf LOCAL oder eine bestimmte UUID gesetzt wird.

Der Standardwert ist OFF, was bedeutet, dass die Funktion nicht verwendet wird.

LOCAL weist eine GTID einschließlich der eigenen UUID des Replikats (der server_uuid-Einstellung) zu.

Das Übergeben eines Parameters, bei dem es sich um eine UUID handelt, weist eine GTID zu, die die angegebene UUID enthält, z. B. die server_uuid-Einstellung für den Replikationsquellserver.

Beispiele

So deaktivieren Sie diese Funktion:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-------------------------------------------------------------+
| Message  |
+-------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-------------------------------------------------------------+
1 row in set (0.07 sec)
```

So verwenden Sie die eigene UUID des Replikats:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+--------------------------------------------------------------+
| Message  |
+--------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+--------------------------------------------------------------+
1 row in set (0.07 sec)
```

So verwenden Sie eine angegebene UUID:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----------------------------------------------------------------------------------------------------------
+
| Message |
+-----------------------------------------------------------------------------------------------------------
+
```

```
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----------------------------------------------------------------------------------------------
+
1 row in set (0.07 sec)
```

mysql.rds_gtid_purged (Aurora-MySQL-Version 3)

Legt den globalen Wert der Systemvariablen `gtid_purged` auf einen bestimmten Satz von globalen Transaktionskennungen (GTID) fest. Die Systemvariable `gtid_purged` ist ein GTID-Satz, der aus den GTIDs aller Transaktionen besteht, für die auf dem Server ein Commit ausgeführt wurde, die jedoch in keiner binären Logdatei auf dem Server vorhanden sind.

Es gibt zwei Möglichkeiten, den Wert `gtid_purged` festzulegen, um Kompatibilität mit MySQL 8.0 zu ermöglichen:

- Ersetzen Sie den Wert `gtid_purged` durch Ihren angegebenen GTID-Set.
- Fügen Sie Ihren angegebenen GTID-Satz an den GTID-Satz an, den `gtid_purged` bereits enthält.

Syntax

So ersetzen Sie den Wert `gtid_purged` durch Ihren angegebenen GTID-Satz:

```
CALL mysql.rds_gtid_purged (gtid_set);
```

So fügen Sie den Wert `gtid_purged` Ihrem angegebenen GTID-Satz an:

```
CALL mysql.rds_gtid_purged (+gtid_set);
```

Parameter

`gtid_set`

Der Wert *`gtid_set`* muss eine Obermenge des aktuellen Werts `gtid_purged` sein und darf sich nicht mit `gtid_subtract(gtid_executed,gtid_purged)` überschneiden. Das heißt, der neue GTID-Satz muss alle GTIDs enthalten, die bereits in `gtid_purged` enthalten waren, und darf keine GTIDs in `gtid_executed` enthalten, die noch nicht gelöscht wurden. Der

Parameter *gtid_set* darf auch keine GTIDs enthalten, die sich im globalen gtid_owned-Satz befinden, die GTIDs für Transaktionen, die gerade auf dem Server verarbeitet werden.

Nutzungshinweise

Die Prozedur mysql.rds_gtid_purged muss vom Hauptbenutzer ausgeführt werden.

Diese Prozedur wird für Aurora-MySQL-Version 3.04 und höher unterstützt.

Beispiele

Im folgenden Beispiel wird die GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 der globalen Variable gtid_purged zugewiesen.

```
CALL mysql.rds_gtid_purged('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_skip_transaction_with_gtid (Aurora-MySQL-Version 2 und 3)

Überspringt die Replikation einer Transaktion mit der angegebenen globalen Transaktionskennung (GTID) auf einer primären Instance von Aurora.

Sie können dieses Verfahren für die Notfallwiederherstellung verwenden, wenn eine bestimmte GTID-Transaktion bekanntermaßen ein Problem verursacht. Verwenden Sie diese gespeicherte Prozedur, um die problematische Transaktion zu überspringen. Problematisch sind beispielsweise Transaktionen, die die Replikation deaktivieren, wichtige Daten löschen oder dafür sorgen, dass die DB-Instance nicht mehr verfügbar ist.

Syntax

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

Parameter

gtid_to_skip

Die GTID der zu überspringenden Replikationstransaktion.

Nutzungshinweise

Die Prozedur `mysql.rds_skip_transaction_with_gtid` muss vom Hauptbenutzer ausgeführt werden.

Diese Prozedur wird für Aurora-MySQL-Version 2 und 3 unterstützt.

Beispiele

Im folgenden Beispiel wird die Replikation der Transaktion mit der GTID übersprunge `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_start_replication_until_gtid (Aurora-MySQL-Version 3)

Initiiert die Replikation von einer/einem Aurora-MySQL-DB-Cluster und stoppt die Replikation unmittelbar nach der angegebenen globalen Transaktionskennung (GTID).

Syntax

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

Parameter

gtid

Die GTID, nach der die Replikation stoppen soll.

Nutzungshinweise

Die Prozedur `mysql.rds_start_replication_until_gtid` muss vom Hauptbenutzer ausgeführt werden.

Diese Prozedur wird für Aurora-MySQL-Version 3.04 und höher unterstützt.

Das gespeicherte `mysql.rds_start_replication_until_gtid`-Verfahren wird für die verwaltete Replikation nicht unterstützt, darunter Folgendes:

- Replizieren von Amazon-Aurora-MySQL-DB-Clustern über AWS-Regionen hinweg

- Migrieren von Daten aus einer RDS-für-MySQL-DB-Instance zu einem Amazon-Aurora-MySQL-DB-Cluster mittels einer Aurora Read Replica (Lesereplikat)

Wenn der Parameter `gtid` eine Transaktion angibt, die bereits von dem Replikat ausgeführt wurde, wird die Replikation sofort gestoppt.

Beispiele

Das folgende Beispiel initiiert die Replikation und repliziert die Änderungen, bis die GTID erreicht wir `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

Rotieren der Abfrageprotokolle

Die folgenden gespeicherten Prozeduren rotieren MySQL-Protokolle in Backup-Tabellen. Weitere Informationen finden Sie unter Aurora-MySQL-Datenbank-Protokolldateien.

Themen

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

mysql.rds_rotate_general_log

Rotiert die Tabelle `mysql.general_log` in eine Sicherungstabelle.

Syntax

```
CALL mysql.rds_rotate_general_log;
```

Nutzungshinweise

Sie können die Tabelle `mysql.general_log` in eine Sicherungstabelle rotieren, indem Sie die Prozedur `mysql.rds_rotate_general_log` aufrufen. Beim Rotieren von Protokolldateien wird die aktuelle Protokolltabelle in eine Sicherungsprotokolltabelle kopiert, und die Einträge in der aktuellen Protokolltabelle werden entfernt. Sofern bereits eine Sicherungsprotokolltabelle vorhanden ist, wird diese gelöscht, bevor die aktuelle Protokolltabelle in die Sicherungsprotokolltabelle kopiert wird. Sie können die Sicherungsprotokolltabelle abfragen, wenn dies nötig ist. Die Backup-Protokolltabelle für die `mysql.general_log`-Tabelle ist als `mysql.general_log_backup` benannt.

Sie können dieses Verfahren nur ausführen, wenn der Parameter `log_output` auf `TABLE` eingestellt ist.

mysql.rds_rotate_slow_log

Rotiert die Tabelle `mysql.slow_log` in eine Sicherungstabelle.

Syntax

```
CALL mysql.rds_rotate_slow_log;
```

Nutzungshinweise

Sie können die Tabelle `mysql.slow_log` in eine Sicherungstabelle rotieren, indem Sie die Prozedur `mysql.rds_rotate_slow_log` aufrufen. Beim Rotieren von Protokolldateien wird die aktuelle Protokolltabelle in eine Sicherungsprotokolltabelle kopiert, und die Einträge in der aktuellen Protokolltabelle werden entfernt. Sofern bereits eine Sicherungsprotokolltabelle vorhanden ist, wird diese gelöscht, bevor die aktuelle Protokolltabelle in die Sicherungsprotokolltabelle kopiert wird.

Sie können die Sicherungsprotokolltabelle abfragen, wenn dies nötig ist. Die Backup-Protokolltabelle für die `mysql.slow_log`-Tabelle ist als `mysql.slow_log_backup` benannt.

Festlegen und Anzeigen der Konfiguration des Binärprotokolls

Die folgenden gespeicherten Prozeduren legen Konfigurationsparameter fest und zeigen sie an, z. B. für die Aufbewahrung binärer Protokolldateien.

Themen

- mysql.rds_set_configuration
- mysql.rds_show_configuration

mysql.rds_set_configuration

Gibt die Anzahl an Stunden an, für die die Binärprotokolle aufbewahrt werden sollen, oder die Anzahl Sekunden, um die die Replikation verzögert werden soll.

Syntax

```
CALL mysql.rds_set_configuration(name,value);
```

Parameter

Name

Der Name des festzulegenden Konfigurationsparameters

Wert

Der Wert des Konfigurationsparameters

Nutzungshinweise

Die Prozedur `mysql.rds_set_configuration` unterstützt die folgenden Konfigurationsparameter:

- binlog retention hours

Die Konfigurationsparameter werden dauerhaft gespeichert und überstehen jeden Neustart oder Failover der DB-Instance.

binlog retention hours

Der Parameter `binlog retention hours` wird verwendet, um die Anzahl der Stunden anzugeben, die Binärprotokolldateien aufbewahrt werden sollen. In der Regel werden binäre Protokolldateien von Amazon Aurora so schnell wie möglich bereinigt. Eine binäre Protokolldatei ist möglicherweise für die Replikation mit einer außerhalb von Aurora ausgeführten MySQL-Datenbank erforderlich.

Der Standardwert von `binlog retention hours` ist `NULL`. Für Aurora MySQL `NULL` bedeutet, dass binäre Protokolle faul aufgeräumt werden. Aurora-MySQL-Binärprotokolle können für einen bestimmten Zeitraum im System verbleiben, normalerweise nicht länger als einen Tag.

Um die Anzahl der Stunden zu bestimmen, für die Binärprotokolle auf einer/einem DB-Cluster aufbewahrt werden sollen, verwenden Sie die gespeicherte Prozedur `mysql.rds_set_configuration` und geben Sie, wie in dem folgenden Beispiel gezeigt, einen ausreichend großen Zeitraum für die gewünschte Replikation an.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> Sie können den Wert `0` nicht für `binlog retention hours` verwenden.

Der maximal zulässige `binlog retention hours`-Wert für DB-Cluster von Aurora MySQL Version 2.11.0 und höher sowie Version 3 ist 2 160 (90 Tage).

Nachdem Sie den Aufbewahrungszeitraum festgelegt haben, überwachen Sie die Speichernutzung für die DB-Instance, um sicherzustellen, dass die aufbewahrten binären Protokolle nicht zu viel Speicherplatz beanspruchen.

mysql.rds_show_configuration

Die Anzahl der Stunden, während der binäre Protokolldateien aufbewahrt werden sollen.

Syntax

```
CALL mysql.rds_show_configuration;
```

Nutzungshinweise

Mit der gespeicherten Prozedur `mysql.rds_show_configuration` überprüfen Sie, wie viele Stunden Amazon RDS die binären Protokolldateien aufbewahrt werden.

Beispiele

Nachfolgend sehen Sie ein Beispiel für die Anzeige des Aufbewahrungszeitraums:

```
call mysql.rds_show_configuration;
                name                          value     description
                binlog retention hours        24        binlog retention hours specifies
 the duration in hours before binary logs are automatically deleted.
```

Aurora-MySQL-spezifische information_schema-Tabellen

Aurora MySQL verfügt über bestimmte `information_schema`-Tabellen, die für Aurora spezifisch sind.

information_schema.aurora_global_db_instance_status

Die Tabelle `information_schema.aurora_global_db_instance_status` enthält Informationen über den Status aller DB-Instances in den primären und sekundären DB-Clustern einer globalen Datenbank. Die Spalten, die Sie verwenden können, sind in der folgenden Tabelle aufgeführt. Die übrigen Spalten sind nur für den internen Gebrauch von Aurora bestimmt.

> **Note**
>
> Diese Informationsschematabelle ist nur mit globalen Aurora-MySQL-Datenbanken ab Version 3.04.0 verfügbar.

Spalte	Datentyp	Beschreibung
SERVER_ID	varchar(100)	Die ID der DB-Instance.
SESSION_ID	varchar(100)	Eindeutiger Bezeichner für die aktuelle Sitzung. Der Wert `MASTER_SESSION_ID`

Spalte	Datentyp	Beschreibung
		bezeichnet die (primäre) Writer-DB-Instance.
AWS_REGION	varchar(100)	Die AWS-Region, in der diese globale DB-Instance ausgeführt wird. Eine Liste der Regionen finden Sie unter Verfügbarkeit in Regionen.
DURABLE_LSN	bigint unsigned	Die Log-Sequenznummer (LSN), die dauerhaft gespeichert wurde. Eine Log-Sequenznummer (LSN) ist eine eindeutige fortlaufende Nummer, die einen Datensatz im Datenbank-Transakt ionsprotokoll identifiziert. LSNs werden so angeordnet, dass eine größere LSN eine spätere Transaktion darstellt.
HIGHEST_LSN_RCVD	bigint unsigned	Die höchste LSN, die die DB-Instance von der Writer-DB-Instance empfangen hat.
OLDEST_READ_VIEW_T RX_ID	bigint unsigned	Die ID der ältesten Transakti on, zu der die Writer-DB-Instance Daten löschen kann.
OLDEST_READ_VIEW_LSN	bigint unsigned	Die älteste LSN, die von der DB-Instance zum Lesen aus dem Speicher verwendet wird.

Spalte	Datentyp	Beschreibung
VISIBILITY_LAG_IN_MSEC	float(10,0) unsigned	Für Reader im primären DB-Cluster, wie weit diese DB-Instance der Writer-DB -Instance in Millisekunden hinterherhinkt. Für Reader in einem sekundären DB-Cluste r, wie weit diese DB-Instance dem sekundären Volume in Millisekunden hinterherhinkt.

information_schema.aurora_global_db_status

Die Tabelle `information_schema.aurora_global_db_status` enthält Informationen über verschiedene Aspekte der globalen Aurora-Datenbankverzögerung an, insbesondere die Verzögerung des zugrunde liegenden Aurora-Speichers (sogenannte Haltbarkeitsverzögerung) und die Verzögerung zwischen dem Recovery Point Objective (RPO). Die Spalten, die Sie verwenden können, sind in der folgenden Tabelle aufgeführt. Die übrigen Spalten sind nur für den internen Gebrauch von Aurora bestimmt.

> **Note**
>
> Diese Informationsschematabelle ist nur mit globalen Aurora-MySQL-Datenbanken ab Version 3.04.0 verfügbar.

Spalte	Datentyp	Beschreibung
AWS_REGION	varchar(100)	Die AWS-Region, in der diese globale DB-Instance ausgeführt wird. Eine Liste der Regionen finden Sie unter Verfügbarkeit in Regionen.
HIGHEST_LSN_WRITTEN	bigint unsigned	Die höchste Log-Seque nznummer (LSN), die derzeit

Spalte	Datentyp	Beschreibung
		auf diesem DB-Cluster vorhanden ist. Eine Log-Sequenznummer (LSN) ist eine eindeutige fortlaufende Nummer, die einen Datensatz im Datenbank-Transakt ionsprotokoll identifiziert. LSNs werden so angeordnet, dass eine größere LSN eine spätere Transaktion darstellt.
DURABILITY_LAG_IN_ MILLISECONDS	float(10,0) unsigned	Die Differenz der Zeitstemp elwerte zwischen `HIGHEST_L SN_WRITTEN` in einem sekundären DB-Cluster und `HIGHEST_LSN_WRITTE N` im primären DB-Cluste r. Der Wert ist immer 0 im primären DB-Cluster der globalen Aurora-Datenbank.

Spalte	Datentyp	Beschreibung
RPO_LAG_IN_MILLISECONDS	float(10,0) unsigned	Die Recovery Point Objective (RPO)-Verzögerung. Die RPO-Verzögerung ist die Zeit, die benötigt wird, bis die letzte Benutzertransaktion COMMIT auf einem sekundären DB-Cluster gespeichert wird, nachdem sie auf dem primären DB-Cluster einer globalen Aurora-Datenbank abgelegt wurde. Der Wert ist immer 0 im primären DB-Cluster der globalen Aurora-Datenbank. Einfach ausgedrückt berechnet diese Metrik das Recovery Point Objective für jeden DB-Cluster voon Aurora MySQL in der globalen Aurora-Datenbank, d. h., wie viele Daten bei einem Ausfall verloren gehen könnten. Wie die Verzögerung wird auch RPO zeitlich gemessen.

Spalte	Datentyp	Beschreibung
LAST_LAG_CALCULATION_TIMESTAMP	datetime	Der Zeitstempel, der angibt, wann die Werte für `DURABILITY_LAG_IN_MILLISECONDS` und `RPO_LAG_IN_MILLISECONDS` zuletzt berechnet wurden. Ein Zeitwert wie `1970-01-01 00:00:00+00` bedeutet, dass dies der primäre DB-Cluster ist.
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	Die ID der ältesten Transaktion, zu der die Writer-DB-Instance Daten löschen kann.

information_schema.replica_host_status

Die Tabelle `information_schema.replica_host_status` enthält Replikationsinformationen. Die Spalten, die Sie verwenden können, sind in der folgenden Tabelle aufgeführt. Die übrigen Spalten sind nur für den internen Gebrauch von Aurora bestimmt.

Spalte	Datentyp	Beschreibung
CPU	double	Die prozentuale CPU-Auslastung des Replikat-Hosts.
IS_CURRENT	tinyint	Gibt an, ob das Replikat aktuell ist.
LAST_UPDATE_TIMESTAMP	datetime(6)	Zeitpunkt, an dem die letzte Aktualisierung stattfand. Wird verwendet, um festzustellen, ob ein Datensatz veraltet ist.

Spalte	Datentyp	Beschreibung
REPLICA_LAG_IN_MILLISECONDS	double	Die Replikatverzögerung in Millisekunden.
SERVER_ID	varchar(100)	Die ID des Datenbankservers.
SESSION_ID	varchar(100)	Die ID der Datenbanksitzung. Wird verwendet, um festzustellen, ob es sich bei einer DB-Instance um eine Writer- oder Reader-Instance handelt.

> **Note**
>
> Wenn eine Replikat-Instance in Verzug gerät, sind die aus ihrer `information_schema.replica_host_status`-Tabelle abgefragten Informationen möglicherweise veraltet. In diesem Fall empfehlen wir, stattdessen eine Abfrage von der Writer-Instance aus durchzuführen.
> Die Tabelle `mysql.ro_replica_status` enthält zwar ähnliche Informationen, wir raten Ihnen jedoch davon ab, diese zu verwenden.

information_schema.aurora_forwarding_processlist

Die Tabelle `information_schema.aurora_forwarding_processlist` enthält Informationen über Prozesse, die an der Schreibweiterleitung beteiligt sind.

Der Inhalt dieser Tabelle ist nur auf der Writer-DB-Instance für einen DB-Cluster sichtbar, bei dem die globale oder clusterinterne Schreibweiterleitung aktiviert ist. Auf Reader-DB-Instances wird ein leerer Ergebnissatz zurückgegeben.

Feld	Datentyp	Beschreibung
ID	bigint	Der Bezeichner der Verbindung auf der Writer-DB-Instance. Dieser Bezeichner ist derselbe Wert, der in der Spalte `Id` der `SHOW PROCESSLIST` -Anweisung angezeigt wird,

Feld	Datentyp	Beschreibung
		und wird von der Funktion `CONNECTION_ID()` innerhalb des Threads zurückgegeben.
USER	varchar(32)	Der MySQL-Benutzer, der die Anweisung erstellt hat.
HOST	varchar(255)	Der MySQL-Client, der die Anweisung erstellt hat. Für weitergeleitete Anweisungen zeigt dieses Feld die Hostadresse des Anwendungsclients an, der die Verbindung auf der weiterleitenden Reader-DB-Instance hergestellt hat.
DB	varchar(64)	Die Standarddatenbank für den Thread.
COMMAND	varchar(16)	Die Art des Befehls, den der Thread im Namen des Clients ausführt, oder `Sleep`, wenn die Sitzung inaktiv ist. Eine Beschreibung der Thread-Befehle finden Sie unter Thread Command Values (Thread-Befehlswerte) in der MySQL-Dokumentation .
TIME	int	Die Zeit in Sekunden, in der sich der Thread in seinem aktuellen Status befand.
STATE	varchar(64)	Eine Aktion, ein Ereignis oder ein Status, der angibt, welche Aktionen der Thread ausführt. Eine Beschreibung der Statuswerte finden Sie unter General Thread Status (Allgemeine Thread-Status) in der MySQL-Dokumentation.
INFO	longtext	Die Anweisung, die der Thread ausführt, oder `NULL`, wenn er keine Anweisung ausführt. Bei der Anweisung kann es sich um die an den Server gesendete oder um eine innerste Anweisung handeln, wenn die Anweisung andere Anweisungen ausführt.
IS_FORWARDED	bigint	Gibt an, ob der Thread von einer Reader-DB-Instance weitergeleitet wird.

Feld	Datentyp	Beschreibung
REPLICA_SESSION_ID	bigint	Die Verbindungs-ID in der Aurora Replica. Dieser Bezeichner ist derselbe Wert, der in der Spalte `Id` der `SHOW PROCESSLIST` -Anweisung auf der weiterleitenden Reader-DB-Instance von Aurora angezeigt wird.
REPLICA_INSTANCE_IDENTIFIER	varchar(64)	Die ID der DB-Instance des weiterleitenden Threads.
REPLICA_CLUSTER_NAME	varchar(64)	Die ID des DB-Clusters des weiterleitenden Threads. Für die In-Cluster-Schreibweiterleitung ist diese ID derselbe DB-Cluster wie die Writer-DB-Instance.
REPLICA_REGION	varchar(64)	Die AWS-Region, aus der der weiterleitende Thread stammt. Für die In-Cluster-Schreibweiterleitung ist diese Region dieselbe AWS-Region wie die Writer-DB-Instance.

Datenbank-Engine-Updates für Amazon Aurora MySQL

Amazon Aurora veröffentlicht regelmäßig Updates. Updates werden auf Aurora-DB-Cluster während des Wartungszeitraums angewendet. Zu welchem Zeitpunkt Updates angewandt werden, hängt von den Einstellungen für die Region und den Wartungszeitraum für das DB-Cluster sowie vom Typ des Updates ab.

Amazon-Aurora-Versionen werden im Laufe mehrerer Tage allen AWS-Regionen zur Verfügung gestellt. Einige Regionen zeigen möglicherweise vorübergehend eine Engine-Version an, die in einer anderen Region noch nicht verfügbar ist.

Updates werden auf alle Instances in einem DB-Cluster gleichzeitig angewendet. Ein Update erfordert einen Neustart einer Datenbank auf allen Instances in einem DB-Cluster. Daher dauert es 20 bis 30 Sekunden, bis Sie wieder mit der Verwendung Ihrer DB-Cluster fortfahren können. Sie können die Einstellungen für den Wartungszeitraum in der anzeigen und ändern AWS-Managementkonsole.

Weitere Informationen zu den Aurora-MySQL-Versionen, die von Amazon Aurora unterstützt werden, finden Sie in den Versionshinweisen für Aurora MySQL.

Im Folgenden erfahren Sie, wie Sie die richtige Aurora MySQL-Version für Ihren Cluster auswählen, wie Sie die Version beim Erstellen oder Aktualisieren eines Clusters angeben, und wie Sie mit minimaler Unterbrechung einen Cluster von einer Version auf eine andere aktualisieren können.

Themen

- Überprüfen von Aurora-MySQL-Versionsnummern
- LTS-Versionen (Long-Term-Support, Langzeit-Support) und Betaversionen für Amazon Aurora MySQL
- Vorbereitung auf das Ende des Standard-Supports für die Amazon-Aurora-MySQL-kompatible Edition der Version 2
- Vorbereitung auf das Lebenszyzklusende der Amazon Aurora MySQL-kompatible Edition Version 1
- Upgrade von Amazon Aurora MySQL-DB-Clustern
- Datenbank-Engine-Updates und -Korrekturen für Amazon Aurora MySQL

Überprüfen von Aurora-MySQL-Versionsnummern

Obwohl Aurora MySQL-kompatible Edition mit den MySQL-Datenbank-Engines kompatibel ist, enthält Aurora MySQL Funktionen und Bugfixes, die spezifisch für bestimmte Aurora MySQL-Versionen sind. Anwendungsentwickler können die Aurora MySQL-Version in ihren Anwendungen mithilfe von SQL überprüfen. Datenbankadministratoren können beim Erstellen oder Aktualisieren von Aurora MySQL-DB-Clustern und DB-Instances Aurora MySQL-Versionen überprüfen und angeben.

Themen

- Prüfen oder Angeben von Aurora-MySQL-Engine-Versionen über AWS
- Überprüfen von Aurora MySQL-Versionen mit SQL

Prüfen oder Angeben von Aurora-MySQL-Engine-Versionen über AWS

Wenn Sie administrative Aufgaben mit der AWS-Managementkonsole-, AWS CLI- oder RDS-API ausführen, geben Sie die Aurora-MySQL-Version in einem alphanumerischen Format an.

Seit Aurora-MySQL-Version 2 wird für Aurora-Engine-Versionen die folgende Syntax verwendet.

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

Der *mysql-major-version*-Anteil ist `5.7` oder `8.0`. Dieser Wert stellt die Version des Clientprotokolls und die allgemeine Ebene der MySQL-Funktionsunterstützung für die entsprechende Aurora MySQL-Version dar.

aurora-mysql-version ist ein gepunkteter Wert mit drei Teilen: der Aurora MySQL-Hauptversion, der Aurora MySQL-Nebenversion und der Patch-Ebene. Die Hauptversion ist `2` oder `3`. Diese Werte stellen Aurora MySQL dar, das mit MySQL 5.7 bzw. 8.0 kompatibel ist. Die Unterversion stellt die Funktionsversion innerhalb der 2.x- oder 3.x-Serie dar. Die Patch-Ebene beginnt bei `0` für jede Nebenversion und stellt die Reihe der nachfolgenden Bugfixes dar, die für die Nebenversion gelten. Gelegentlich wird eine neue Funktion in eine Nebenversion integriert, aber nicht sofort sichtbar gemacht. In diesen Fällen wird die Funktion einer Optimierung unterzogen und in einem späteren Patch-Stufe veröffentlicht.

Alle 2.x-Engine-Versionen von Aurora MySQL sind mit Community MySQL 5.7.12 oder höher drahtkompatibel. Alle 3.x-Engine-Versionen von Aurora MySQL sind mit MySQL 8.0.23 oder höher drahtkompatibel. Sie können auf Versionshinweise einer bestimmten 3.x-Version zurückgreifen, um die entsprechende MySQL-kompatible Version zu finden.

Zum Beispiel sind die Engine-Versionen für Aurora MySQL 3.04.0 und 2.11.2 die folgenden.

```
8.0.mysql_aurora.3.04.0
5.7.mysql_aurora.2.11.2
```

> **Note**
>
> Es gibt keine Eins-zu-Eins-Korrespondenz zwischen den Community-MySQL-Versionen und den Aurora-MySQL-2.x-Versionen. Für Aurora-MySQL-Version 3 gibt es ein direkteres Mapping. Um zu überprüfen, welche Bugfixes und neuen Funktionen in einer bestimmten Aurora-MySQL-Version enthalten sind, lesen Sie Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 3 und Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 2 in den Versionshinweisen für Aurora MySQL. Eine chronologische Liste der neuen Funktionen und Releases finden Sie unter Dokumentverlauf. Informationen zum Überprüfen der Mindestversion, die für eine sicherheitsbezogene Fehlerbehebung erforderlich ist, finden Sie unter In Aurora MySQL behobene Sicherheitsschwachstellen in den Versionshinweisen für Aurora MySQL.

Sie können die Aurora-MySQL-Engine-Version bei einigen AWS CLI-Befehlen und RDS-API-Operationen angeben. Beispielsweise geben Sie die `--engine-version`-Option an, wenn Sie die AWS CLI-Befehle create-db-cluster und modify-db-cluster ausführen. Sie geben den Parameter `EngineVersion` an, wenn Sie die RDS-API-Operationen CreateDBCluster und ModifyDBCluster ausführen.

In Aurora-MySQL-Version 2 und höher enthält die Engine-Version in der AWS-Managementkonsole auch die Aurora-Version. Beim Upgraden des Clusters ändert sich der angezeigte Wert. Diese Änderung hilft Ihnen, die genauen Aurora MySQL-Versionen anzugeben und zu überprüfen, ohne dass Sie eine Verbindung zum Cluster herstellen oder SQL-Befehle ausführen müssen.

> **Tip**
>
> Für Aurora-Cluster, die über CloudFormation verwaltet werden, kann diese Änderung in der `EngineVersion`-Einstellung Aktionen von CloudFormation auslösen. Weitere Informationen dazu, wie CloudFormation Änderungen an der Einstellung `EngineVersion` behandelt werden, finden Sie inder CloudFormationDokumentation.

Überprüfen von Aurora MySQL-Versionen mit SQL

Die Aurora-Versionsnummern, die Sie in Ihrer Anwendung mit SQL-Abfragen abrufen können, verwenden das Format *<major version>*.*<minor version>*.*<patch version>*. Sie können diese Versionsnummer für jede DB-Instance in Ihrem Aurora MySQL-Cluster abrufen, indem Sie die `AURORA_VERSION`-Systemvariable abfragen. Verwenden Sie eine der folgenden Abfragen, um diese Versionsnummer zu erhalten.

```
select aurora_version();
select @@aurora_version;
```

Diese Abfragen erzeugen eine Ausgabe ähnlich der folgenden.

```
mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 3.05.2           | 3.05.2           |
+------------------+------------------+
```

Die Versionsnummern, die von der Konsole, CLI und der RDS-API mit den unter Prüfen oder Angeben von Aurora-MySQL-Engine-Versionen über AWS beschriebenen Techniken zurückgegeben werden, sind in der Regel aussagekräftiger.

LTS-Versionen (Long-Term-Support, Langzeit-Support) und Betaversionen für Amazon Aurora MySQL

Aurora MySQL bietet LTS-Versionen (Long-Term-Support, Langzeit-Support) und Betaversionen für einige Engine-Versionen von Aurora MySQL.

Themen

- Aurora MySQL Long-Term Support- (LTS, Langzeit-Support) Versionen
- Aurora-MySQL-Betaversionen

Aurora MySQL Long-Term Support- (LTS, Langzeit-Support) Versionen

Jede neue Aurora MySQL-Version bleibt für eine bestimmte Zeit zum Erstellen oder Aktualisieren eines DB-Clusters verfügbar. Nach diesem Zeitraum müssen Sie alle Cluster aktualisieren, die diese

Version verwenden. Sie können Ihren Cluster vor Ablauf des Supportzeitraums manuell aktualisieren, oder Aurora kann ihn automatisch für Sie aktualisieren, wenn die Aurora MySQL-Version nicht mehr unterstützt wird.

Aurora bezeichnet bestimmte Aurora MySQL-Versionen als „Long-term Support“ (LTS, Langzeitsupport)-Versionen. DB-Cluster, die LTS-Versionen verwenden, können länger dieselbe Version nutzen und weniger Upgradezyklen durchlaufen als Cluster, die Nicht-LTS-Versionen verwenden. Datenbank-Cluster, die LTS-Versionen verwenden, können mindestens drei Jahre dieselbe Nebenversion nutzen, oder bis zum Ende des Standard-Supports für die Hauptversion, je nachdem, was zuerst eintritt. Wenn ein DB-Cluster mit einer LTS-Version ein Upgrade benötigt, aktualisiert Aurora ihn auf die nächste LTS-Version. Auf diese Weise muss lange Zeit kein erneutes Upgrade für den Cluster durchgeführt werden.

Während der Lebensdauer einer Aurora MySQL-LTS-Version sorgen neue Patchlevels für Korrekturen bei wichtigen Problemen. Die Patchlevels enthalten keine neuen Funktionen. Sie können wählen, ob Sie solche Patches auf DB-Cluster anwenden möchten, die die LTS-Version ausführen. Wir empfehlen Kunden, die LTS-Versionen ausführen, mindestens einmal im Jahr ein Upgrade auf die neueste Patch-Version der LTS-Nebenversion durchzuführen, um von Sicherheits- und Betriebskorrekturen mit hohem Schweregrad zu profitieren. Für bestimmte kritische Korrekturen kann Amazon innerhalb derselben LTS-Version ein verwaltetes Upgrade auf einen Patchlevel durchführen. Solche verwalteten Upgrades werden automatisch innerhalb des Wartungsfensters des Clusters durchgeführt. Alle Aurora-MySQL-Versionen (sowohl LTS- als auch Nicht-LTS-Versionen) werden umfangreichen Stabilitäts- und Betriebstests unterzogen. Ausgewählte Nebenversionen werden als LTS-Versionen festgelegt, damit Kunden diese Nebenversionen länger verwenden können, ohne auf eine neuere Nebenversion aktualisieren zu müssen.

Wir empfehlen, dass Sie für die meisten Ihrer Aurora MySQL-Cluster auf die neueste Version upgraden, anstatt die LTS-Version zu verwenden. Auf diese Weise wird Aurora als Managed verwalteter Service genutzt und Sie erhalten Zugriff auf die neuesten Funktionen und Bugfixes. Die LTS-Releases sind für Cluster mit folgenden Merkmalen gedacht:

- Sie können sich, abgesehen von seltenen Fällen, keine Upgrade-Ausfallzeiten Ihrer Aurora MySQL-Anwendung für kritische Patches leisten.
- Der Testzyklus für den Cluster und die zugehörigen Anwendungen dauert bei einer Aktualisierung der Aurora MySQL-Datenbank-Engine sehr lange.
- Die Datenbankversion für Ihren Aurora MySQL-Cluster enthält alle Funktionen der DB-Engine und Bugfixes, die Ihre Anwendung benötigt.

Die aktuellen LTS-Versionen für Aurora MySQL sind:

- Aurora-MySQL-Version 3.10.*.
- Aurora MySQL Version 3.04.*.

Weitere Informationen zur LTS-Version finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL Version 3 in den Versionshinweisen für Aurora MySQL.

> **Note**
>
> Wir empfehlen, automatische Nebenversions-Upgrades für LTS-Versionen zu deaktivieren. Setzen Sie den Parameter `AutoMinorVersionUpgrade` auf `false` oder deaktivieren Sie das Kontrollkästchen Automatische Aktualisierung von Unterversionen aktivieren in der AWS-Managementkonsole.
> Wenn Sie es nicht deaktivieren, kann Ihr DB-Cluster auf eine Nicht-LTS-Version aktualisiert werden.

Aurora-MySQL-Betaversionen

Bei einer Aurora MySQL-Betaversion handelt es sich um ein frühes Sicherheitsupdate, das nur in einer begrenzten Anzahl von Versionen verfügbar ist. AWS-Regionen Diese Korrekturen werden später mit der nächsten Patch-Version in größerem Umfang in allen Regionen verfügbar sein.

Die Nummerierung für eine Betaversion ähnelt der einer Aurora-MySQL-Nebenversion, jedoch mit einer zusätzlichen vierten Ziffer, zum Beispiel 2.12.0.1 oder 3.05.0.1.

Weitere Informationen finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 2 und Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 3 in den Versionshinweisen für Aurora MySQL.

Vorbereitung auf das Ende des Standard-Supports für die Amazon-Aurora-MySQL-kompatible Edition der Version 2

Die Amazon-Aurora-MySQL-kompatible Edition der Version 2 (mit MySQL-5.7-Kompatibilität) soll am 31. Oktober 2024 das Ende des Standard-Supports erreichen. Wir empfehlen, alle Cluster mit Aurora-MySQL-Version 2 zur Standard–Aurora-MySQL-Version 3 (mit MySQL-8.0-Kompatibilität) oder höher

zu aktualisieren, bevor Aurora-MySQL-Version 2 das Ende des Standard-Supportzeitraums erreicht. Am 31. Oktober 2024 registriert Amazon RDS Ihre Datenbanken automatisch beim erweiterten Amazon-RDS-Support. Wenn Sie Amazon-Aurora-MySQL-Version 2 (mit MySQL-5.7-Kompatibilität) in einem Cluster von Aurora Serverless-Version 1 ausführen, gilt dies nicht für Sie. Wenn Sie Ihre Cluster von Aurora Serverless-Version 1 auf Aurora-MySQL-Version 3 aktualisieren möchten, finden Sie weitere Informationen unter Upgrade-Pfad für Aurora Serverless v1-DB-Cluster.

Die bevorstehenden Termine für das Support-Ende von Aurora-MySQL-Hauptversionen finden Sie im Veröffentlichungskalender für Aurora-MySQL-Hauptversionen.

Wenn Sie Cluster haben, auf denen Aurora-MySQL-Version 2 ausgeführt wird, erhalten Sie regelmäßig Benachrichtigungen mit den neuesten Informationen darüber, wie Sie ein Upgrade durchführen, sobald wir uns dem Datum des Endes des Standard-Supports nähern. Wir werden diese Seite regelmäßig mit den neuesten Informationen aktualisieren.

Ende des Standard-Supportzeitplans

1. Jetzt bis zum 31. Oktober 2024 – Sie können jederzeit Upgrades von Aurora-MySQL-Clustern der Version 2 (mit MySQL 5.7-Kompatibilität) auf Aurora MySQL Version 3 (mit MySQL 8.0-Kompatibilität) starten.
2. 31. Oktober 2024 – Zu diesem Zeitpunkt erreicht Aurora-MySQL-Version 2 das Ende des Standard-Supports und Amazon RDS registriert Ihre Cluster automatisch beim erweiterten Amazon-RDS-Support.

Wir registrieren Sie automatisch beim erweiterten RDS-Support. Weitere Informationen finden Sie unter Amazon RDS Extended Support mit Amazon Aurora.

Finden von Clustern, die von diesem Lebenszyklusende-Prozess betroffen sind

Gehen Sie wie folgt vor, um Cluster zu finden, die von diesem Lebenszyklusende-Prozess betroffen sind.

> **Important**
>
> Stellen Sie sicher, dass Sie diese Anweisungen in jeder AWS-Region und für jedes AWS-Konto ausführen, in der sich Ihre Ressourcen befinden.

Konsole

So finden Sie einen Aurora-MySQL-Cluster der Version 2

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Geben Sie im Feld Nach Datenbanken filtern 5.7 ein.
4. Suchen Sie in der Engine-Spalte nach Aurora MySQL.

AWS CLI

Um unter Verwendung von AWS CLI Cluster zu finden, die von diesem Lebenszyklusende-Prozess betroffen sind, rufen Sie den describe-db-clusters-Befehl auf. Sie können das folgende Beispielskript verwenden.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?(Engine==`aurora-
mysql` && contains(EngineVersion,`5.7.mysql_aurora`))].{EngineVersion:EngineVersion,
 DBClusterIdentifier:DBClusterIdentifier, EngineMode:EngineMode}' --output table
 --region us-east-1
```

```
+-----------------------------------------------------------------+
|                       DescribeDBClusters                        |
+----------------------+---------------+--------------------------+
|         DBCI         |      EM       |          EV              |
+----------------------+---------------+--------------------------+
|    aurora-mysql2     |  provisioned  | 5.7.mysql_aurora.2.11.3 |
| aurora-serverlessv1 |   serverless  | 5.7.mysql_aurora.2.11.3 |
+----------------------+---------------+--------------------------+
```

RDS-API

Um Aurora-MySQL-DB-Cluster zu finden, auf denen Aurora MySQL Version 2 ausgeführt wird, verwenden Sie den RDS-DescribeDBClusters-API-Vorgang mit folgenden erforderlichen Parametern:

- `DescribeDBClusters`
  - Filters.Filter.N

- Name
  - engine
- Values.Value.N
  - ['aurora']

Amazon RDS Extended Support

Sie können den erweiterten Amazon-RDS-Support bis zum Ende des Supportdatums, dem 31. Oktober 2024, kostenlos über Community-MySQL 5.7 nutzen. Am 31. Oktober 2024 registriert Amazon RDS Ihre Datenbanken automatisch beim erweiterten RDS-Support für Aurora-MySQL-Version 2. Der erweiterte RDS-Support für Aurora ist ein kostenpflichtiger Service, der bis zum Ende des erweiterten RDS-Supports im Februar 2027 bis zu 28 zusätzliche Monate Support für Aurora-MySQL-Version 2 bietet. RDS Extended Support wird nur für die Aurora-MySQL-Nebenversionen 2.11 und 2.12 angeboten. Wenn Sie Amazon-Aurora-MySQL-Version 2 über das Ende des Standardsupports hinaus verwenden möchten, planen Sie, Ihre Datenbanken vor dem 31. Oktober 2024 auf einer dieser Nebenversionen auszuführen.

Weitere Informationen zum erweiterten RDS-Support, z. B. zu Gebühren und anderen Überlegungen, finden Sie unter Amazon RDS Extended Support mit Amazon Aurora.

Durchführen eines Upgrades

Das Upgrade zwischen Hauptversionen erfordert umfangreichere Planung und Tests als für eine Nebenversion. Der Prozess kann beträchtliche Zeit in Anspruch nehmen. Wir möchten das Upgrade als dreistufigen Prozess mit Aktivitäten vor dem Upgrade, für das Upgrade und nach dem Upgrade betrachten.

Vor dem Upgrade:

Wir empfehlen, vor der Aktualisierung die Anwendungskompatibilität, die Leistung, Wartungsverfahren und ähnliche Dinge für den aktualisierten Cluster zu prüfen, um sicherzustellen, dass Ihre Anwendungen nach der Aktualisierung wie erwartet funktionieren werden. Im Folgenden finden Sie fünf Empfehlungen, die Ihnen zu einem besseren Upgrade-Komfort verhelfen sollen.

- Zunächst ist es wichtig, Funktionsweise des Aurora MySQL direkten Upgrade der Hauptversion zu verstehen.
- Machen Sie sich als Nächstes mit den Ihnen zur Verfügung stehenden Upgrade-Techniken vertraut, wenn Upgrade von Aurora-MySQL-Version 2 auf Version 3.

- Um Ihnen bei der Entscheidung für den richtigen Zeitpunkt und den richtigen Ansatz für das Upgrade zu helfen, können Sie die Unterschiede zwischen Aurora-MySQL-Version 3 und Ihrer aktuellen Umgebung mit Vergleich von Aurora-MySQL-Version 2 und Aurora-MySQL-Version 3 kennenlernen.
- Nachdem Sie sich für die Option entschieden haben, die für Sie praktisch ist und am besten funktioniert, versuchen Sie es mit einem simulierten direkten Upgrade auf einem geklonten Cluster. Verwenden Sie dazu Planen eines Hauptversionsupgrades für einen Aurora MySQL-Cluster.
- Überprüfen Sie das Vorabprüfungen für Hauptversions-Upgrades von Aurora MySQL. Die Upgrade-Vorabprüfung kann ausgeführt werden und feststellen, ob die Datenbank erfolgreich aktualisiert werden kann und ob es nach dem Upgrade Probleme mit der Anwendungsinkompatibilität gibt, ebenso wie Leistung, Wartungsverfahren und ähnliche Überlegungen.
- Nicht alle Arten oder Versionen von Aurora MySQL Clustern können den integrierten Upgrade-Mechanismus verwenden. Weitere Informationen finden Sie unter Aurora MySQL Upgrade-Vorgang für die Hauptversion.

Bei Fragen oder Bedenken steht Ihnen das AWS-Support-Team in den Community-Foren und der Premium-Support zur Verfügung.

Durchführen der Aktualisierung:

Verwenden Sie eine der folgenden Upgrade-Techniken. Wie lange Ihr System ausfallen wird, hängt von der gewählten Technik ab.

- Blau/Grün-Bereitstellungen – In Situationen, in denen die Reduzierung von Ausfallzeiten oberste Priorität hat, können Sie auch Amazon-RDS-Blau/Grün-Bereitstellungen verwenden, um das Hauptversions-Upgrade in bereitgestellten DB-Clustern von Amazon Aurora durchzuführen. Mit einer Blau/Grün-Bereitstellung wird eine Staging-Umgebung erstellt, die die Produktionsumgebung kopiert. Sie können bestimmte Änderungen am Aurora-DB-Cluster in der grünen (Staging-)Umgebung vornehmen, ohne die Produktions-Workloads zu beeinträchtigen. Die Umstellung dauert in der Regel weniger als eine Minute, ohne dass Daten verloren gehen. Weitere Informationen finden Sie unter Überblick über Aurora Blue/Green Aurora-Bereitstellungen. Dadurch werden Ausfallzeiten minimiert. Sie müssen jedoch während der Durchführung des Upgrades zusätzliche Ressourcen einsetzen.
- Direkte Upgrades – Sie können ein direktes Upgrade durchführen, bei dem Aurora automatisch eine Vorabprüfung für Sie ausführt, den Cluster offline nimmt, Ihren Cluster sichert, das Upgrade durchführt und Ihren Cluster wieder online stellt. Ein direktes Upgrade der Hauptversion kann mit

wenigen Klicks durchgeführt werden und erfordert keine weitere Koordination oder Failovers mit anderen Clustern, ist jedoch mit Ausfallzeiten verbunden. Weitere Informationen finden Sie unter Erläuterung der Durchführung eines direkten Upgrades

- Snapshot-Wiederherstellung – Sie können Ihren Cluster von Aurora-MySQL-Version 2 aktualisieren, indem Sie eine Wiederherstellung aus einem Snapshot von Aurora-MySQL-Version 2 zu einem Cluster von Aurora-MySQL-Version 3 durchführen. Um dies zu tun, sollten Sie den Prozess zum Erstellen eines Snapshots und zum Wiederherstellen von diesem befolgen. Dieser Prozess beinhaltet eine Unterbrechung der Datenbank, weil Sie aus einem Snapshot wiederherstellen.

Nach dem Upgrade:

Nach dem Upgrade müssen Sie Ihr System (Anwendung und Datenbank) genau überwachen und gegebenenfalls Feinabstimmungen vornehmen. Wenn Sie die Schritte vor dem Upgrade genau befolgen, werden die erforderlichen Änderungen auf ein Minimum reduziert. Weitere Informationen finden Sie unter Beheben von Problemen mit der Datenbankleistung von Amazon Aurora MySQL.

Wenn Sie mehr über die Methoden, die Planung, das Testen und die Fehlerbehebung von Aurora-MySQL-Hauptversions-Upgrades erfahren möchten, lesen Sie unbedingt Aktualisieren der Hauptversion eines DB-Clusters von Amazon Aurora MySQL, einschließlich Problembehandlung für Aurora MySQL direkte Upgrades. Beachten Sie weiterhin, dass manche Instance-Typen nicht für Aurora-MySQL-Version 3 unterstützt werden. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Upgrade-Pfad für Aurora Serverless v1-DB-Cluster

Das Upgrade zwischen Hauptversionen erfordert umfangreichere Planung und Tests als für eine Nebenversion. Der Prozess kann beträchtliche Zeit in Anspruch nehmen. Wir möchten das Upgrade als dreistufigen Prozess mit Aktivitäten vor dem Upgrade, für das Upgrade und nach dem Upgrade betrachten.

Aurora-MySQL-Version 2 (mit MySQL-5.7-Kompatibilität) erhält weiterhin den Standard-Support für Aurora Serverless v1-Cluster.

Wenn Sie ein Upgrade zu Amazon Aurora MySQL 3 (mit MySQL-8.0-Kompatibilität) durchführen und weiterhin Aurora Serverless ausführen möchten, können Sie Amazon Aurora Serverless v2 verwenden. Informationen zu den Unterschieden zwischen Aurora Serverless v1 und Aurora Serverless v2 finden Sie unter Aurora Serverless v2 und Aurora Serverless v1 im Vergleich.

Upgrade auf Aurora Serverless v2: Sie können einen Aurora Serverless v1-Cluster auf Aurora Serverless v2 aktualisieren. Weitere Informationen finden Sie unter Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2.

Vorbereitung auf das Lebenszyzklusende der Amazon Aurora MySQL-kompatible Edition Version 1

Amazon-Aurora-MySQL-kompatible Edition Version 1 (mit MySQL 5.6-Kompatibilität) soll am 28. Februar 2023 das Lebenszyzklusende erreichen. Amazon rät, dass Sie alle (bereitgestellten) Cluster (und Aurora Serverless) mit Aurora MySQL Version 1 auf Aurora MySQL Version 2 (mit MySQL 5.7-Kompatibilität) oder Aurora MySQL Version 3 (mit MySQL 8.0 Kompatibilität) aktualisieren. Tun Sie dies, bevor Aurora-MySQL-Version 1 das Ende ihres Supportzeitraums erreicht.

Für Aurora-bereitgestellte DB-Cluster können Sie Upgrades von Aurora MySQL Version 1 auf Aurora MySQL Version 2 mit verschiedenen Methoden durchführen. Eine Anleitung zum Direkt-Upgrade-Mechanismus finden Sie unter Erläuterung der Durchführung eines direkten Upgrades. Eine weitere Möglichkeit, das Upgrade abzuschließen, besteht darin, einen Snapshot eines Aurora-MySQL-Version-1-Clusters zu erstellen und den Snapshot in einem Aurora-MySQL-Version-2-Cluster wiederherzustellen. Oder Sie können einen mehrstufigen Prozess verfolgen, bei dem die alten und neuen Cluster nebeneinander ausgeführt werden. Weitere Einzelheiten zu den einzelnen Methoden finden Sie unter Aktualisieren der Hauptversion eines DB-Clusters von Amazon Aurora MySQL.

Für DB-Cluster von Aurora Serverless v1 können Sie ein direktes Upgrade von Aurora-MySQL-Version 1 auf Aurora-MySQL-Version 2 durchführen. Weitere Einzelheiten zu dieser Methode finden Sie unter Ändern eines Aurora Serverless v1-DB-Clusters.

Für Aurora-bereitgestellte DB-Cluster können Sie Upgrades von Aurora-MySQL-Version 1 auf Aurora-MySQL-Version 3 mithilfe eines zweistufigen Upgrade-Prozesses durchführen:

1. Führen Sie ein Upgrade von Aurora-MySQL-Version 1 auf Aurora-MySQL-Version 2 unter Verwendung der zuvor beschriebenen Methoden durch.
2. Aktualisieren Sie von Aurora-MySQL-Version 2 auf Aurora-MySQL-Version 3 mit den gleichen Methoden wie von Version 1 auf Version 2. Weitere Details finden Sie unter Upgrade von Aurora-MySQL-Version 2 auf Version 3. Beachten Sie den Funktionsunterschiede zwischen Aurora MySQL Version 2 und 3.

Die Datumsangaben zum Ende des Lebenszyklus für Aurora-Hauptversionen finden Sie in Amazon-Aurora-Versionen. Amazon aktualisiert automatisch alle Cluster, die Sie vor dem

Ende des Lebenszyklus nicht selbst aktualisieren. Nach dem Ende des Lebenszyklus erfolgen diese automatischen Upgrades auf die nachfolgende Hauptversion während eines geplanten Wartungsfensters für Cluster.

Im Folgenden finden Sie zusätzliche Meilensteine für das Upgrade von Aurora-MySQL-Cluster der Version 1 (bereitgestellt und Aurora Serverless), die das Ende des Lebenszyklus erreichen. Für alle ist die Startzeit 00:00 Uhr (UTC).

1. Jetzt bis zum 28. Februar 2023 – Sie können jederzeit Upgrades von Aurora-MySQL-Cluster der Version 1 (mit MySQL 5.6-Kompatibilität) auf Aurora MySQL Version 2 (mit MySQL 5.7-Kompatibilität) starten. Von Aurora-MySQL-Version 2 aus können Sie ein weiteres Upgrade auf Aurora-MySQL-Version 3 (mit MySQL 8.0-Kompatibilität) für von Aurora bereitgestellte DB-Cluster durchführen.
2. 16. Januar 2023 – Ab diesem Zeitpunkt können Sie keine neuen Cluster oder Instances von Aurora MySQL Version 1 über die AWS-Managementkonsole oder die AWS Command Line Interface (AWS CLI) erstellen. Sie können einer globalen Aurora-Datenbank auch keine neuen sekundären Regionen hinzufügen. Dies kann sich auf Ihre Fähigkeit auswirken, sich nach einem ungeplanten Ausfall wie in Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall beschrieben zu erholen, da Sie die Schritte 5 und 6 nach dieser Zeit nicht ausführen können. Sie können auch kein neues regionsübergreifendes Lesereplikat erstellen, auf dem Aurora MySQL Version 1 ausgeführt wird. Für bestehende Aurora-MySQL-Cluster der Version 1 können Sie bis zum 28. Februar 2023 immer noch Folgendes tun:
   - Wiederherstellen eines Snapshots eines Clusters von Aurora-MySQL-Version 1 auf die Version des ursprünglichen Snapshot-Clusters.
   - Lesereplikate hinzufügen (gilt nicht für Aurora Serverless-DB-Cluster).
   - Ändern der Instance-Konfiguration.
   - Durchführen einer Point-in-Time-Wiederherstellung.
   - Erstellen von Klonen bestehender Cluster der Version 1.
   - Erstellen eines neuen regionsübergreifendes Lesereplikats, auf dem Aurora-MySQL-Version 2 oder höher ausgeführt wird.
3. 28. Februar 2023 – Nach dieser Zeit planen wir, Aurora-MySQL-Cluster der Version 1 innerhalb eines darauf folgenden geplanten Wartungsfensters automatisch auf die Standardversion von Aurora-MySQL-Version 2 zu aktualisieren. Das Wiederherstellen von Aurora-MySQL-Version-1-DB-Snapshots führt zu einem automatischen Upgrade des wiederhergestellten Clusters auf die Standardversion von Aurora MySQL Version 2 zu diesem Zeitpunkt.

Das Upgrade zwischen Hauptversionen erfordert umfangreichere Planung und Tests als für eine Nebenversion. Der Prozess kann beträchtliche Zeit in Anspruch nehmen.

In Situationen, in denen die Reduzierung von Ausfallzeiten oberste Priorität hat, können Sie auch Blau/Grün-Bereitstellungen verwenden, um das Hauptversions-Upgrade in bereitgestellten DB-Clustern von Amazon Aurora durchzuführen. Mit einer Blau/Grün-Bereitstellung wird eine Staging-Umgebung erstellt, die die Produktionsumgebung kopiert. Sie können Änderungen am Aurora-DB-Cluster in der grünen (Staging-) Umgebung vornehmen, ohne die Produktions-Workloads zu beeinträchtigen. Die Umstellung dauert in der Regel weniger als eine Minute, ohne dass Daten verloren gehen und Anwendungsänderungen erforderlich sind. Weitere Informationen finden Sie unter Überblick über Aurora Blue/Green Aurora-Bereitstellungen.

Nachdem das Upgrade abgeschlossen ist, müssen Sie möglicherweise auch Follow-up-Arbeiten ausführen. Beispielsweise müssen Sie möglicherweise aufgrund von Unterschieden in der SQL-Kompatibilität, der Funktionsweise bestimmter MySQL-bezogener Funktionen oder Parametereinstellungen zwischen der alten und der neuen Version nachverfolgen.

Um mehr über die Methoden, die Planung, das Testen und die Fehlerbehebung von Aurora-MySQL-Hauptversions-Upgrades zu erfahren, lesen Sie unbedingt Aktualisieren der Hauptversion eines DB-Clusters von Amazon Aurora MySQL .

Finden von Clustern, die von diesem Lebenszyklusende-Prozess betroffen sind

Gehen Sie wie folgt vor, um Cluster zu finden, die von diesem Lebenszyklusende-Prozess betroffen sind.

> **Important**
>
> Stellen Sie sicher, dass Sie diese Anweisungen in jeder AWS-Region und für jedes AWS-Konto ausführen, in der sich Ihre Ressourcen befinden.

Konsole

Suchen Sie einen Aurora-MySQL-Cluster der Version 1 wie folgt:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.

3. Geben Sie im Feld Nach Datenbanken filtern 5.6 ein.
4. Suchen Sie in der Engine-Spalte nach Aurora MySQL.

AWS CLI

Um unter Verwendung von AWS CLI Cluster zu finden, die von diesem Lebenszyklusende-Prozess betroffen sind, rufen Sie den describe-db-clusters-Befehl auf. Sie können das folgende Beispielskript verwenden.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].
{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region
 us-east-1

        +-----------------------------------------+
        |            DescribeDBClusters           |
        +---------------+--------------+----------+
        |     DBCI      |     EM       |    EV    |
        +---------------+--------------+----------+
        |  my-database-1|  serverless  |  5.6.10a |
        +---------------+--------------+----------+
```

RDS-API

Um Aurora-MySQL-DB-Cluster zu finden, auf denen Aurora MySQL Version 1 ausgeführt wird, verwenden Sie den RDS-DescribeDBClusters-API-Vorgang mit folgenden erforderlichen Parametern:

- `DescribeDBClusters`
  - Filters.Filter.N
    - Name
      - engine
    - Values.Value.N
      - ['aurora']

Upgrade von Amazon Aurora MySQL-DB-Clustern

Sie können einen Aurora MySQL-DB-Cluster aktualisieren, um Bugfixes oder neue Aurora MySQL-Funktionen zu erhalten oder zu einer völlig neuen Version der zugrunde liegenden Datenbank-Engine zu wechseln. Die folgenden Abschnitte zeigen wie.

> **Note**
>
> Die Art des Upgrades, das Sie durchführen, hängt davon ab, wie viel Ausfallzeit Sie sich für Ihren Cluster leisten können, wie viele Verifizierungstests Sie durchführen möchten, wie wichtig die spezifischen Bugfixes oder neuen Funktionen für Ihren Anwendungsfall sind und ob Sie vorhaben häufig kleine Upgrades oder gelegentliche Upgrades durchführen, die mehrere Zwischenversionen überspringen. Für jedes Upgrade können Sie die Hauptversion, die Nebenversion und die Patch-Ebene für Ihren Cluster ändern. Wenn Sie mit der Unterscheidung zwischen Aurora MySQL-Hauptversionen, Nebenversionen und Patch-Ebenen nicht vertraut sind, können Sie die Hintergrundinformationen unter lese Überprüfen von Aurora-MySQL-Versionsnummern.

> **Tip**
>
> Sie können die für ein DB-Cluster-Upgrade erforderlichen Ausfallzeiten minimieren, indem Sie eine Blau/Grün-Bereitstellung verwenden. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

Themen

- Upgrade der Nebenversion oder der Patch-Ebene eines Aurora MySQL-DB-Clusters
- Aktualisieren der Hauptversion eines DB-Clusters von Amazon Aurora MySQL

Upgrade der Nebenversion oder der Patch-Ebene eines Aurora MySQL-DB-Clusters

Sie können die folgenden Methoden verwenden, um die Nebenversion eines DB-Clusters zu aktualisieren oder einen DB-Cluster zu patchen:

- Upgrade von Aurora MySQL durch Ändern der Engine-Version (für Aurora-MySQL-Version 2) und 3
- Aktivieren von automatischen Upgrades zwischen Aurora MySQL-Nebenversionen

Informationen dazu, wie Patches ohne Ausfallzeiten Unterbrechungen während des Upgrade-Vorgangs reduzieren können, finden Sie unter Verwendung von Zero-Downtime-Patching (Patchen ohne Ausfallzeiten).

Informationen zum Durchführen eines Nebenversions-Upgrades für Ihren DB-Cluster von Aurora MySQL finden Sie in den folgenden Themen.

Themen

- Vor dem Durchführen eines Nebenversions-Upgrades
- Vorabprüfungen für Nebenversions-Upgrades von Aurora MySQL
- Upgrade von Aurora MySQL durch Ändern der Engine-Version
- Aktivieren von automatischen Upgrades zwischen Aurora MySQL-Nebenversionen
- Verwendung von Zero-Downtime-Patching (Patchen ohne Ausfallzeiten)
- Alternatives Blau/Grün-Upgradeverfahren

Vor dem Durchführen eines Nebenversions-Upgrades

Wir empfehlen Ihnen, die folgenden Aktionen durchzuführen, um die Ausfallzeit während eines Nebenversions-Upgrades zu reduzieren:

- Die Wartung des Aurora-DB-Clusters sollte in Zeiten mit geringem Datenverkehr durchgeführt werden. Verwenden Sie Performance Insights, um diese Zeiträume zu identifizieren und die Wartungsfenster korrekt zu konfigurieren. Weitere Informationen zu Performance Insights finden Sie unter Überwachen der DB-Auslastung mit Performance Insights in Amazon RDS. Weitere Informationen zu DB-Cluster-Wartungsfenstern finden Sie unter Anpassen des bevorzugten DB-Cluster-Wartungsfensters.
- Verwenden Sie als bewährte Methode AWS SDKs, die exponentielles Backoff und Jitter unterstützen. Weitere Informationen finden Sie unter Exponentielles Backoff und Jitter.

Vorabprüfungen für Nebenversions-Upgrades von Aurora MySQL

Wenn Sie ein Nebenversions-Upgrade starten, führt Amazon Aurora automatisch Vorabprüfungen durch.

Diese Vorabprüfungen müssen durchgeführt werden. Sie können nicht ausgelassen werden. Die Vorabprüfungen bieten folgende Vorteile:

- Sie können ungeplante Ausfallzeiten während des Upgrades vermeiden.
- Wenn es Inkompatibilitäten gibt, verhindert Amazon Aurora das Upgrade und stellt Ihnen ein Protokoll mit Informationen zu den Inkompatibilitäten bereit. Sie können das Protokoll für die Vorbereitung Ihrer Datenbank auf das Upgrade verwenden, indem Sie die Inkompatibilitäten reduzieren. Detaillierte Informationen zum Entfernen von Inkompatibilitäten finden Sie unter Vorbereiten Ihrer Installation auf ein Upgrade in der MySQL-Dokumentation.

Die Vorabprüfungen werden ausgeführt, bevor die DB-Instance aufgrund des Upgrades angehalten wird. Sie verursachen also keine Ausfallzeiten. Wird während der Vorabprüfungen eine Inkompatibilität entdeckt, bricht Aurora automatisch das Upgrade ab, ehe die DB-Instance angehalten wird. Aurora generiert auch ein Ereignis für die Inkompatibilität. Weitere Informationen über Amazon-Aurora-Ereignisse finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

Aurora zeichnet detaillierte Informationen zu allen Inkompatibilitäten in der Protokolldatei `PrePatchCompatibility.log` auf. In den meisten Fällen enthalten die Protokolleinträge einen Link zur MySQL-Dokumentation mit Informationen zur Lösung des Inkompatibilitätsproblems. Weitere Informationen zum Anzeigen von Protokolldateien finden Sie unter Anzeigen und Auflisten von Datenbank-Protokolldateien.

Aufgrund der Art der Vorabprüfungen werden die Objekte in Ihrer Datenbank geprüft. Diese Analyse verbraucht Ressourcen und verlängert die Zeit, die für die Durchführung des Upgrades benötigt wird.

Upgrade von Aurora MySQL durch Ändern der Engine-Version

Beim Upgrade der Nebenversion eines DB-Clusters von Aurora MySQL werden zusätzliche Korrekturen und neue Funktionen auf einen vorhandenen Cluster angewendet.

Diese Art von Upgrade gilt für Aurora-MySQL-Cluster, bei denen die ursprüngliche Version und die aktualisierte Version dieselbe Aurora-MySQL-Hauptversion haben, entweder 2 oder 3. Der Prozess ist schnell und unkompliziert, da er keine Konvertierung für die Aurora-MySQL-Metadaten oder Reorganisation Ihrer Tabellendaten erfordert.

Sie führen diese Art von Upgrade durch, indem Sie die Engine-Version des DB-Clusters mithilfe der AWS-Managementkonsole, AWS CLI oder der RDS-API ändern. Wenn beispielsweise in Ihrem Cluster Aurora MySQL 3.x ausgeführt wird, wählen Sie eine höhere 3.x-Version.

Wenn Sie ein Nebenversions-Upgrade für eine Aurora Global Database durchführen, aktualisieren Sie alle sekundären Cluster, bevor Sie den primären Cluster aktualisieren.

> **Note**
>
> Gehen Sie wie folgt vor, um ein Unterversion-Upgrade auf Aurora-MySQL-Version 3.04.* oder höher bzw. Version 2.12* durchzuführen:
>
> 1. Entfernen Sie alle sekundären Regionen aus dem globalen Cluster. Führen Sie die Schritte unter au Entfernen eines Clusters aus einer Amazon Aurora Global Database.
> 2. Aktualisieren Sie die Engine-Version der primären Region auf Version 3.04.* oder höher bzw. Version 2.12.*, falls zutreffend. Führen Sie die Schritte unter au To modify the engine version of a DB cluster.
> 3. Fügen Sie dem globalen Cluster sekundäre Regionen hinzu. Führen Sie die Schritte unter au Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.

So ändern Sie die Engine-Version eines DB-Clusters:

- Über die Konsole – Ändern Sie die Eigenschaften Ihres Clusters. Ändern Sie im Fenster DB-Cluster ändern die Aurora-MySQL-Engine-Version im Feld DB-Engine-Version. Wenn Sie mit dem allgemeinen Verfahren zum Ändern eines Clusters nicht vertraut sind, folgen Sie den Anweisungen unter Ändern des DB-Clusters über die Konsole, die CLI und die API.
- Über die AWS CLI Rufen Sie den AWS CLI-Befehl modify-db-cluster auf und geben Sie den Namen Ihres DB-Clusters für die Option `--db-cluster-identifier` sowie die Engine-Version für die Option `--engine-version` an.

  Um beispielsweise auf Aurora-MySQL-Version 3.04.1 zu aktualisieren, legen Sie die Option `--engine-version` auf `8.0.mysql_aurora.3.04.1` fest. Geben Sie die Option `--apply-immediately` an, um die Engine-Version für Ihren DB-Cluster sofort zu aktualisieren.
- Über die RDS-API Rufen Sie die API-Operation ModifyDBCluster auf und geben Sie den Namen Ihres DB-Clusters für den Parameter `DBClusterIdentifier` sowie die Engine-Version für den Parameter `EngineVersion` an. Legen Sie den Parameter `ApplyImmediately` auf `true` fest, um die Engine-Version für Ihren DB-Cluster sofort zu aktualisieren.

Aktivieren von automatischen Upgrades zwischen Aurora MySQL-Nebenversionen

Für einen DB-Cluster von Amazon Aurora MySQL können Sie angeben, dass Aurora den DB-Cluster automatisch auf neue Nebenversionen aktualisiert. Sie tun dies, indem Sie die

`AutoMinorVersionUpgrade`-Eigenschaft (Automatisches Unterversion-Upgrade in der AWS-Managementkonsole) des DB-Clusters festlegen.

Automatische Aktualisierungen erfolgen während des Wartungsfensters. Wenn die einzelnen DB-Instances im DB-Cluster andere Wartungsfenster haben als das Cluster-Wartungsfenster, hat das Cluster-Wartungsfenster Vorrang.

Das automatische Nebenversions-Upgrade gilt nicht für die folgenden Aurora MySQL-Clustertypen:

- Cluster, die Teil einer globalen Aurora-Datenbank sind
- Cluster, die über regionsübergreifende Replikate verfügen

Die Ausfalldauer hängt vom Workload, der Cluster-Größe, der Menge der binären Protokolldaten und davon ab, ob Aurora die Funktion „Patching ohne Ausfallzeiten“ (Zero Downtime Patching, ZDP) verwenden kann. Aurora startet den Datenbankcluster neu, sodass Sie möglicherweise eine kurze Zeit nicht verfügbar sind, bevor Sie Ihren Cluster wieder verwenden. Insbesondere wirkt sich die Menge der Binärprotokolldaten auf die Wiederherstellungszeit aus. Die DB-Instance verarbeitet die binären Protokolldaten während der Wiederherstellung. Somit erhöht eine hohe Menge an binären Protokolldaten die Wiederherstellungszeit.

> **Note**
>
> Aurora führt automatische Upgrades nur durch, wenn alle DB-Instances in Ihrem DB-Cluster die `AutoMinorVersionUpgrade`-Einstellung aktiviert haben. Informationen zum Festlegen der Einstellung und zu deren Funktionsweise, wenn sie auf Cluster- und Instance-Ebene angewendet wird, finden Sie unter Automatische Nebenversions-Upgrades für Aurora-DB-Cluster.
>
> Wenn dann ein Upgrade-Pfad für die Instances des DB-Clusters auf eine DB-Engine-Nebenversion existiert, bei der `AutoUpgrade` auf „true“ gesetzt ist, wird das Upgrade durchgeführt. Die `AutoUpgrade`-Einstellung ist dynamisch und wird von RDS festgelegt. Automatische Nebenversions-Upgrades werden auf die standardmäßige Nebenversion durchgeführt.

Sie können einen CLI-Befehl wie den folgenden verwenden, um den Status der Einstellung `AutoMinorVersionUpgrade` für alle DB-Instances in Ihren Aurora-MySQL-Clustern zu überprüfen.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Die Ausgabe dieses Befehls sieht etwa so aus:

```
[
  {
      "DBInstanceIdentifier": "db-t2-medium-instance",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-t2-small-original-size",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "instance-2020-05-01-2332",
      "DBClusterIdentifier": "cluster-57-2020-05-01-4615",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

In diesem Beispiel ist die Einstellung Automatisches Nebenversions-Upgrade aktivieren für den DB-Cluster `cluster-57-2020-06-03-6411` deaktiviert, da sie für eine der DB-Instances im Cluster deaktiviert ist.

Verwendung von Zero-Downtime-Patching (Patchen ohne Ausfallzeiten)

Das Durchführen von Upgrades für Aurora MySQL DB-Cluster beinhaltet die Möglichkeit eines Ausfalls, wenn die Datenbank heruntergefahren wird und während sie aktualisiert wird. Wenn Sie das Upgrade starten, während die Datenbank ausgelastet ist, verlieren Sie standardmäßig alle Verbindungen und Transaktionen, die der DB-Cluster verarbeitet. Wenn Sie warten, bis die Datenbank im Leerlauf ist, um das Upgrade durchzuführen, müssen Sie möglicherweise lange warten.

Beim Feature des Patchens ohne Ausfallzeiten (ZDP – Zero-Downtime Patching) wird versucht, Client-Verbindungen auf Best-Effort-Basis vor der Aurora MySQL-Aktualisierung zu bewahren. Wenn das ZDP erfolgreich abgeschlossen wird, werden Anwendungssitzungen bewahrt und die Datenbank-

Engine wird während der laufenden Aktualisierung neu gestartet. Der Neustart der Datenbank-Engine kann zu einem Abfall des Durchsatzes von eingen Sekunden bis 1 Minute führen.

ZDP gilt nicht für Folgendes:

- Patches und Upgrades für das Betriebssystem
- Hauptversions-Upgrades

ZDP ist für alle unterstützten Aurora-MySQL-Versionen und DB-Instance-Klassen verfügbar.

ZDP wird für Aurora Serverless v1- oder globale Aurora-Datenbanken nicht unterstützt.

> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter Verwendung von T-Instance-Klassen für Entwicklung und Tests.

Sie können Metriken wichtiger Attribute während des ZDP im MySQL-Fehlerprotokoll sehen. Sie können auch Informationen darüber sehen, wann Aurora MySQL ZDP verwendet oder nicht verwendet und zwar auf der Seite Ereignisse der AWS-Managementkonsole.

In Aurora MySQL kann Aurora einen Patch ohne Ausfallzeit ausführen, unabhängig davon, ob die binäre Protokollreplikation aktiviert ist. Wenn die binäre Protokollreplikation aktiviert ist, löscht Aurora MySQL während eines ZDP-Vorgangs automatisch die Verbindung zum Binärprotokollziel. Aurora MySQL stellt automatisch eine Verbindung zum binlog-Ziel her und setzt die Replikation fort, nachdem der Neustart abgeschlossen ist.

Das ZDP arbeitet auch in Kombination mit den Neustartverbesserungen in Aurora MySQL. Durch das Patchen der Writer-DB-Instance werden die Leser automatisch gleichzeitig gepatcht. Aurora stellt nach dem Ausführen des Patches die Verbindungen sowohl auf den Writer- als auch der Reader-DB-Instances wieder

ZDP kann unter den folgenden Bedingungen möglicherweise nicht erfolgreich abgeschlossen werden:

- Wenn langandauernde Abfragen oder Transaktionen ausgeführt werden. Wenn Aurora in diesem Fall ZDP durchführen kann, werden alle offenen Transaktionen abgebrochen, aber ihre Verbindungen beibehalten.

- Es werden temporäre Tabellen, Benutzer- oder Tabellensperren verwendet, beispielsweise während Anweisungen für die Datendefinitionssprache (DDL) ausgeführt werden. Aurora trennt diese Verbindungen.
- Wenn ausstehende Parameteränderungen vorhanden sind.

Wenn sich kein passendes Zeitfenster für die Durchführung von ZDP finden lässt, weil eine dieser Bedingungen vorliegt, wird das Patchen im normalen Modus ausgeführt.

Obwohl Verbindungen nach einem erfolgreichen ZDP-Vorgang intakt bleiben, werden einige Variablen und Funktionen neu initialisiert. Die folgenden Arten von Informationen werden durch einen Neustart, der durch Patchen ohne Ausfallzeiten verursacht wird, nicht beibehalten:

- Globale Variablen Aurora stellt Sitzungsvariablen wieder her, stellt jedoch nach dem Neustart keine globalen Variablen wieder her.
- Statusvariablen. Insbesondere wird der vom Engine-Status gemeldete Betriebszeit-Wert nach einem Neustart zurückgesetzt, der die ZDR- oder ZDP-Mechanismen verwendet.
- `LAST_INSERT_ID`.
- In-Memory-`auto_increment`-Status für Tabellen. Der Status des automatischen In-Memory-Inkrements wird neu initialisiert. Weitere Informationen zu automatischen Inkrementwerten finden Sie im MySQL-Referenzhandbuch.
- Diagnoseinformationen aus `INFORMATION_SCHEMA`- und `PERFORMANCE_SCHEMA`-Tabellen. Diese Diagnoseinformationen erscheinen auch in der Ausgabe von Befehlen wie `SHOW PROFILE` und `SHOW PROFILES`.

Die folgenden Aktivitäten im Zusammenhang mit einem Neustart ohne Ausfallzeiten werden auf der Seite Ereignisse gemeldet:

- Es wird versucht, die Datenbank ohne Ausfallzeiten zu aktualisieren.
- Der Versuch, die Datenbank ohne Ausfallzeiten zu aktualisieren, ist beendet. Die Veranstaltung berichtet, wie lange der Prozess gedauert hat. Das Ereignis meldet auch, wie viele Verbindungen während des Neustarts beibehalten wurden und wie viele Verbindungen gelöscht wurden. Sie können das Datenbankfehlerprotokoll einsehen, um weitere Details darüber zu erfahren, was während des Neustarts passiert ist.

Alternatives Blau/Grün-Upgradeverfahren

In einigen Situationen ist es Ihre oberste Priorität, eine sofortige Umstellung vom alten auf einen aktualisierten Cluster durchzuführen. In diesen Fällen können Sie einen mehrstufigen Prozess verwenden, bei dem die alten und neuen Cluster nebeneinander ausgeführt werden. Hier replizieren Sie Daten vom alten auf den neuen Cluster, bis der neuen Cluster zur Übernahme bereit ist. Details hierzu finden Sie unter Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

Aktualisieren der Hauptversion eines DB-Clusters von Amazon Aurora MySQL

In einer Aurora-MySQL-Versionsnummer wie 3.04.1 steht die 3 für die Hauptversion. Aurora-MySQL-Version 2 ist mit MySQL 5.7 kompatibel. Aurora-MySQL-Version 3 ist mit MySQL 8.0 kompatibel.

Das Upgrade zwischen Hauptversionen erfordert umfangreichere Planung und Tests als für eine Nebenversion. Der Prozess kann beträchtliche Zeit in Anspruch nehmen. Nachdem das Upgrade abgeschlossen ist, müssen Sie möglicherweise auch Follow-up-Arbeiten ausführen. Dies kann beispielsweise aufgrund von Unterschieden in der SQL-Kompatibilität oder der Funktionsweise bestimmter Funktionen im Zusammenhang mit MySQL nötig sein. Oder es kann aufgrund unterschiedlicher Parametereinstellungen zwischen der alten und der neuen Version erforderlich sein.

Inhalt

- Upgrade von Aurora-MySQL-Version 2 auf Version 3
- Aurora MySQL Upgrade-Vorgang für die Hauptversion
- Funktionsweise des Aurora MySQL direkten Upgrade der Hauptversion
- Planen eines Hauptversionsupgrades für einen Aurora MySQL-Cluster
  - Simulieren des Upgrades durch Klonen Ihres DB-Clusters
  - Blau/Grün-Bereitstellungen
- Vorabprüfungen für Hauptversions-Upgrades von Aurora MySQL
  - Vorabprüfungsprozess für Aurora MySQL
  - Vorabprüfungs-Protokollformat für Aurora MySQL
  - Beispiele für Protokollausgaben der Vorabprüfung bei Aurora MySQL
  - Leistung der Vorabprüfung bei Aurora MySQL
  - Zusammenfassung der Vorabprüfungen für Community-MySQL-Upgrades

- Zusammenfassung der Vorabprüfungen für Aurora-MySQL-Upgrades
- Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL
  - Fehler
    - MySQL-Vorabprüfungen, die Fehler melden
    - Aurora-MySQL-Vorabprüfungen, die Fehler melden
  - Warnungen
    - MySQL-Vorabprüfungen, die Warnungen melden
    - Aurora-MySQL-Vorabprüfungen, die Warnungen melden
  - Hinweise
  - Fehler, Warnungen oder Hinweise
- Erläuterung der Durchführung eines direkten Upgrades
  - Wie sich direkte Upgrades auf die Parametergruppen für einen Cluster auswirken
  - Änderungen an Cluster-Eigenschaften zwischen Aurora-MySQL-Versionen
  - In-Situ-Hauptversions-Upgrades für globale Datenbanken
  - Direkte Upgrades für DB-Cluster mit regionsübergreifenden Lesereplikaten
- Aurora MySQL direktes Upgrade
- Finden der Gründe für Fehler bei Hauptversion-Upgrades von Aurora MySQL
- Problembehandlung für Aurora MySQL direkte Upgrades
- Bereinigung nach dem Upgrade für Aurora MySQL Version 3
  - SPATIAL-Index

Upgrade von Aurora-MySQL-Version 2 auf Version 3

Wenn Sie einen mit MySQL 5.7 kompatiblen Cluster auf einen mit MySQL 8.0 kompatiblen Cluster aktualisieren möchten, führen Sie einen Upgrade-Prozess auf dem Cluster selbst aus. Diese Art von Upgrade ist ein direktes Upgrade im Gegensatz zu Upgrades, die Sie durch die Erstellung eines neuen Clusters durchführen. Diese Technik behält den gleichen Endpunkt und andere Eigenschaften des Clusters bei. Das Upgrade ist relativ schnell, da nicht alle Ihre Daten auf ein neues Cluster-Volume kopiert werden müssen. Diese Stabilität hilft, Konfigurationsänderungen in Ihren Anwendungen zu minimieren. Sie trägt auch dazu bei, die Anzahl der Tests für den aktualisierten Cluster zu reduzieren. Das liegt daran, dass die Anzahl der DB-Instances und ihrer Instance-Klassen gleich bleibt.

Der direkte Upgrade-Mechanismus umfasst das Herunterfahren Ihres DB-Clusters, während der Vorgang ausgeführt wird. Aurora führt ein sauberes Herunterfahren durch und schließt ausstehende Vorgänge wie Transaktions-Rollback und Rückgängig-Bereinigung ab. Weitere Informationen finden Sie unter Funktionsweise des Aurora MySQL direkten Upgrade der Hauptversion.

Die direkte Upgrade-Methode ist praktisch, da sie einfach durchzuführen ist und Konfigurationsänderungen an zugehörigen Anwendungen minimiert. Beispielsweise behält ein direktes Upgrade die Endpunkte und die Gruppe von DB-Instances für Ihren Cluster bei. Die für ein direktes Upgrade benötigte Zeit kann jedoch je nach den Eigenschaften Ihres Schemas und der Auslastung des Clusters variieren. Abhängig von den Anforderungen Ihres Clusters können Sie daher zwischen den Upgrade-Methoden wählen:

- Direkt-Upgrade
- Blau/Grün-Bereitstellung
- Snapshot-Wiederherstellung

> **Note**
>
> Wenn Sie die AWS CLI oder die RDS-API für das Snapshot-Wiederherstellungs-Upgrade verwenden, müssen Sie eine nachfolgende Operation ausführen, um eine Writer-DB-Instance im wiederhergestellten DB-Cluster zu erstellen.

Allgemeine Informationen zu Aurora-MySQL-Version 3 und den neuen Funktionen finden Sie unter Aurora mit MySQL Version 3 ist kompatibel mit MySQL 8.0.

Einzelheiten zur Planung eines Upgrades finden Sie unter Planen eines Hauptversionsupgrades für einen Aurora MySQL-Cluster und Erläuterung der Durchführung eines direkten Upgrades.

Aurora MySQL Upgrade-Vorgang für die Hauptversion

Nicht alle Arten oder Versionen von Aurora MySQL Clustern können den integrierten Upgrade-Mechanismus verwenden. Sie können den geeigneten Upgrade-Pfad für jeden Aurora MySQL-Cluster finden, indem Sie die folgende Tabelle konsultieren.

Typ des Aurora MySQL-DB-Clusters	Kann ein direktes Upgrade verwende werden?	Aktion
Von Aurora MySQL bereitgestellter Cluster, Version 2	Ja	Das direkte Upgrade wird für MySQL-5.7-kompatible Aurora-MySQL-Cluster unterstützt. Informationen zum Upgrade auf Aurora-MySQL-Versi on 3 finden Sie unter Planen eines Hauptversionsupgra des für einen Aurora MySQL-Cluster und Erläuterung der Durchführung eines direkten Upgrades.
Von Aurora MySQL bereitgestellter Cluster, Version 3	Nicht zutreffen d	Verwenden Sie ein Upgrade-Verfahren für Nebenvers ionen, um ein Upgrade zwischen Versionen von Aurora-MySQL-Version 3 durchzuführen.
Aurora Serverless v1-Cluster	Nicht zutreffen d	Aurora Serverless v1 wird für Aurora MySQL nur auf Version 2 unterstützt.
Aurora Serverless v2-Cluster	Nicht zutreffen d	Aurora Serverless v2 wird für Aurora MySQL nur auf Version 3 unterstützt.
Cluster in einer Aurora globalen Datenbank	Ja	Wenn Sie Aurora-MySQL-Version 2 zu Version 3 aktualisieren möchten, folgen Sie den Anweisungen für ein direktes Upgrade für Cluster in einer globalen Aurora-Datenbank. Führen Sie das Upgrade auf dem globalen Cluster durch. Aurora aktualisiert den primären Cluster und alle sekundären Cluster in der globalen Datenbank gleichzeitig. Wenn Sie die RDS-API oder AWS CLI verwenden, rufen Sie den `modify-global-cluster` -Befehl oder die

Typ des Aurora MySQL-DB-Clusters	Kann ein direktes Upgrade verwende werden?	Aktion
		`ModifyGlobalCluster` -Operation anstelle von `modify-db-cluster` oder `ModifyDBCluster` auf. Sie können ein direktes Upgrade von Aurora-MySQL-Version 2 auf Version 3 nur durchführen, wenn der `lower_case_table_names` -Parameter auf den Standardwert gesetzt ist und Sie Ihre globale Datenbank neu starten. Weitere Informationen finden Sie unter Hauptversions-Upgrades.
Paralleler Abfrage-Cluster	Ja	Sie können ein direktes Upgrade durchführen.
Cluster, der das Ziel der binären Protokollreplikation ist	Vielleicht	Wenn die binäre Protokollreplikation von einem Aurora-MySQL-Cluster stammt, können Sie ein direktes Upgrade durchführen. Sie können das Upgrade nicht durchführen, wenn die binäre Protokollreplikation von einer RDS-for-MySQL- oder einer On-Premises MySQL-DB-Instance stammt. In diesem Fall können Sie ein Upgrade mit dem Snapshot-Wiederherstellungsmechanismus durchführen.

Typ des Aurora MySQL-DB-Clusters	Kann ein direktes Upgrade verwende werden?	Aktion
Cluster mit Null DB-Instances	Nein	Mit AWS CLI oder RDS-API können Sie einen Aurora-MySQL-Cluster ohne angeschlossene DB-Instances erstellen. Auf die gleiche Weise können Sie auch alle DB-Instances aus einem Aurora MySQL-Cluster entfernen, während die Daten im Cluster-Volume intakt bleiben. Während ein Cluster keine DB-Instances hat, können Sie kein direktes Upgrade durchführen. Der Upgrade-Mechanismus erfordert eine Writer-Instance im Cluster, um Conversions für die Systemtabellen, Datendateien usw. durchzuführen. Verwenden Sie in diesem Fall AWS CLI oder die RDS-API, um eine Writer-Instance für den Cluster zu erstellen. Dann können Sie ein direktes Upgrade durchführen.
Cluster mit aktivierter Rückverfolgung	Ja	Sie können ein direktes Upgrade für einen Aurora-MySQL-Cluster durchführen, der die Rückverfolgungsfunktion verwendet. Nach dem Upgrade können Sie den Cluster jedoch nicht auf einen Zeitpunkt vor dem Upgrade zurückverfolgen.

Funktionsweise des Aurora MySQL direkten Upgrade der Hauptversion

Aurora MySQL führt das Upgrade einer Hauptversion als mehrstufigen Prozess durch. Sie können den aktuellen Status eines Upgrades überprüfen. Einige der Upgrade-Schritte enthalten auch Fortschrittsinformationen. Beim Start jeder Phase, zeichnet Aurora MySQL ein Ereignis auf. Sie können Ereignisse so untersuchen, wenn sie auf der Seite Ereignisse in der RDS-Konsole auftreten. Weitere Informationen zur Arbeit mit -Ereignissen finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

> **Important**
>
> Sobald der Prozess beginnt, wird er ausgeführt, bis das Upgrade erfolgreich ist oder fehlschlägt. Sie können das Upgrade nicht abbrechen, während es läuft. Wenn das Upgrade fehlschlägt, setzt Aurora alle Änderungen zurück und Ihr Cluster hat die gleiche Engine-Version, Metadaten usw. wie zuvor.

Der Upgrade-Prozess besteht aus drei Schritten:

1. Aurora führt vor Beginn des Upgrade-Prozesses eine Reihe von Vorabprüfungen durch. Ihr Cluster läuft weiter, während Aurora diese Prüfungen durchführt. Zum Beispiel kann der Cluster keine XA-Transaktionen im vorbereiteten Zustand haben oder irgendwelche DDL-Anweisungen (Data Definition Language) verarbeiten. Beispielsweise müssen Sie möglicherweise Anwendungen herunterfahren, die bestimmte Arten von SQL-Anweisungen einreichen. Oder Sie könnten einfach warten, bis bestimmte lang andauernde Anweisungen beendet sind. Versuchen Sie dann erneut das Upgrade durchzuführen. Einige Prüfungen testen auf Bedingungen, die das Upgrade nicht verhindern, aber dazu führen können, dass das Upgrade möglicherweise länger dauert.

   Wenn Aurora feststellt, dass die erforderlichen Bedingungen nicht erfüllt sind, ändern Sie die in den Ereignisdetails angegebenen Bedingungen. Befolgen Sie die Anweisungen unter Problembehandlung für Aurora MySQL direkte Upgrades. Wenn Aurora Bedingungen erkennt, die ein langsames Upgrade verursachen könnten, planen Sie ein, das Upgrade über einen längeren Zeitraum zu überwachen.
2. Aurora nimmt Ihren Cluster offline. Aurora führt dann ähnliche Tests wie in der vorherigen Phase durch, um zu bestätigen, dass während des Shutdown-Vorgangs keine neuen Probleme aufgetreten sind. Wenn Aurora zu diesem Zeitpunkt Bedingungen erkennt, die das Upgrade verhindern würden, bricht Aurora das Upgrade ab und bringt den Cluster wieder online. Bestätigen Sie in diesem Fall, wenn die Bedingungen nicht mehr gelten, und starten Sie das Upgrade erneut.
3. Aurora erstellt einen Snapshot Ihres Cluster-Volumes. Stellen Sie sich vor, Sie stellen nach Abschluss des Upgrades Kompatibilitäts- oder andere Probleme fest. Oder angenommen, Sie möchten Tests sowohl mit den ursprünglichen als auch mit den aktualisierten Clustern durchführen. In solchen Fällen können Sie aus diesem Snapshot den Cluster wiederherstellen, um einen neuen Cluster mit der ursprünglichen Engine-Version und den ursprünglichen Daten zu erstellen.

> **Tip**
>
> Dieser Snapshot ist ein manueller Snapshot. Aurora kann den Snapshot jedoch erstellen und den Upgrade-Prozess fortsetzen, auch wenn Sie Ihr Limit für manuelle Snapshots erreicht haben. Dieser Snapshot verbleibt dauerhaft (falls erforderlich), bis Sie ihn löschen. Nachdem Sie alle Tests nach dem Upgrade abgeschlossen haben, können Sie diesen Snapshot löschen, um die Speicherkosten zu minimieren.

4. Aurora klont Ihr Cluster-Volume. Das Klonen ist ein schneller Vorgang, bei dem die tatsächlichen Tabellendaten nicht kopiert werden müssen. Wenn Aurora während des Upgrades ein Problem feststellt, werden die Originaldaten des geklonten Cluster-Volumes zurückgesetzt und der Cluster wieder online gebracht. Das temporär geklonte Volume während des Upgrades unterliegt nicht dem üblichen Limit für die Anzahl der Klone für ein einzelnes Cluster-Volume.
5. Aurora führt ein sauberes Herunterfahren für die Writer-DB-Instance durch. Während des sauberen Herunterfahrens werden Fortschrittsereignisse alle 15 Minuten für die folgenden Vorgänge aufgezeichnet. Sie können Ereignisse so untersuchen, wenn sie auf der Seite Ereignisse in der RDS-Konsole auftreten.
   - Aurora bereinigt die Undo-Datensätze für alte Versionen von Zeilen.
   - Aurora setzt alle nicht festgeschriebenen Transaktionen zurück.
6. Aurora aktualisiert die Engine-Version auf der Writer-DB-Instance:
   - Aurora installiert die Binärdatei für die neue Engine-Version auf der Writer-DB-Instance.
   - Aurora verwendet die Writer-DB-Instance, um Ihre Daten auf das mit MySQL 5.7-kompatible Format zu aktualisieren. In dieser Phase ändert Aurora die Systemtabellen und führt andere Konvertierungen durch, die sich auf die Daten in Ihrem Cluster-Volume auswirken. Insbesondere Aurora aktualisiert die Partitions-Metadaten in den Systemtabellen, um mit dem Partitionsformat MySQL 5.7 kompatibel zu sein. Diese Phase kann lange dauern, wenn die Tabellen in Ihrem Cluster eine große Anzahl von Partitionen haben.

     Wenn während dieser Phase Fehler auftreten, finden Sie die Details in den MySQL-Fehlerprotokollen. Wenn diese Phase nach dem Start der Upgradevorgang aus irgendeinem Grund fehlschlägt, stellt Aurora die Originaldaten aus dem geklonten Cluster-Volume wieder her.
7. Aurora aktualisiert die Engine-Version auf den Reader-DB-Instances.
8. Der Upgrade-Vorgang ist abgeschlossen. Aurora zeichnet ein letztes Ereignis auf, um anzuzeigen, dass der Upgrade-Prozess erfolgreich abgeschlossen wurde. Jetzt läuft Ihr DB-Cluster mit der neuen Hauptversion.

Planen eines Hauptversionsupgrades für einen Aurora MySQL-Cluster

Um Ihnen bei der Entscheidung für den richtigen Zeitpunkt und den richtigen Ansatz für das Upgrade zu helfen, können Sie die Unterschiede zwischen Aurora-MySQL-Version 3 und Ihrer aktuellen Umgebung kennenlernen:

- Wenn Sie von RDS für MySQL 8.0 oder MySQL 8.0 Community Edition konvertieren, lesen Sie bitte Vergleich von Aurora-MySQL-Version 3 und MySQL 8.0 Community Edition.
- Wenn Sie ein Upgrade von Aurora-MySQL-Version 2, RDS für MySQL 5.7 oder Community-MySQL 5.7 durchführen, vgl. Vergleich von Aurora-MySQL-Version 2 und Aurora-MySQL-Version 3.
- Erstellen Sie neue MySQL 8.0-kompatible Versionen beliebiger benutzerdefinierter Parametergruppen. Wenden Sie alle erforderlichen benutzerdefinierten Parameterwerte auf die neuen Parametergruppen an. Konsultieren Sie Parameteränderungen für Aurora MySQL Version 3, um mehr über Parameteränderungen zu erfahren.
- Überprüfen Sie das Datenbankschema und die Objektdefinitionen für Aurora-MySQL-Version 2 auf die Verwendung neuer reservierter Schlüsselwörter, die in der MySQL 8.0 Community Edition eingeführt wurden. Führen Sie diesen Schritt vor dem Upgrade aus. Weitere Informationen finden Sie unter MySQL 8.0 Neue Schlüsselwörter und reservierte Wörter in der MySQL-Dokumentation.

Weitere MySQL-Spezifische Upgrades und Tipps finden Sie auch unterÄnderungen in MySQL 8.0imMySQL-Referenzhandbuchaus. Sie können beispielsweise den Befehl `mysqlcheck --check-upgrade` verwenden, um Ihre bestehenden Aurora-MySQL-Datenbanken zu analysieren und potenzielle Upgrade-Probleme zu identifizieren.

> **Note**
>
> Wir empfehlen die Verwendung größerer DB-Instance-Klassen beim Upgrade auf Aurora-MySQL-Version 3 mit dem direkten Upgrade oder der Snapshot-Wiederherstellungsmethode. Beispiele sind db.r5.24xlarge und db.r6g.16xlarge. Dadurch kann der Upgrade-Prozess schneller abgeschlossen werden, da der Großteil der verfügbaren CPU-Kapazität auf der DB-Instance genutzt wird. Sie können zu der gewünschten DB-Instance-Klasse wechseln, nachdem das Upgrade der Hauptversion abgeschlossen ist.

Nachdem Sie das Upgrade selbst abgeschlossen haben, können Sie die Verfahren nach dem Upgrade unter Bereinigung nach dem Upgrade für Aurora MySQL Version 3 befolgen. Testen Sie abschließend die Funktionalität und Leistung Ihrer Anwendung.

Wenn Sie von RDS aus MySQL oder Community-MySQL konvertieren, befolgen Sie das unter Migrieren von Daten zu einem Amazon Aurora MySQL-DB-Cluster beschriebene Verfahren. In einigen Fällen können Sie die Binärprotokollreplikation verwenden, um Ihre Daten im Rahmen der Migration mit einem Aurora MySQL-Cluster der Version 3 zu synchronisieren. In diesem Fall muss das Quellsystem eine Version ausführen, die mit Ihrem Ziel-DB-Cluster kompatibel ist.

Damit sichergestellt wird, dass Ihre Anwendungen und Verwaltungsverfahren nach dem Upgrade eines Clusters zwischen Hauptversionen reibungslos funktionieren, führen Sie einige Vorausplanungen und Vorbereitungen durch. Informationen darüber, welche Arten von Verwaltungscodes für Ihre AWS CLI-Skripts oder RDS-API-basierten Anwendungen aktualisiert werden müssen, finden Sie unter Wie sich direkte Upgrades auf die Parametergruppen für einen Cluster auswirken. Lesen Sie auch Änderungen an Cluster-Eigenschaften zwischen Aurora-MySQL-Versionen.

Informationen zu Problemen, auf die Sie während des Upgrades stoßen können, finden Sie unter Problembehandlung für Aurora MySQL direkte Upgrades. Bei Problemen, die dazu führen können, dass das Upgrade sehr lange dauert, können Sie diese Bedingungen im Voraus testen und korrigieren.

> **Note**
>
> Ein direktes Upgrade umfasst das Herunterfahren Ihres DB-Clusters, während der Vorgang ausgeführt wird. Aurora MySQL führt ein sauberes Herunterfahren durch und schließt ausstehende Vorgänge wie Undo-Bereinigungen ab. Ein Upgrade kann lange dauern, wenn viele Undo-Datensätze bereinigt werden müssen. Wir empfehlen, das Upgrade erst durchzuführen, wenn die Länge der Verlaufsliste (History List Length, HLL) niedrig ist. Ein allgemein akzeptabler Wert für die HLL ist 100 000 oder weniger. Weitere Informationen finden Sie in diesem Blog-Beitrag.

Simulieren des Upgrades durch Klonen Ihres DB-Clusters

Sie können die Anwendungskompatibilität, die Leistung, die Wartungsverfahren und ähnliche Überlegungen für den aktualisierten Cluster überprüfen. Zu diesem Zweck können Sie vor dem eigentlichen Upgrade eine Simulation des Upgrades durchführen. Diese Technik kann besonders

für Produktionscluster nützlich sein. Hier ist es wichtig, Ausfallzeiten zu minimieren und den aktualisierten Cluster einsatzbereit zu haben, sobald das Upgrade abgeschlossen ist.

Gehen Sie dazu wie folgt vor:

1. Erstellen Sie einen Klon des ursprünglichen Clusters. Folgen Sie dem Verfahren unter Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster.
2. Richten Sie einen ähnlichen Satz von Writer- und Reader-DB-Instances wie im ursprünglichen Cluster ein.
3. Führen Sie ein direktes Upgrade des geklonten Clusters durch. Folgen Sie dem Verfahren unter Erläuterung der Durchführung eines direkten Upgrades.

   Starten Sie das Upgrade sofort nach dem Erstellen des Klons. Auf diese Weise ist das Cluster-Volume immer noch identisch mit dem Zustand des ursprünglichen Clusters. Wenn der Klon vor dem Upgrade im Leerlauf ist, führt Aurora Datenbankbereinigungsprozesse im Hintergrund durch. In diesem Fall ist das Upgrade des Klons keine genaue Simulation für das Upgrade des ursprünglichen Clusters.
4. Sie können die Anwendungskompatibilität, Performance, Administrationsprozeduren usw. mit dem geklonten Cluster testen.
5. Wenn Probleme auftreten, passen Sie Ihre Upgrade-Pläne an, um sie zu beheben. Passen Sie beispielsweise jeden Anwendungscode so an, dass er mit dem Funktionsumfang der höheren Version kompatibel ist. Sie können abschätzen, wie lange das Upgrade wahrscheinlich auf der Grundlage der Datenmenge in Ihrem Cluster dauern wird. Sie können das Upgrade auch für eine Zeit planen, in der der Cluster nicht ausgelastet ist.
6. Nachdem Sie sich vergewissert haben, dass Ihre Anwendungen und Workload mit dem Testcluster ordnungsgemäß funktionieren, können Sie das direkte Upgrade für Ihren Produktionscluster durchführen.
7. Minimieren Sie möglichst die Gesamtausfallzeit Ihres Clusters während des Upgrades einer Hauptversion. Stellen Sie dazu sicher, dass die Workload auf dem Cluster zum Zeitpunkt des Upgrades niedrig oder Null ist. Stellen Sie insbesondere sicher, dass beim Start des Upgrades keine länger laufenden Transaktionen durchgeführt werden.

Blau/Grün-Bereitstellungen

In einigen Situationen ist es Ihre oberste Priorität, eine sofortige Umstellung vom alten auf einen aktualisierten Cluster durchzuführen. In diesen Fällen können Sie einen mehrstufigen Prozess

verwenden, bei dem die alten und neuen Cluster nebeneinander ausgeführt werden. Hier replizieren Sie Daten vom alten auf den neuen Cluster, bis der neuen Cluster zur Übernahme bereit ist. Details hierzu finden Sie unter Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

Vorabprüfungen für Hauptversions-Upgrades von Aurora MySQL

Ein Upgrade von MySQL von einer Hauptversion auf eine andere, wie z. B. die Umstellung von MySQL 5.7 auf MySQL 8.0, beinhaltet einige wichtige architektonische Änderungen, die eine sorgfältige Planung und Vorbereitung erfordern. Im Gegensatz zu Nebenversions-Upgrades, bei denen der Schwerpunkt hauptsächlich auf der Aktualisierung der Datenbank-Engine-Software und in einigen Fällen der Systemtabellen liegt, führen Hauptversions-Upgrades von MySQL häufig zu grundlegenden Änderungen daran, wie die Datenbank ihre Metadaten speichert und verwaltet.

Zur Unterstützung bei der Identifizierung solcher Inkompatibilitäten führt Aurora beim Upgrade von Aurora-MySQL-Version 2 auf Version 3 automatisch Upgrade-Kompatibilitätsprüfungen (Vorabprüfungen) durch. Dabei werden Objekte in Ihrem Datenbank-Cluster untersucht und bekannte Inkompatibilitäten identifiziert, die das Upgrade blockieren können. Weitere Informationen zu Aurora-MySQL-Vorabprüfungen finden Sie unter Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL. Die Aurora-Vorabprüfungen werden zusätzlich zu denen ausgeführt, die vom Upgrade-Checker-Dienstprogramm von Community MySQL durchgeführt werden.

Diese Vorabprüfungen müssen durchgeführt werden. Sie können nicht ausgelassen werden. Die Vorabprüfungen bieten folgende Vorteile:

- Sie können die Wahrscheinlichkeit verringern, dass Upgrade-Fehler auftreten, die zu längeren Ausfallzeiten führen können.
- Wenn es Inkompatibilitäten gibt, verhindert Amazon Aurora, dass das Upgrade fortgesetzt wird, und stellt Ihnen ein Protokoll mit Informationen zu den Inkompatibilitäten bereit. Sie können das Protokoll für die Vorbereitung Ihrer Datenbank auf das Upgrade auf Version 3 verwenden, indem Sie die Inkompatibilitäten beheben. Detaillierte Informationen zum Beheben von Inkompatibilitäten finden Sie unter Vorbereiten Ihrer Installation auf ein Upgrade in der MySQL-Dokumentation und unter Upgrade auf MySQL 8.0?. Dies müssen Sie wissen ... im MySQL Server Blog.

  Weitere Informationen zum Ausführen von Upgrades auf MySQL 8.0 finden Sie unter Ausführen von MySQL-Upgrades in der MySQL-Dokumentation.

Die Vorabprüfungen werden ausgeführt, bevor Ihr DB-Cluster für das Hauptversions-Upgrade offline geschaltet wird. Wird während der Vorabprüfungen eine Inkompatibilität entdeckt, bricht Aurora automatisch das Upgrade ab, ehe die DB-Instance angehalten wird. Aurora generiert auch ein Ereignis für die Inkompatibilität. Weitere Informationen über Amazon-Aurora-Ereignisse finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

Nach Abschluss der Vorabprüfungen zeichnet Aurora detaillierte Informationen zu allen Inkompatibilitäten in der Datei `upgrade-prechecks.log` auf. In den meisten Fällen enthalten die Protokolleinträge einen Link zur MySQL-Dokumentation mit Informationen zur Lösung des Inkompatibilitätsproblems. Weitere Informationen zum Anzeigen von Protokolldateien finden Sie unter Anzeigen und Auflisten von Datenbank-Protokolldateien.

> **Note**
>
> Aufgrund der Art der Vorabprüfungen werden die Objekte in Ihrer Datenbank geprüft. Diese Analyse verbraucht Ressourcen und verlängert die Zeit, die für die Durchführung des Upgrades benötigt wird. Weitere Informationen zu Leistungsaspekten bei der Vorabprüfung finden Sie unter Vorabprüfungsprozess für Aurora MySQL.

Inhalt

- Vorabprüfungsprozess für Aurora MySQL
- Vorabprüfungs-Protokollformat für Aurora MySQL
- Beispiele für Protokollausgaben der Vorabprüfung bei Aurora MySQL
- Leistung der Vorabprüfung bei Aurora MySQL
- Zusammenfassung der Vorabprüfungen für Community-MySQL-Upgrades
- Zusammenfassung der Vorabprüfungen für Aurora-MySQL-Upgrades
- Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL
  - Fehler
    - MySQL-Vorabprüfungen, die Fehler melden
    - Aurora-MySQL-Vorabprüfungen, die Fehler melden
  - Warnungen
    - MySQL-Vorabprüfungen, die Warnungen melden
    - Aurora-MySQL-Vorabprüfungen, die Warnungen melden
  - Hinweise
  - Fehler, Warnungen oder Hinweise

Vorabprüfungsprozess für Aurora MySQL

Wie bereits beschrieben, beinhaltet der Upgrade-Prozess von Aurora MySQL die Durchführung von Kompatibilitätsprüfungen (Vorabprüfungen) für Ihre Datenbank, bevor das Hauptversions-Upgrade fortgesetzt werden kann.

Bei direkten Upgrades werden die Vorabprüfungen auf Ihrer Schreiber-DB-Instance ausgeführt, während diese online ist. Wenn die Vorabprüfung erfolgreich ist, wird das Upgrade fortgesetzt. Wenn Fehler gefunden werden, werden sie in der Datei `upgrade-prechecks.log` protokolliert und das Upgrade wird abgebrochen. Bevor Sie das Upgrade erneut versuchen, beheben Sie alle in der Datei `upgrade-prechecks.log` zurückgegebenen Fehler.

Bei Snapshot-Wiederherstellungs-Upgrades wird die Vorabprüfung während des Wiederherstellungsprozesses ausgeführt. Wenn diese erfolgreich ist, wird Ihre Datenbank auf die neue Aurora-MySQL-Version aktualisiert. Wenn Fehler gefunden werden, werden sie in der Datei `upgrade-prechecks.log` protokolliert und das Upgrade wird abgebrochen. Bevor Sie das Upgrade erneut versuchen, beheben Sie alle in der Datei `upgrade-prechecks.log` zurückgegebenen Fehler.

Weitere Informationen erhalten Sie unter Finden der Gründe für Fehler bei Hauptversion-Upgrades von Aurora MySQL und Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL.

Wenn Sie den Vorabprüfungsstatus überwachen möchten, können Sie die folgenden Ereignisse in Ihrem DB-Cluster anzeigen.

Vorabprüfungsstatus	Ereignismeldung	Aktion
Gestartet	Upgrade-Vorbereitung im Gange: Die Online-Upgrade-Vorabprüfungen werden gestartet.	Keine
Fehlgeschlagen	Der Datenbank-Cluster befindet sich in einem Zustand, der nicht aktualisiert werden kann. Upgrade-Vorabprüfungen sind fehlgeschlagen. Weitere Informati	Überprüfen Sie `upgrade-prechecks.log` auf Fehler. Beheben Sie die Fehler. Versuchen Sie das Upgrade erneut.

Vorabprüfungsstatus	Ereignismeldung	Aktion
	onen finden Sie in der Datei upgrade-prechecks.log. Weitere Informationen zur Behebung der Ursache des Upgrade-Fehlers finden Sie unter https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Upgrading.Troubleshooting.html	
Erfolgreich	Upgrade-Vorbereitung im Gange: Die Online-Upgrade-Vorabprüfungen wurden abgeschlossen.	Die Vorabprüfung wurde erfolgreich abgeschlossen. Es wurden keine Fehler erkannt. Überprüfen Sie `upgrade-prechecks.log` auf Warnungen und Hinweise.

Weitere Informationen zum Anzeigen von Ereignissen finden Sie unter Anzeigen von Amazon-RDS-Ereignissen.

Vorabprüfungs-Protokollformat für Aurora MySQL

Nachdem die Upgrade-Kompatibilitätsprüfungen (Vorabprüfungen) abgeschlossen sind, können Sie die Datei `upgrade-prechecks.log` überprüfen. Die Protokolldatei enthält die Ergebnisse, die betroffenen Objekte und Informationen zur Behebung für jede Vorabprüfung.

Fehler blockieren das Upgrade. Sie müssen sie beheben, bevor Sie das Upgrade erneut versuchen können.

Warnungen und Hinweise sind weniger wichtig, aber wir empfehlen dennoch, sie sorgfältig zu überprüfen, um sicherzustellen, dass keine Kompatibilitätsprobleme mit dem Anwendungs-Workload vorliegen. Beheben Sie erkannte Probleme zeitnah.

Die Protokolldatei hat folgendes Format:

- `targetVersion` – Die MySQL-kompatible Version des Aurora-MySQL-Upgrades
- `auroraServerVersion` – Die Aurora-MySQL-Version, auf der die Vorabprüfung ausgeführt wurde
- `auroraTargetVersion` – Die Aurora-MySQL-Version, auf die Sie aktualisieren
- `checksPerformed` – Enthält die Liste der durchgeführten Vorabprüfungen
- `id` – Der Name der gerade ausgeführten Vorabprüfung
- `title` – Eine Beschreibung der gerade ausgeführten Vorabprüfung
- `status` – Dies gibt nicht an, ob die Vorabprüfung erfolgreich war oder nicht, sondern zeigt den Status der Abfrage zur Vorabprüfung an.
  - `OK` – Die Abfrage zur Vorabprüfung wurde erfolgreich ausgeführt und abgeschlossen.
  - `ERROR` – Die Abfrage zur Vorabprüfung konnte nicht ausgeführt werden. Dies kann auf Probleme wie Ressourcenbeschränkungen, unerwartete Instance-Neustarts oder die Unterbrechung der Abfrage zur Vorabprüfung der Kompatibilität zurückzuführen sein.

    Weitere Informationen finden Sie in diesem Beispiel.
- `description` – Eine allgemeine Beschreibung der Inkompatibilität und wie das Problem behoben werden kann
- `documentationLink` – Gegebenenfalls finden Sie hier einen Link zur entsprechenden Aurora-MySQL- oder MySQL-Dokumentation. Weitere Informationen finden Sie unter Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL.
- `detectedProblems` – Wenn bei der Vorabprüfung ein Fehler, eine Warnung oder ein Hinweis zurückgegeben wird, werden hier Details zur Inkompatibilität und gegebenenfalls inkompatible Objekte angezeigt.
  - `level` – Die bei der Vorabprüfung festgestellte Stufe der Inkompatibilität. Folgende Stufen sind gültig:
    - `Error` – Das Upgrade kann erst fortgesetzt werden, wenn Sie die Inkompatibilität behoben haben
    - `Warning` – Das Upgrade kann fortgesetzt werden, es wurde jedoch ein veraltetes Objekt, eine veraltete Syntax oder Konfiguration erkannt. Lesen Sie die Warnungen sorgfältig und beheben Sie sie zeitnah, um Probleme in künftigen Versionen zu vermeiden.

    - `Notice` – Das Upgrade kann fortgesetzt werden, es wurde jedoch ein veraltetes Objekt, eine veraltete Syntax oder Konfiguration erkannt. Lesen Sie die Hinweise sorgfältig und beheben Sie sie zeitnah, um Probleme in künftigen Versionen zu vermeiden.
  - `dbObject` – Der Name des Datenbankobjekts, in dem die Inkompatibilität festgestellt wurde
  - `description` – Eine detaillierte Beschreibung der Inkompatibilität und wie das Problem behoben werden kann
- `errorCount` – Die Anzahl der erkannten Inkompatibilitätsfehler. Diese blockieren das Upgrade.
- `warningCount` – Die Anzahl der erkannten Inkompatibilitätswarnungen. Diese blockieren das Upgrade nicht, beheben Sie sie jedoch zeitnah, um Probleme in künftigen Versionen zu vermeiden.
- `noticeCount` – Die Anzahl der erkannten Inkompatibilitätshinweise. Diese verhindern das Upgrade nicht, sollten jedoch zeitnah behoben werden, um Probleme in künftigen Versionen zu vermeiden.
- `Summary` – Eine Zusammenfassung der Anzahl von Kompatibilitätsfehlern, -warnungen und -hinweisen aus der Vorabprüfung

Beispiele für Protokollausgaben der Vorabprüfung bei Aurora MySQL

Die folgenden Beispiele zeigen die Protokollausgaben der Vorabprüfung, die Sie möglicherweise angezeigt bekommen. Weitere Informationen zu den ausgeführten Vorabprüfungen finden Sie unter Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL.

Vorabprüfungsstatus OK, keine Inkompatibilität erkannt

Die Abfrage zur Vorabprüfung wurde erfolgreich abgeschlossen. Es wurden keine Inkompatibilitäten erkannt.

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinytext",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny text columns",
  "status": "OK",
  "detectedProblems": []
},
```

Vorabprüfungsstatus OK, Fehler erkannt

Die Abfrage zur Vorabprüfung wurde erfolgreich abgeschlossen. Es wurde ein Fehler erkannt.

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexes",
  "status": "OK",
  "description": "Consider dropping the prefix indexes of geometry columns and
 restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test25.sbtest1",
        "description": "Table `test25`.`sbtest1` has an index `idx_t1` on geometry
 column/s. Mysql 8.0 does not support this type of index on a geometry column
 https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run 'DROP
 INDEX `idx_t1` ON `test25`.`sbtest1`;"
      },
 }
```

Vorabprüfungsstatus OK, Warnung erkannt

Warnungen können zurückgegeben werden, wenn eine Vorabprüfung erfolgreich oder nicht erfolgreich war.

Hier wurde die Abfrage zur Vorabprüfung erfolgreich abgeschlossen. Es wurden zwei Warnungen erkannt.

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
```

```
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
    ]
}
```

Vorabprüfungsstatus ERROR, keine Inkompatibilitäten gemeldet

Die Abfrage zur Vorabprüfung schlug mit einem Fehler fehl, sodass Inkompatibilitäten nicht verifiziert werden konnten.

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "ERROR",
  "description": "Can't connect to MySQL server on 'localhost:3306' (111) at
 13/08/2024 12:22:20 UTC. This failure can occur due to low memory available on the
 instance for executing upgrade prechecks. Please check 'FreeableMemory' Cloudwatch
 metric to verify the available memory on the instance while executing prechecks. If
 instance ran out of memory, we recommend to retry the upgrade on a higher instance
 class."
}
```

Dieser Fehler kann auftreten, wenn die Instance unerwartet neu gestartet wurde oder eine Abfrage zur Vorabprüfung der Kompatibilität in der Datenbank während der Ausführung unterbrochen wurde. Bei kleineren DB-Instance-Klassen kann dies beispielsweise auftreten, wenn der verfügbare Speicher auf der Instance knapp wird.

Sie können die Amazon-CloudWatch-Metrik `FreeableMemory` verwenden, um den verfügbaren Arbeitsspeicher auf der Instance zu überprüfen, während Sie Vorabprüfungen ausführen. Wenn der Instance nicht mehr genügend Arbeitsspeicher zur Verfügung steht, empfehlen wir, das Upgrade auf einer größeren DB-Instance-Klasse erneut durchzuführen. In einigen Fällen können Sie eine Blau/Grün-Bereitstellung verwenden. Auf diese Weise können Vorabprüfungen und Upgrades im „grünen“ DB-Cluster unabhängig vom Produktions-Workload ausgeführt werden, wodurch auch Systemressourcen verbraucht werden.

Weitere Informationen finden Sie unter Beheben von Problemen mit der Speichernutzung bei Aurora-MySQL-Datenbanken.

Zusammenfassung der Vorabprüfung: Ein Fehler und drei Warnungen erkannt

Die Vorabprüfungen der Kompatibilität enthalten auch Informationen zu den Quell- und Zielversionen von Aurora MySQL sowie eine Zusammenfassung der Anzahl von Fehlern, Warnungen und Hinweisen am Ende der Ausgabe der Vorabprüfung.

Die folgende Ausgabe zeigt beispielsweise, dass versucht wurde, ein Upgrade von Aurora MySQL 2.11.6 auf Aurora MySQL 3.07.1 durchzuführen. Das Upgrade hat einen Fehler, drei Warnungen und keine Hinweise zurückgegeben. Da Upgrades nicht fortgesetzt werden können, wenn ein Fehler zurückgegeben wird, müssen Sie das Kompatibilitätsproblem routineSyntaxCheck beheben und das Upgrade erneut versuchen.

```
{
  "serverAddress": "/tmp%2Fmysql.sock",
  "serverVersion": "5.7.12 - MySQL Community Server (GPL)",
  "targetVersion": "8.0.36",
  "auroraServerVersion": "2.11.6",
  "auroraTargetVersion": "3.07.1",
  "outfilePath": "/rdsdbdata/tmp/PreChecker.log",
  "checksPerformed": [{
      ... output for each individual precheck ...
      .
      .
      {
        "id": "oldTemporalCheck",
        "title": "Usage of old temporal type",
        "status": "OK",
          "detectedProblems": []
      },
      {
        "id": "routinesSyntaxCheck",
        "title": "MySQL 8.0 syntax check for routine-like objects",
        "status": "OK",
        "description": "The following objects did not pass a syntax check with
 the latest MySQL 8.0 grammar. A common reason is that they reference names that
 conflict with new reserved keywords. You must update these routine definitions and
 `quote` any such references before upgrading.",
        "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
        "detectedProblems": [{
            "level": "Error",
```

```
            "dbObject": "test.select_res_word",
            "description": "at line 2,18: unexpected token 'except'"
        }]
      },
      .
      .
      .
      {
        "id": "zeroDatesCheck",
        "title": "Zero Date, Datetime, and Timestamp values",
        "status": "OK",
        "description": "Warning: By default zero date/datetime/timestamp values
 are no longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are
 included in SQL_MODE by default. These modes should be used with strict mode as
 they will be merged with strict mode in a future release. If you do not include
 these modes in your SQL_MODE setting, you are able to insert date/datetime/
timestamp values that contain zeros. It is strongly advised to replace zero values
 with valid ones, as they may not work correctly in the future.",
        "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
        "detectedProblems": [{
            "level": "Warning",
            "dbObject": "global.sql_mode",
            "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
            },
            {
            "level": "Warning",
            "dbObject": "session.sql_mode",
            "description": " of 8 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
            }
          ]
       },
        .
        .
        .
  }],
  "errorCount": 1,
  "warningCount": 3,
  "noticeCount": 0,
  "Summary": "1 errors were found. Please correct these issues before upgrading to
 avoid compatibility issues."
}
```

Leistung der Vorabprüfung bei Aurora MySQL

Die Vorabprüfungen der Kompatibilität erfolgen vor der Offline-Schaltung der DB-Instance für das Upgrade und führen unter normalen Umständen zu keiner Ausfallzeit. Sie können jedoch Auswirkungen auf den Anwendungs-Workload haben, der auf der Schreiber-DB-Instance ausgeführt wird. Die Vorabprüfungen greifen über information_schema-Tabellen auf das Datenwörterbuch zu, was bei vielen Datenbankobjekten langsam sein kann. Berücksichtigen Sie folgende Faktoren:

- Die Dauer der Vorabprüfung hängt von der Anzahl der Datenbankobjekte wie Tabellen, Spalten, Routinen und Einschränkungen ab. Die Ausführung von DB-Clustern mit einer großen Anzahl von Objekten kann länger dauern.

  Beispielsweise kann der removedFunctionsCheck länger dauern und je nach Anzahl der gespeicherten Objekte mehr Ressourcen beanspruchen.
- Bei direkten Upgrades kann die Verwendung einer größeren DB-Instance-Klasse (z. B. db.r5.24xlarge oder db.r6g.16xlarge) dazu beitragen, dass das Upgrade schneller abgeschlossen wird, da mehr CPU verwendet wird. Eine Verkleinerung ist nach dem Upgrade möglich.
- Abfragen zum `information_schema` über mehrere Datenbanken hinweg können langsam sein, insbesondere bei vielen Objekten und auf kleineren DB-Instances. In solchen Fällen sollten Sie die Verwendung von Klonen, Snapshot-Wiederherstellungen oder einer Blau/Grün-Bereitstellung für Upgrades in Betracht ziehen.
- Die Vorabprüfungs-Ressourcennutzung (CPU, Arbeitsspeicher) kann mit mehr Objekten zunehmen, was zu längeren Laufzeiten auf kleineren DB-Instances führt. In solchen Fällen sollten Sie erwägen, Tests mithilfe von Klonen, Snapshot-Wiederherstellung oder einer Blau/Grün-Bereitstellung für Upgrades durchzuführen.

  Wenn die Vorabprüfungen aufgrund fehlender Ressourcen fehlschlagen, können Sie dies anhand der Statusausgabe im Vorabprüfungsprotokoll erkennen:

```
"status": "ERROR",
```

Weitere Informationen erhalten Sie unter Funktionsweise des Aurora MySQL direkten Upgrade der Hauptversion und Planen eines Hauptversionsupgrades für einen Aurora MySQL-Cluster.

Zusammenfassung der Vorabprüfungen für Community-MySQL-Upgrades

Im Folgenden finden Sie eine allgemeine Liste der Inkompatibilitäten zwischen MySQL 5.7 und 8.0:

- Ihr MySQL-5.7-kompatibler DB-Cluster darf keine Funktionen verwenden, die in MySQL 8.0 nicht unterstützt werden.

  Weitere Informationen finden Sie unter In MySQL 8.0 entfernte Funktionen in der MySQL-Dokumentation.
- Es darf keine Verletzungen von Schlüsselwörtern oder reservierten Wörtern geben. Einige Schlüsselwörter sind in MySQL 8.0 möglicherweise reserviert, die zuvor nicht reserviert waren.

  Weitere Informationen finden Sie unter Schlüsselwörter und reservierte Wörter in der MySQL-Dokumentation.
- Um die Unicode-Unterstützung zu verbessern, sollten Sie die Konvertierung von Objekten, die den `utf8mb3`-Zeichensatz verwenden, in Objekte in Betracht ziehen, die den `utf8mb4`-Zeichensatz verwenden. Der `utf8mb3`-Zeichensatz ist veraltet. Sie sollten darüber hinaus anstelle von `utf8mb4` die Verwendung von `utf8` für Zeichensatzverweise in Betracht ziehen, da `utf8` zurzeit ein Alias für den `utf8mb3`-Zeichensatz ist.

  Weitere Informationen finden Sie unter Der utf8mb3-Zeichensatz (UTF-8-Unicode-Kodierung mit 3 Bytes) in der MySQL-Dokumentation.
- Es darf keine InnoDB-Tabellen mit einem nicht standardmäßigen Zeilenformat geben.
- Es darf keine `ZEROFILL`- oder `display`-Längentyp-Attribute geben.
- Es darf keine partitionierte Tabelle mit einer Speicher-Engine geben, für die es keine native Partitionierungsunterstützung gibt.
- Es darf keine Tabellen in der MySQL 5.7 `mysql`-Systemdatenbank geben, die denselben Namen wie eine Tabelle haben, die vom MySQL 8.0-Daten-Dictionary verwendet wird.
- Es darf keine Tabellen geben, die veraltete Datentypen oder Funktionen verwenden.
- Es darf keine Namen für Fremdschlüsseleinschränkungen mit mehr als 64 Zeichen geben.
- Es dürfen keine veralteten SQL-Modi in Ihrer `sql_mode`-Systemvariableneinstellung definiert sein.
- Es darf keine Tabellen oder gespeicherte Prozeduren mit einzelnen `ENUM`- oder `SET`-Spaltenelementen geben, deren Länge 255 Zeichen überschreitet.
- Es darf keine Tabellenpartitionen innerhalb von freigegebenen InnoDB-Tablespaces geben.
- Es darf keine zirkulären Referenzen in Tablespace-Datendateipfaden geben.
- Es darf keine Abfragen oder gespeicherten Programmdefinitionen geben, die `ASC`- oder `DESC`-Qualifizierer für `GROUP BY`-Klauseln verwenden.
- Es darf keine entfernten Systemvariablen geben und Systemvariablen müssen die neuen Standardwerte für MySQL 8.0 verwenden.

- Es darf keine Nullwerte (`0`) für Date, Datetime oder Timestamp geben.
- Es darf keine Schemainkonsistenzen geben, die auf das Entfernen oder die Beschädigung von Dateien zurückzuführen sind.
- Es darf keine Tabellennamen geben, die die `FTS`-Zeichenfolge enthalten.
- Es darf keine InnoDB-Tabellen geben, die zu einer anderen Engine gehören.
- Es darf keine Tabellen- oder Schemanamen geben, die für MySQL 5.7 ungültig sind.

Weitere Informationen zu den ausgeführten Vorabprüfungen finden Sie unter Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL.

Weitere Informationen zum Ausführen von Upgrades auf MySQL 8.0 finden Sie unter Ausführen von MySQL-Upgrades in der MySQL-Dokumentation. Eine allgemeine Beschreibung der Änderungen in MySQL 8.0 finden Sie unter Was ist neu in MySQL 8.0 in der MySQL-Dokumentation.

Zusammenfassung der Vorabprüfungen für Aurora-MySQL-Upgrades

Aurora MySQL hat beim Upgrade von Version 2 auf Version 3 seine eigenen spezifischen Anforderungen, darunter die folgenden:

- Es darf keine veraltete SQL-Syntax wie `SQL_CACHE`, `SQL_NO_CACHE` und `QUERY_CACHE` in Ansichten, Routinen, Triggern und Ereignissen geben.
- Es darf in keiner Tabelle eine `FTS_DOC_ID`-Spalte ohne `FTS`-Index geben.
- Es darf keine Nichtübereinstimmung von Spaltendefinitionen zwischen dem InnoDB-Datenwörterbuch und der tatsächlichen Tabellendefinition geben.
- Alle Datenbank- und Tabellennamen dürfen nur Kleinbuchstaben enthalten, wenn der Parameter `lower_case_table_names` auf `1` gesetzt ist.
- Es darf keine Ereignisse und Trigger mit fehlenden oder leeren Definern oder ungültigen Erstellungskontexten geben.
- Alle Triggernamen in einer Datenbank müssen eindeutig sein.
- Die DDL-Wiederherstellung und schnelle DDL werden in Aurora-MySQL-Version 3 nicht unterstützt. Es darf keine Artefakte in Datenbanken geben, die mit diesen Funktionen in Zusammenhang stehen.
- Tabellen mit dem Zeilenformat `REDUNDANT` oder `COMPACT` dürfen keine Indizes größer als 767 Byte enthalten.

- Die Präfixlänge von Indizes, die für `tiny`-Textspalten definiert sind, darf 255 Byte nicht überschreiten. Mit dem `utf8mb4`-Zeichensatz ist die unterstützte Präfixlänge auf 63 Zeichen begrenzt.

  Eine größere Präfixlänge war in MySQL 5.7 unter Verwendung des Parameters `innodb_large_prefix` zulässig. Dieser Parameter ist seit MySQL 8.0 veraltet.
- Es darf keine Inkonsistenzen der InnoDB-Metadaten in der Tabelle `mysql.host` geben.
- Es darf keine Nichtübereinstimmung von Spaltendatentypen in den Systemtabellen geben.
- Es darf keine XA-Transaktionen im `prepared`-Status geben.
- Spaltennamen in Ansichten dürfen nicht länger als 64 Zeichen sein.
- Sonderzeichen in gespeicherten Prozeduren dürfen nicht inkonsistent sein.
- Es darf keine Inkonsistenz bei den Datendateipfaden von Tabellen geben.

Weitere Informationen zu den ausgeführten Vorabprüfungen finden Sie unter Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL.

Referenz zu Vorabprüfungsbeschreibungen für Aurora MySQL

Die Upgrade-Vorabprüfungen für Aurora MySQL werden hier ausführlich beschrieben.

Inhalt

- Fehler
  - MySQL-Vorabprüfungen, die Fehler melden
  - Aurora-MySQL-Vorabprüfungen, die Fehler melden
- Warnungen
  - MySQL-Vorabprüfungen, die Warnungen melden
  - Aurora-MySQL-Vorabprüfungen, die Warnungen melden
- Hinweise
- Fehler, Warnungen oder Hinweise

Fehler

Die folgenden Vorabprüfungen generieren Fehler, wenn die Vorabprüfung fehlschlägt und das Upgrade nicht fortgesetzt werden kann.

Themen

- MySQL-Vorabprüfungen, die Fehler melden
- Aurora-MySQL-Vorabprüfungen, die Fehler melden

MySQL-Vorabprüfungen, die Fehler melden

Die folgenden Vorabprüfungen stammen von Community MySQL:

- checkTableMysqlSchema
- circularDirectoryCheck
- columnsWhichCannotHaveDefaultsCheck
- depreciatedSyntaxCheck
- engineMixupCheck
- enumSetElementLengthCheck
- foreignKeyLengthCheck
- getDuplicateTriggers
- getEventsWithNullDefiner
- getMismatchedMetadata
- getTriggersWithNullDefiner
- getValueOfVariablelower_case_table_names
- groupByAscSyntaxCheck
- mysqlEmptyDotTableSyntaxCheck
- mysqlIndexTooLargeCheck
- mysqlInvalid57NamesCheck
- mysqlOrphanedRoutinesCheck
- mysqlSchemaCheck
- nonNativePartitioningCheck
- oldTemporalCheck
- partitionedTablesInSharedTablespaceCheck
- removedFunctionsCheck
- routineSyntaxCheck
- schemaInconsistencyCheck

checkTableMysqlSchema

Stufe der Vorabprüfung: Fehler

Probleme, die vom Befehl **`check table x for upgrade`** für das **`mysql`**-Schema gemeldet wurden

Bevor das Upgrade auf Aurora-MySQL-Version 3 gestartet wird, wird `check table for upgrade` für jede Tabelle im `mysql`-Schema auf der DB-Instance ausgeführt. Der Befehl `check table for upgrade` untersucht Tabellen auf mögliche Probleme, die bei einem Upgrade auf eine neuere Version von MySQL auftreten könnten. Das Ausführen dieses Befehls vor dem Upgrade kann helfen, etwaige Inkompatibilitäten im Voraus zu identifizieren und zu beheben, sodass der eigentliche Upgrade-Prozess reibungsloser abläuft.

Mit diesem Befehl werden verschiedene Prüfungen für jede Tabelle durchgeführt, z. B. die folgenden:

- Überprüfung, ob die Tabellenstruktur und die Metadaten mit der Ziel-MySQL-Version kompatibel sind
- Prüfung auf veraltete oder entfernte Funktionen, die von der Tabelle verwendet werden
- Sicherstellung, dass die Tabelle ordnungsgemäß und ohne Datenverlust aktualisiert werden kann

Weitere Informationen finden Sie unter CHECK-TABLE-Anweisung in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "checkTableMysqlSchema",
  "title": "Issues reported by 'check table x for upgrade' command for mysql
 schema.",
  "status": "OK",
  "detectedProblems": []
}
```

Die Ausgabe dieser Vorabprüfung hängt davon ab, welcher Fehler aufgetreten ist und wann er aufgetreten ist, da `check table for upgrade` mehrere Prüfungen durchführt.

Wenn Sie bei dieser Vorabprüfung auf Fehler stoßen, wenden Sie sich an den AWS Support, um die Behebung der Metadaten-Inkonsistenz zu beantragen. Alternativ können Sie das Upgrade

erneut versuchen, indem Sie einen logischen Dump durchführen und dann in einem neuen DB-Cluster von Aurora-MySQL-Version 3 wiederherstellen.

circularDirectoryCheck

Stufe der Vorabprüfung: Fehler

Zirkuläre Verzeichnisverweise in Tablespace-Datendateipfaden

Ab MySQL 8.0.17 erlaubt die Klausel `CREATE TABLESPACE ... ADD DATAFILE` keine zirkulären Verzeichnisverweise mehr. Entfernen Sie zur Vermeidung von Upgrade-Problemen alle zirkulären Verzeichnisverweise aus den Tablespace-Datendateipfaden, bevor Sie auf Aurora-MySQL-Version 3 aktualisieren.

Beispielausgabe:

```
{
  "id": "circularDirectory",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "description": "Error: Following tablespaces contain circular directory references
 (e.g. the reference '/../') in data file paths which as of MySQL 8.0.17 are not
 permitted by the CREATE TABLESPACE ... ADD DATAFILE clause. An exception to the
 restriction exists on Linux, where a circular directory reference is permitted if
 the preceding directory is a symbolic link. To avoid upgrade issues, remove any
 circular directory references from tablespace data file paths before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-innodb-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "ts2",
        "description": "circular reference in datafile path: '/home/ec2-user/dbdata/
mysql_5_7_44/../ts2.ibd'",
        "dbObjectType": "Tablespace"
      }
  ]
}
```

Wenn Sie diesen Fehler erhalten, erstellen Sie Ihre Tabellen neu, indem Sie einen Datei-pro-Tabelle-Tablespace verwenden. Verwenden Sie Standarddateipfade für alle Tablespace- und Tabellendefinitionen.

> **Note**
>
> Aurora MySQL unterstützt keine allgemeinen Tablespaces oder `CREATE TABLESPACE`-Befehle.
> Bevor Sie Tablespaces neu erstellen, finden Sie in der MySQL-Dokumentation unter Online-DDL-Operationen Informationen zu den Auswirkungen von Sperren und Datenverschiebungen auf Vordergrundtransaktionen.

Nach dem Neuerstellen ist die Vorabprüfung erfolgreich, sodass das Upgrade fortgesetzt werden kann.

```
{
  "id": "circularDirectoryCheck",
  "title": "Circular directory references in tablespace data file paths",
  "status": "OK",
  "detectedProblems": []
},
```

columnsWhichCannotHaveDefaultsCheck

Stufe der Vorabprüfung: Fehler

Spalten, die keine Standardwerte haben dürfen

Vor MySQL 8.0.13 dürfen `BLOB`-, `TEXT`-, `GEOMETRY`- und `JSON`-Spalten keine Standardwerte haben. Entfernen Sie alle Standardklauseln in diesen Spalten, bevor Sie ein Upgrade auf Aurora-MySQL-Version 3 durchführen. Weitere Informationen zu Änderungen an der Standardbehandlung für diese Datentypen finden Sie unter Standardwerte für Datentypen in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "description": "Error: The following columns are defined as either BLOB, TEXT,
 GEOMETRY or JSON and have a default value set. These data types cannot have default
 values in MySQL versions prior to 8.0.13, while starting with 8.0.13, the default
```

```
 value must be specified as an expression. In order to fix this issue, please use
 the ALTER TABLE ... ALTER COLUMN ... DROP DEFAULT statement.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/data-type-
defaults.html#data-type-defaults-explicit",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test_blob_default.geo_col",
        "description": "geometry"
      }
  ]
},
```

Die Vorabprüfung gibt einen Fehler zurück, weil die Spalte `geo_col` in der Tabelle `test.test_blob_default` einen `BLOB`-, `TEXT`-, `GEOMETRY`- oder `JSON`-Datentyp mit einem angegebenen Standardwert verwendet.

Ein Blick auf die Tabellendefinition zeigt, dass die Spalte `geo_col` als `geo_col geometry NOT NULL default ''` definiert ist.

```
mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL DEFAULT ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Durch das Entfernen dieser Standardklausel kann die Vorabprüfung erfolgreich abgeschlossen werden.

> **Note**
>
> Bevor Sie `ALTER TABLE`-Anweisungen ausführen oder Tablespaces neu erstellen, finden Sie in der MySQL-Dokumentation unter Online-DDL-Operationen Informationen zu den Auswirkungen von Sperren und Datenverschiebungen auf Vordergrundtransaktionen.

```
mysql> ALTER TABLE test_blob_default modify COLUMN geo_col geometry NOT NULL;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> show create table test_blob_default\G
*************************** 1. row ***************************
       Table: test_blob_default
Create Table: CREATE TABLE `test_blob_default` (
  `geo_col` geometry NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Die Vorabprüfung ist erfolgreich und Sie können das Upgrade erneut versuchen.

```
{
  "id": "columnsWhichCannotHaveDefaultsCheck",
  "title": "Columns which cannot have default values",
  "status": "OK",
  "detectedProblems": []
},
```

depreciatedSyntaxCheck

Stufe der Vorabprüfung: Fehler

Verwendung von veralteten Schlüsselwörtern in der Definition

MySQL 8.0 hat den Abfrage-Cache entfernt. Infolgedessen wurde ein Teil der für den Abfrage-Cache spezifischen SQL-Syntax entfernt. Wenn eines Ihrer Datenbankobjekte die Schlüsselwörter `QUERY CACHE`, `SQL_CACHE` oder `SQL_NO_CACHE` enthält, wird ein Fehler bei der Vorabprüfung zurückgegeben. Sie beheben dieses Problem, indem Sie diese Objekte erneut erstellen und die genannten Schlüsselwörter entfernen.

Beispielausgabe:

```
{
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "description": "Error: The following DB objects contain keywords like 'QUERY
 CACHE', 'SQL_CACHE', 'SQL_NO_CACHE' which are not supported in major version 8.0.
 It is recommended to drop these DB objects or rebuild without any of the above
 keywords before upgrade.",
  "detectedProblems": [
      {
```

```
"level": "Error",
"dbObject": "test.no_query_cache_check",
"description": "PROCEDURE uses depreciated words in definition"
      }
  ]
}
```

Die Vorabprüfung meldet, dass die gespeicherte Prozedur `test.no_query_cache_check` eines der entfernten Schlüsselwörter verwendet. Ein Blick auf die Prozedurdefinition zeigt, dass sie `SQL_NO_CACHE` verwendet.

```
mysql> show create procedure test.no_query_cache_check\G
*************************** 1. row ***************************
           Procedure: no_query_cache_check
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `no_query_cache_check`()
BEGIN
    SELECT SQL_NO_CACHE k from sbtest1 where id > 10 and id < 20 group by k asc;
END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Entfernen Sie das Schlüsselwort.

```
mysql> drop procedure test.no_query_cache_check;
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter //

mysql> CREATE DEFINER=`reinvent`@`%` PROCEDURE `no_query_cache_check`() BEGIN
 SELECT k from sbtest1 where id > 10 and id < 20 group by k asc; END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Nach dem Entfernen des Schlüsselworts wird die Vorabprüfung erfolgreich abgeschlossen.

```
{
```

```
  "id": "depreciatedSyntaxCheck",
  "title": "Usage of depreciated keywords in definition",
  "status": "OK",
  "detectedProblems": []
}
```

engineMixupCheck

Stufe der Vorabprüfung: Fehler

Von InnoDB erkannte Tabellen, die zu einer anderen Engine gehören

Ähnlich wie schemaInconsistencyCheck stellt diese Vorabprüfung sicher, dass die Tabellenmetadaten in MySQL konsistent sind, bevor mit dem Upgrade fortgefahren wird.

Wenn Sie bei dieser Vorabprüfung auf Fehler stoßen, wenden Sie sich an den AWS Support, um die Behebung der Metadaten-Inkonsistenz zu beantragen. Alternativ können Sie das Upgrade erneut versuchen, indem Sie einen logischen Dump durchführen und dann in einem neuen DB-Cluster von Aurora-MySQL-Version 3 wiederherstellen.

Beispielausgabe:

```
{
  "id": "engineMixupCheck",
  "title": "Tables recognized by InnoDB that belong to a different engine",
  "status": "OK",
  "description": "Error: Following tables are recognized by InnoDB engine while the
 SQL layer believes they belong to a different engine. Such situation may happen
 when one removes InnoDB table files manually from the disk and creates e.g. a
 MyISAM table with the same name.\n\nA possible way to solve this situation is to
 e.g. in case of MyISAM table:\n\n1. Rename the MyISAM table to a temporary name
 (RENAME TABLE).\n2. Create some dummy InnoDB table (its definition does not need to
 match), then copy (copy, not move) and rename the dummy .frm and .ibd files to the
 orphan name using OS file commands.\n3. The orphan table can be then dropped (DROP
 TABLE), as well as the dummy table.\n4. Finally the MyISAM table can be renamed
 back to its original name.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.general_log_backup",
        "description": "recognized by the InnoDB engine but belongs to CSV"
      }
  ]
```

```
}
```

enumSetElementLengthCheck

Stufe der Vorabprüfung: Fehler

ENUM- und **SET**-Spaltendefinitionen mit Elementen, die länger als 255 Zeichen sind

Tabellen und gespeicherte Prozeduren dürfen keine ENUM- oder SET-Spaltenelemente enthalten, die 255 Zeichen oder 1 020 Byte überschreiten. Vor MySQL 8.0 betrug die maximale kombinierte Länge 64 KB, aber 8.0 begrenzt einzelne Elemente auf 255 Zeichen oder 1 020 Byte (mit Unterstützung für Multibyte). Wenn bei der Vorabprüfung für `enumSetElementLengthCheck` ein Fehler auftritt, ändern Sie alle Elemente, die diese neuen Grenzwerte überschreiten, bevor Sie das Upgrade erneut versuchen.

Beispielausgabe:

```
{
  "id": "enumSetElementLengthCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "description": "Error: The following columns are defined as either ENUM or SET and
 contain at least one element longer that 255 characters. They need to be altered so
 that all elements fit into the 255 characters limit.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/string-type-
overview.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.large_set.s",
        "description": "SET contains element longer than 255 characters"
      }
  ]
},
```

Bei der Vorabprüfung wird ein Fehler gemeldet, da die Spalte `s` in der Tabelle `test.large_set` ein SET-Element mit mehr als 255 Zeichen enthält.

Nach dem Reduzieren der SET-Größe für diese Spalte ist die Vorabprüfung erfolgreich, sodass das Upgrade fortgesetzt werden kann.

```
{
  "id": "enumSetElementLenghtCheck",
  "title": "ENUM/SET column definitions containing elements longer than 255
 characters",
  "status": "OK",
  "detectedProblems": []
},
```

foreignKeyLengthCheck

Stufe der Vorabprüfung: Fehler

Namen für Fremdschlüsseleinschränkungen mit mehr als 64 Zeichen

In MySQL ist die Länge von Bezeichnern auf 64 Zeichen begrenzt, wie in der MySQL-Dokumentation beschrieben. Aufgrund von Problemen, bei denen Fremdschlüssellängen diesem Wert entsprechen oder diesen überschreiten könnten, was zu Upgrade-Fehlern führte, wurde diese Vorabprüfung implementiert. Wenn bei dieser Vorabprüfung Fehler auftreten, sollten Sie Ihre Einschränkung ändern oder umbenennen, sodass sie weniger als 64 Zeichen umfasst, bevor Sie das Upgrade erneut versuchen.

Beispielausgabe:

```
{
  "id": "foreignKeyLength",
  "title": "Foreign key constraint names longer than 64 characters",
  "status": "OK",
  "detectedProblems": []
}
```

getDuplicateTriggers

Stufe der Vorabprüfung: Fehler

Alle Triggernamen in einer Datenbank müssen eindeutig sein.

Aufgrund von Änderungen in der Implementierung des Datenwörterbuchs unterstützt MySQL 8.0 in einer Datenbank keine Trigger, bei denen zwischen Groß- und Kleinschreibung unterschieden wird. Diese Vorabprüfung validiert, dass Ihr DB-Cluster keine Datenbanken mit doppelten Triggern enthält. Weitere Informationen finden Sie unter Groß- und Kleinschreibung von Bezeichnern in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "description": "Error: You have one or more database containing duplicate
 triggers. Mysql 8.0 does not support case sensitive triggers within a database
 https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html. To
 upgrade to MySQL 8.0, drop the triggers with case-insensitive duplicate names and
 recreate with distinct names.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_product"
      },
      {
        "level": "Error",
        "dbObject": "test",
        "description": "before_insert_PRODUCT"
      }
  ]
}
```

Bei der Vorabprüfung wird ein Fehler gemeldet, dass der Datenbank-Cluster zwei Trigger mit demselben Namen enthält, jedoch in unterschiedlicher Groß- und Kleinschreibung: `test.before_insert_product` und `test.before_insert_PRODUCT`.

Benennen Sie die Trigger vor dem Upgrade um oder löschen Sie sie und erstellen Sie sie mit einem neuen Namen neu.

Nach dem Umbenennen von `test.before_insert_PRODUCT` in `test.before_insert_product_2` ist die Vorabprüfung erfolgreich.

```
{
  "id": "getDuplicateTriggers",
  "title": "MySQL pre-checks that all trigger names in a database are unique or
 not.",
  "status": "OK",
  "detectedProblems": []
```

```
}
```

getEventsWithNullDefiner

Stufe der Vorabprüfung: Fehler

Die Definer-Spalte für **`mysql.event`** darf weder Null noch leer sein.

Das `DEFINER`-Attribut gibt das MySQL-Konto an, das Eigentümer einer gespeicherten Objektdefinition ist, z. B. eines Triggers, einer gespeicherten Prozedur oder eines Ereignisses. Dieses Attribut ist besonders nützlich in Situationen, in denen Sie den Sicherheitskontext kontrollieren möchten, in dem das gespeicherte Objekt ausgeführt wird. Wenn beim Erstellen eines gespeicherten Objekts kein `DEFINER` angegeben wird, wird standardmäßig der Benutzer verwendet, der das Objekt erstellt hat.

Beim Upgrade auf MySQL 8.0 dürfen sich keine gespeicherten Objekte im MySQL-Datenwörterbuch befinden, die einen `null`- oder leeren Definer aufweisen. Wenn solche gespeicherten Objekte vorhanden sind, wird ein Fehler bei der Vorabprüfung ausgelöst. Sie müssen diesen beheben, bevor das Upgrade fortgesetzt werden kann.

Fehlerbeispiel:

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "description": "Error: Set definer column in mysql.event to a valid non-null
 definer.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.get_version",
        "description": "Set definer for event get_version in Schema test"
      }
  ]
}
```

Die Vorabprüfung gibt einen Fehler für das `test.get_version`-Ereignis zurück, da es einen `null`-Definer aufweist.

Zum Beheben dieses Problems können Sie die Ereignisdefinition überprüfen. Wie Sie sehen können, ist der Definer `null` oder leer.

```
mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+---------+
| db   | name        | definer |
+------+-------------+---------+
| test | get_version |         |
+------+-------------+---------+
1 row in set (0.00 sec)
```

Löschen Sie das Ereignis oder erstellen Sie es mit einem gültigen Definer neu.

> **Note**
>
> Bevor Sie einen `DEFINER` löschen oder neu definieren, sollten Sie Ihre Anwendungs- und Berechtigungsanforderungen sorgfältig überprüfen. Weitere Informationen finden Sie unter Zugriffssteuerung für gespeicherte Objekte in der MySQL-Dokumentation.

```
mysql> drop event test.get_version;
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> delimiter $$
mysql> CREATE EVENT get_version
    ->     ON SCHEDULE
    ->       EVERY 1 DAY
    ->     DO
    ->      ///DO SOMETHING //
    -> $$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> select db,name,definer from mysql.event where name='get_version';
+------+-------------+------------+
| db   | name        | definer    |
+------+-------------+------------+
| test | get_version | reinvent@% |
+------+-------------+------------+
1 row in set (0.00 sec)
```

Die Vorabprüfung ist jetzt erfolgreich.

```
{
  "id": "getEventsWithNullDefiner",
  "title": "The definer column for mysql.event cannot be null or blank.",
  "status": "OK",
  "detectedProblems": []},
```

getMismatchedMetadata

Stufe der Vorabprüfung: Fehler

Nichtübereinstimmung von Spaltendefinitionen zwischen dem InnoDB-Datenwörterbuch und der tatsächlichen Tabellendefinition

Ähnlich wie schemaInconsistencyCheck stellt diese Vorabprüfung sicher, dass die Tabellenmetadaten in MySQL konsistent sind, bevor mit dem Upgrade fortgefahren wird. In diesem Fall stellt die Vorabprüfung sicher, dass die Spaltendefinitionen zwischen dem InnoDB-Datenwörterbuch und der MySQL-Tabellendefinition übereinstimmen. Wenn eine Nichtübereinstimmung festgestellt wird, wird das Upgrade nicht fortgesetzt.

Wenn Sie bei dieser Vorabprüfung auf Fehler stoßen, wenden Sie sich an den AWS Support, um die Behebung der Metadaten-Inkonsistenz zu beantragen. Alternativ können Sie das Upgrade erneut versuchen, indem Sie einen logischen Dump durchführen und dann in einem neuen DB-Cluster von Aurora-MySQL-Version 3 wiederherstellen.

Beispielausgabe:

```
{
  "id": "getMismatchedMetadata",
  "title": "Column definition mismatch between InnoDB Data Dictionary and actual
 table definition.",
  "status": "OK",
  "description": "Error: Your database has mismatched metadata. The upgrade to mysql
 8.0 will not succeed until this is fixed.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.mismatchTable",
        "description": "Table `test/mismatchTable` column names mismatch with InnoDb
 dictionary column names: iD <> id"
      }
  ]
}
```

Bei der Vorabprüfung wird eine Nichtübereinstimmung in den Metadaten für die Spalte `id` in der Tabelle `test.mismatchTable` gemeldet. Insbesondere führen die MySQL-Metadaten die Spalte als `iD`, InnoDB hingegen als `id`.

getTriggersWithNullDefiner

Stufe der Vorabprüfung: Fehler

Die Definer-Spalte für **`information_schema.triggers`** darf nicht **`null`** oder leer sein.

Die Vorabprüfung validiert, dass Ihre Datenbank keine Trigger enthält, die mit `null`- oder leeren Definern definiert sind. Weitere Informationen zu den Definer-Anforderungen für gespeicherte Objekte finden Sie unter getEventsWithNullDefiner.

Beispielausgabe:

```
{
  "id": "getTriggersWithNullDefiner",
  "title": "The definer column for information_schema.triggers cannot be null or
 blank.",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.example_trigger",
        "description": "Set definer for trigger example_trigger in Schema test"
      }
  ]
}
```

Die Vorabprüfung gibt einen Fehler zurück, da der Trigger `example_trigger` im `test`-Schema über einen `null`-Definer verfügt. Zur Behebung dieses Problems sollte der Definer korrigiert werden, indem der Trigger mit einem gültigen Benutzer neu erstellt oder entfernt wird. Weitere Informationen finden Sie im Beispiel in getEventsWithNullDefiner.

> **Note**
>
> Bevor Sie einen `DEFINER` löschen oder neu definieren, sollten Sie Ihre Anwendungs- und Berechtigungsanforderungen sorgfältig überprüfen. Weitere Informationen finden Sie unter Zugriffssteuerung für gespeicherte Objekte in der MySQL-Dokumentation.

getValueOfVariablelower_case_table_names

Stufe der Vorabprüfung: Fehler

Alle Datenbank- oder Tabellennamen dürfen nur Kleinbuchstaben enthalten, wenn der Parameter **`lower_case_table_names`** auf **`1`** gesetzt ist.

Vor MySQL 8.0 entsprachen Datenbanknamen, Tabellennamen und andere Objekte Dateien im Datenverzeichnis, wie z. B. dateibasierten Metadaten (.frm). Mit der Systemvariablen lower_case_table_names können Benutzer steuern, wie der Server die Groß- und Kleinschreibung von Bezeichnern bei Datenbankobjekten behandelt und wie solche Metadatenobjekte gespeichert werden. Dieser Parameter könnte auf einem bereits initialisierten Server nach einem Neustart geändert werden.

In MySQL 8.0 steuert dieser Parameter zwar weiterhin, wie der Server die Groß- und Kleinschreibung von Bezeichnern behandelt, er kann jedoch nicht geändert werden, nachdem das Datenwörterbuch initialisiert wurde. Beim Upgrade oder der Erstellung einer MySQL-8.0-Datenbank wird der Wert, der für `lower_case_table_names` beim ersten Start des Datenwörterbuchs in MySQL festgelegt wird, für die gesamte Lebensdauer dieser Datenbank verwendet. Diese Einschränkung wurde im Rahmen der Implementierung des Atomic Data Dictionary eingeführt, bei der Datenbankobjekte von dateibasierten Metadaten zu internen InnoDB-Tabellen im `mysql`-Schema migriert werden.

Weitere Informationen finden Sie unter Änderungen am Datenwörterbuch in der MySQL-Dokumentation.

Um Probleme während dem Upgrade bei der Aktualisierung dateibasierter Metadaten auf das neue Atomic Data Dictionary zu vermeiden, validiert diese Vorabprüfung, dass bei `lower_case_table_names = 1` alle Tabellen in Kleinbuchstaben auf der Festplatte gespeichert sind. Wenn dies nicht der Fall ist, wird ein Fehler bei der Vorabprüfung zurückgegeben und Sie müssen die Metadaten korrigieren, bevor Sie mit dem Upgrade fortfahren können.

Beispielausgabe:

```
{
  "id": "getValueOfVariablelower_case_table_names",
  "title": "MySQL pre-checks that all database or table names are lowercase when the
 lower_case_table_names parameter is set to 1.",
  "status": "OK",
```

```
  "description": "Error: You have one or more databases or tables with uppercase
 letters in the names, but the lower_case_table_names parameter is set to 1. To
 upgrade to MySQL 8.0, either change all database or table names to lowercase, or
 set the parameter to 0.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.TEST",
        "description": "Table test.TEST contains one or more capital letters in name
 while lower_case_table_names = 1"
      }
  ]
}
```

Ein Fehler wird zurückgegeben, da die Tabelle `test.TEST` Großbuchstaben enthält, aber `lower_case_table_names` auf 1 gesetzt ist.

Zum Beheben dieses Problems können Sie die Tabelle umbenennen, sodass Kleinbuchstaben verwendet werden, oder Sie können den Parameter `lower_case_table_names` im DB-Cluster ändern, bevor Sie mit dem Upgrade beginnen.

> **Note**
>
> Testen und lesen Sie die Dokumentation zur Berücksichtigung der Groß-/Kleinschreibung in MySQL sorgfältig und überprüfen Sie, wie sich solche Änderungen auf Ihre Anwendung auswirken könnten.
> Lesen Sie auch in der MySQL-8.0-Dokumentation nach, wie lower_case_table_names in MySQL 8.0 unterschiedlich behandelt werden.

groupByAscSyntaxCheck

Stufe der Vorabprüfung: Fehler

Verwendung einer entfernten **`GROUP BY ASC/DESC`**-Syntax

Ab MySQL 8.0.13 wurde die veraltete `ASC`- oder `DESC`-Syntax für `GROUP BY`-Klauseln entfernt. Abfragen, die auf der `GROUP BY`-Sortierung basieren, können jetzt unterschiedliche Ergebnisse liefern. Um eine bestimmte Sortierreihenfolge zu erhalten, verwenden Sie stattdessen eine `ORDER BY`-Klausel. Wenn in Ihrer Datenbank Objekte existieren, die diese Syntax verwenden, müssen

Sie sie mithilfe einer ORDER BY-Klausel neu erstellen, bevor Sie das Upgrade erneut versuchen. Weitere Informationen finden Sie unter SQL-Änderungen in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "groupbyAscSyntaxCheck",
  "title": "Usage of removed GROUP BY ASC/DESC syntax",
  "status": "OK",
  "description": "Error: The following DB objects use removed GROUP BY ASC/DESC
 syntax. They need to be altered so that ASC/DESC keyword is removed from GROUP BY
 clause and placed in appropriate ORDER BY clause.",
  "documentationLink": "https://dev.mysql.com/doc/relnotes/mysql/8.0/en/
news-8-0-13.html#mysqld-8-0-13-sql-syntax",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.groupbyasc",
        "description": "PROCEDURE uses removed GROUP BY ASC syntax",
        "dbObjectType": "Routine"
      }
  ]
}
```

mysqlEmptyDotTableSyntaxCheck

Stufe der Vorabprüfung: Fehler

Suchen Sie nach veralteter **.<table>**-Syntax, die in Routinen verwendet wird.

In MySQL 8.0 dürfen Routinen keine veraltete Bezeichnersyntax (\".<table>\") mehr enthalten. Wenn gespeicherte Routinen oder Trigger solche Bezeichner enthalten, schlägt das Upgrade fehl. Die folgende .dot_table-Referenz ist beispielsweise nicht mehr zulässig:

```
mysql> show create procedure incorrect_procedure\G
*************************** 1. row ***************************
           Procedure: incorrect_procedure
            sql_mode:
    Create Procedure: CREATE DEFINER=`reinvent`@`%` PROCEDURE
 `incorrect_procedure`()
BEGIN delete FROM .dot_table; select * from .dot_table where 1=1; END
character_set_client: utf8mb4
```

```
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Nachdem Sie die Routinen und Trigger neu erstellt haben, um die korrekte Bezeichnersyntax und das richtige Maskieren zu verwenden, ist die Vorabprüfung erfolgreich und das Upgrade kann fortgesetzt werden. Weitere Informationen finden Sie unter Namen von Schemaobjekten in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "mysqlEmptyDotTableSyntaxCheck",
  "title": "Check for deprecated '.<table>' syntax used in routines.",
  "status": "OK",
  "description": "Error: The following routines contain identifiers in deprecated
 identifier syntax (\".<table>\"), and should be corrected before upgrade:\n",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.incorrect_procedure",
        "description": " routine body contains deprecated identifiers."
      }
  ]
}
```

Die Vorabprüfung gibt einen Fehler für die `incorrect_procedure`-Routine in der `test`-Datenbank zurück, da sie eine veraltete Syntax enthält.

Nachdem Sie die Routine korrigiert haben, ist die Vorabprüfung erfolgreich und Sie können das Upgrade erneut versuchen.

mysqlIndexTooLargeCheck

Stufe der Vorabprüfung: Fehler

Prüfen auf Indizes, die zu groß sind, um mit MySQL-Versionen höher als 5.7 zu funktionieren

Bei kompakten oder redundanten Zeilenformaten sollte es nicht möglich sein, einen Index mit einem Präfix von mehr als 767 Byte zu erstellen. Vor der MySQL-Version 5.7.35 war dies jedoch möglich. Weitere Informationen finden Sie unter Versionshinweise für MySQL 5.7.35.

Auf alle Indizes, die von diesem Fehler betroffen waren, kann nach dem Upgrade auf MySQL 8.0 nicht mehr zugegriffen werden. Diese Vorabprüfung identifiziert fehlerhafte Indizes, die neu erstellt werden müssen, bevor das Upgrade fortgesetzt werden kann.

```
{
 "id": "mysqlIndexTooLargeCheck",
 "title": "Check for indexes that are too large to work on higher versions of MySQL
Server than 5.7",
 "status": "OK",
 "description": "Error: The following indexes ware made too large for their format
in an older version of MySQL (older than 5.7.34). Normally those indexes within
tables with compact or redundant row formats shouldn't be larger than 767 bytes. To
fix this problem those indexes should be dropped before upgrading or those tables
will be inaccessible.",
 "detectedProblems": [
     {
       "level": "Error",
       "dbObject": "test.table_with_large_idx",
       "description": "IDX_2"
     }
 ]
}
```

Die Vorabprüfung gibt einen Fehler zurück, da die Tabelle `test.table_with_large_idx` einen Index für eine Tabelle mit kompaktem oder redundantem Zeilenformat enthält, dessen Größe 767 Byte überschreitet. Auf diese Tabellen kann nach dem Upgrade auf MySQL 8.0 nicht mehr zugegriffen werden. Führen Sie einen der folgenden Schritte aus, bevor Sie mit dem Upgrade fortfahren:

- Löschen Sie den in der Vorabprüfung genannten Index.
- Fügen Sie einen in der Vorabprüfung genannten Index hinzu.
- Ändern Sie das von der Tabelle verwendete Zeilenformat.

Hier erstellen wir die Tabelle neu, um den Fehler bei der Vorabprüfung zu beheben. Vergewissern Sie sich vor dem Neuerstellen der Tabelle, dass innodb_file_format auf `Barracuda` und innodb_default_row_format auf `dynamic` gesetzt ist. Dies sind die Standardwerte in MySQL 5.7. Weitere Informationen finden Sie unter InnoDB-Zeilenformate und InnoDB-Dateiformatverwaltung in der MySQL-Dokumentation.

> **Note**
>
> Bevor Sie Tablespaces neu erstellen, finden Sie in der MySQL-Dokumentation unter Online-DDL-Operationen Informationen zu den Auswirkungen von Sperren und Datenverschiebungen auf Vordergrundtransaktionen.

```
mysql > select @@innodb_file_format,@@innodb_default_row_format;
+----------------------+----------------------------+
| @@innodb_file_format | @@innodb_default_row_format |
+----------------------+----------------------------+
| Barracuda            | dynamic                     |
+----------------------+----------------------------+
1 row in set (0.00 sec)

mysql> optimize table table_with_large_idx;
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| Table                     | Op       | Msg_type | Msg_text
                                |
+---------------------------+----------+----------
+-------------------------------------------------------------------+
| test.table_with_large_idx | optimize | note     | Table does not support optimize,
 doing recreate + analyze instead |
| test.table_with_large_idx | optimize | status   | OK
                                |
+---------------------------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.02 sec)

# Verify FILE_FORMAT and ROW_FORMAT
mysql>  select * from information_schema.innodb_sys_tables where name like 'test/
table_with_large_idx';
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
| TABLE_ID | NAME                      | FLAG | N_COLS | SPACE | FILE_FORMAT |
 ROW_FORMAT | ZIP_PAGE_SIZE | SPACE_TYPE |
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
|       43 | test/table_with_large_idx |   33 |      4 |    26 | Barracuda   |
 Dynamic    |             0 | Single     |
```

```
+----------+---------------------------+------+--------+-------+-------------
+------------+---------------+------------+
1 row in set (0.00 sec)
```

Nach dem Neuerstellen der Tabelle ist die Vorabprüfung erfolgreich und das Upgrade kann fortgesetzt werden.

```
{
  "id": "mysqlIndexTooLargeCheck",
  "title": "Check for indexes that are too large to work on higher versions of MySQL
 Server than 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlInvalid57NamesCheck

Stufe der Vorabprüfung: Fehler

Prüfen auf ungültige Tabellen- und Schemanamen, die in MySQL 5.7 verwendet wurden

Bei der Migration zum neuen Datenwörterbuch in MySQL 8.0 darf Ihre Datenbank-Instance keine Schemas oder Tabellen enthalten, die mit dem Präfix `#mysql50#` versehen sind. Wenn solche Objekte existieren, schlägt das Upgrade fehl. Führen Sie zum Beheben dieses Problems mysqlcheck für die zurückgegebenen Schemas und Tabellen aus.

> **Note**
>
> Stellen Sie sicher, dass Sie eine MySQL-5.7-Version von `mysqlcheck` verwenden, da --fix-db-names und --fix-table-names aus MySQL 8.0 entfernt wurden.

Beispielausgabe:

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "description": "The following tables and/or schemas have invalid names. In order
 to fix them use the mysqlcheck utility as follows:\n\n  $ mysqlcheck --check-
upgrade --all-databases\n  $ mysqlcheck --fix-db-names --fix-table-names --all-
```

```
databases\n\nOR via mysql client, for eg:\n\n  ALTER DATABASE `#mysql50#lost+found`
 UPGRADE DATA DIRECTORY NAME;",
  "documentationLink": "https://dev.mysql.com/doc/refman/5.7/en/identifier-
mapping.html https://dev.mysql.com/doc/refman/5.7/en/alter-database.html https://
dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "#mysql50#fix_db_names",
        "description": "Schema name"
      }
  ]
}
```

Die Vorabprüfung meldet, dass das Schema `#mysql50#fix_db_names` einen ungültigen Namen hat.

Nach der Korrektur des Schemanamens ist die Vorabprüfung erfolgreich, sodass das Upgrade fortgesetzt werden kann.

```
{
  "id": "mysqlInvalid57NamesCheck",
  "title": "Check for invalid table names and schema names used in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlOrphanedRoutinesCheck

Stufe der Vorabprüfung: Fehler

Prüfen auf verwaiste Routinen in 5.7

Wenn bei der Migration zum neuen Datenwörterbuch in MySQL 8.0 gespeicherte Prozeduren in der Datenbank vorhanden sind, in denen das Schema nicht mehr existiert, schlägt das Upgrade fehl. Diese Vorabprüfung stellt sicher, dass alle Schemas, auf die in gespeicherten Prozeduren in Ihrer DB-Instance verwiesen wird, noch vorhanden sind. Löschen Sie diese gespeicherten Prozeduren, damit das Upgrade fortgesetzt werden kann.

Beispielausgabe:

```
{
```

```
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "description": "Error: The following routines have been orphaned. Schemas that
 they are referencing no longer exists.\nThey have to be cleaned up or the upgrade
 will fail.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "dropped_db.get_version",
        "description": "is orphaned"
      }
  ]
},
```

Die Vorabprüfung meldet, dass die gespeicherte Prozedur `get_version` in der `dropped_db`-Datenbank verwaist ist.

Zur Bereinigung dieser Prozedur können Sie das fehlende Schema neu erstellen.

```
mysql> create database dropped_db;
Query OK, 1 row affected (0.01 sec)
```

Nachdem das Schema neu erstellt wurde, können Sie die Prozedur löschen, damit das Upgrade fortgesetzt werden kann.

```
{
  "id": "mysqlOrphanedRoutinesCheck",
  "title": "Check for orphaned routines in 5.7",
  "status": "OK",
  "detectedProblems": []
},
```

mysqlSchemaCheck

Stufe der Vorabprüfung: Fehler

Tabellennamen im **`mysql`**-Schema kollidieren mit neuen Tabellen in MySQL 8.0

Das neue Atomic Data Dictionary, das in MySQL 8.0 eingeführt wurde, speichert alle Metadaten in einer Reihe interner InnoDB-Tabellen im `mysql`-Schema. Während des Upgrades werden

die neuen internen Datenwörterbuchtabellen im `mysql`-Schema erstellt. Zur Vermeidung von Namenskonflikten, die zu Upgrade-Fehlern führen würden, untersucht die Vorabprüfung alle Tabellennamen im `mysql`-Schema, um sicherzustellen, dass keiner der neuen Tabellennamen bereits verwendet wird. Falls doch, wird ein Fehler zurückgegeben und das Upgrade kann nicht fortgesetzt werden.

Beispielausgabe:

```
{
  "id": "mysqlSchema",
  "title": "Table names in the mysql schema conflicting with new tables in the
 latest MySQL.",
  "status": "OK",
  "description": "Error: The following tables in mysql schema have names that will
 conflict with the ones introduced in the latest version. They must be renamed or
 removed before upgrading (use RENAME TABLE command). This may also entail changes
 to applications that use the affected tables.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrade-before-you-
begin.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.tablespaces",
        "description": "Table name used in mysql schema.",
        "dbObjectType": "Table"
      }
  ]
}
```

Ein Fehler wird zurückgegeben, da im `mysql`-Schema bereits eine Tabelle mit dem Namen `tablespaces` existiert. Dies ist einer der neuen Tabellennamen für interne Datenwörterbücher in MySQL 8.0. Sie müssen solche Tabellen vor dem Upgrade mithilfe des Befehls `RENAME TABLE` umbenennen oder entfernen.

nonNativePartitioningCheck

Stufe der Vorabprüfung: Fehler

Partitionierte Tabellen mit Engines ohne nativer Partitionierung

Laut der MySQL-8.0-Dokumentation bieten derzeit zwei Speicher-Engines native Partitionierungsunterstützung: InnoDB und NDB. Von diesen wird nur InnoDB in Aurora-MySQL-

Version 3 unterstützt, kompatibel mit MySQL 8.0. Jeder Versuch, partitionierte Tabellen in MySQL 8.0 mit einer anderen Speicher-Engine zu erstellen, schlägt fehl. Diese Vorabprüfung sucht nach Tabellen in Ihrem DB-Cluster, die eine nicht-native Partitionierung verwenden. Wenn welche zurückgegeben werden, müssen Sie die Partitionierung entfernen oder die Speicher-Engine in InnoDB konvertieren.

Beispielausgabe:

```
{
  "id": "nonNativePartitioning",
  "title": "Partitioned tables using engines with non native partitioning",
  "status": "OK",
  "description": "Error: In the latest MySQL storage engine is responsible for
 providing its own partitioning handler, and the MySQL server no longer provides
 generic partitioning support. InnoDB and NDB are the only storage engines that
 provide a native partitioning handler that is supported in the latest MySQL. A
 partitioned table using any other storage engine must be altered—either to convert
 it to InnoDB or NDB, or to remove its partitioning—before upgrading the server,
 else it cannot be used afterwards.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-configuration-changes",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partMyisamTable",
         "description": "MyISAM engine does not support native partitioning",
         "dbObjectType": "Table"
      }
  ]
}
```

Hier verwendet eine MyISAM-Tabelle Partitionierung, was eine Aktion erfordert, bevor das Upgrade fortgesetzt werden kann.

oldTemporalCheck

Stufe der Vorabprüfung: Fehler

Verwendung eines alten temporalen Typs

„Alte Temporale“ beziehen sich auf Spalten mit temporalen Typen (wie `TIMESTAMP` und`DATETIME`), die in MySQL-Versionen 5.5 und niedriger erstellt wurden. In MySQL 8.0 wurde

die Unterstützung für diese alten temporalen Datentypen entfernt, was bedeutet, dass direkte Upgrades von MySQL 5.7 auf 8.0 nicht möglich sind, wenn die Datenbank diese alten temporalen Datentypen enthält. Zum Beheben dieses Problems müssen Sie alle Tabellen, die solche alten temporalen Datentypen enthalten, neu erstellen, bevor Sie mit dem Upgrade fortfahren.

Weitere Informationen zur Einstellung alter temporaler Datentypen in MySQL 5.7 finden Sie in diesem Blog. Weitere Informationen zur Entfernung alter temporaler Datentypen in MySQL 8.0 finden Sie in diesem Blog.

> **Note**
>
> Bevor Sie Tablespaces neu erstellen, finden Sie in der MySQL-Dokumentation unter Online-DDL-Operationen Informationen zu den Auswirkungen von Sperren und Datenverschiebungen auf Vordergrundtransaktionen.

Beispielausgabe:

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "description": "Error: Following table columns use a deprecated and no longer
 supported temporal disk storage format. They must be converted to the new format
 before upgrading. It can by done by rebuilding the table using 'ALTER TABLE
 <table_name> FORCE' command",
  "documentationLink": "https://dev.mysql.com/blog-archive/mysql-8-0-removing-
support-for-old-temporal-datatypes/",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.55_temporal_table.timestamp_column",
        "description": "timestamp /* 5.5 binary format */",
        "dbObjectType": "Column"
      }
  ]
},
```

Für die Spalte `timestamp_column` in der Tabelle `test.55_temporal_table` wird ein Fehler gemeldet, da sie ein altes temporales Festplattenspeicherformat verwendet, das nicht mehr unterstützt wird.

Um dieses Problem zu beheben und das Upgrade fortsetzen zu können, erstellen Sie die Tabelle neu, um das alte temporale Festplattenspeicherformat in das neue Format zu konvertieren, das in MySQL 5.6 eingeführt wurde. Weitere Informationen und Voraussetzungen dafür finden Sie unter Konvertierung zwischen 3-Byte- und 4-Byte-Unicode-Zeichensätzen in der MySQL-Dokumentation.

Wenn Sie den folgenden Befehl ausführen, um die in dieser Vorabprüfung genannten Tabellen neu zu erstellen, werden die alten temporalen Typen mit einer Genauigkeit von Sekundenbruchteilen in das neuere Format konvertiert.

```
ALTER TABLE ... ENGINE=InnoDB;
```

Weitere Informationen finden Sie unter ALTER-TABLE-Anweisung in der MySQL-Dokumentation.

Nachdem Sie die fragliche Tabelle neu erstellt und das Upgrade neu gestartet haben, ist die Kompatibilitätsprüfung erfolgreich, sodass das Upgrade fortgesetzt werden kann.

```
{
  "id": "oldTemporalCheck",
  "title": "Usage of old temporal type",
  "status": "OK",
  "detectedProblems": []
}
```

partitionedTablesInSharedTablespaceCheck

Stufe der Vorabprüfung: Fehler

Verwendung partitionierter Tabellen in freigegebenen Tablespaces

Ab MySQL 8.0.13 wurde die Unterstützung für das Platzieren von Tabellenpartitionen in freigegebenen Tablespaces entfernt. Verschieben Sie vor dem Upgrade alle derartigen Tabellen von freigegebenen Tablespaces in Datei-pro-Tabelle-Tablespaces.

> **Note**
>
> Bevor Sie Tablespaces neu erstellen, finden Sie in der MySQL-Dokumentation unter Partitionierungsoperationen Informationen zu den Auswirkungen von Sperren und Datenverschiebungen auf Vordergrundtransaktionen.

Beispielausgabe:

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "description": "Error: The following tables have partitions in shared tablespaces.
 They need to be moved to file-per-table tablespace before upgrading. You can do
 this by running query like 'ALTER TABLE table_name REORGANIZE PARTITION X INTO
 (PARTITION X VALUES LESS THAN (30) TABLESPACE=innodb_file_per_table);'",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.partInnoDBTable",
        "description": "Partition p1 is in shared tablespace innodb",
        "dbObjectType": "Table"
      }
  ]
}
```

Die Vorabprüfung schlägt fehl, weil sich die Partition p1 aus der Tabelle `test.partInnoDBTable` im System-Tablespace befindet.

Zum Beheben dieses Problems erstellen Sie die Tabelle `test.partInnodbTable` neu und platzieren die fehlerhafte Partition p1 in einem Datei-pro-Tabelle-Tablespace.

```
mysql > ALTER TABLE partInnodbTable REORGANIZE PARTITION p1
    ->   INTO (PARTITION p1 VALUES LESS THAN ('2014-01-01')
 TABLESPACE=innodb_file_per_table);
Query OK, 0 rows affected, 1 warning (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Danach ist die Vorabprüfung erfolgreich.

```
{
  "id": "partitionedTablesInSharedTablespaceCheck",
  "title": "Usage of partitioned tables in shared tablespaces",
  "status": "OK",
  "detectedProblems": []
}
```

removedFunctionsCheck

Stufe der Vorabprüfung: Fehler

Verwendung von Funktionen, die aus der neuesten MySQL-Version entfernt wurden

In MySQL 8.0 wurden eine Reihe von integrierten Funktionen entfernt. Bei dieser Vorabprüfung wird Ihre Datenbank auf Objekte untersucht, die diese Funktionen verwenden könnten. Wenn sie gefunden werden, wird ein Fehler zurückgegeben. Sie müssen die Probleme beheben, bevor Sie das Upgrade erneut versuchen können.

Bei den meisten entfernten Funktionen handelt es sich um Geofunktionen, die durch äquivalente ST_*-Funktionen ersetzt wurden. In diesen Fällen ändern Sie die Datenbankobjekte so, dass sie die neue Prozedurbenennung verwenden. Weitere Informationen finden Sie unter In MySQL 8.0 entfernte Funktionen in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "description": "Error: The following DB objects use functions that were removed
 in the latest MySQL version. Please make sure to update them to use supported
 alternatives before upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.GetLocationsInPolygon",
        "description": "PROCEDURE uses removed function POLYGONFROMTEXT (consider
 using ST_POLYGONFROMTEXT instead)",
        "dbObjectType": "Routine"
      },
      {
        "level": "Error",
        "dbObject": "test.InsertLocation",
        "description": "PROCEDURE uses removed function POINTFROMTEXT (consider
 using ST_POINTFROMTEXT instead)",
        "dbObjectType": "Routine"
      }
  ]
```

```
},
```

Die Vorabprüfung meldet, dass die gespeicherte Prozedur `test.GetLocationsInPolygon` zwei entfernte Funktionen verwendet: POLYGONFROMTEXT und POINTFROMTEXT. Außerdem wird empfohlen, die neuen ST_POLYGONFROMTEXT und ST_POINTFROMTEXT als Ersatz zu verwenden. Nachdem Sie die Prozedur anhand der Vorschläge neu erstellt haben, wird die Vorabprüfung erfolgreich abgeschlossen.

```
{
  "id": "removedFunctionsCheck",
  "title": "Usage of removed functions",
  "status": "OK",
  "detectedProblems": []
},
```

> **Note**
>
> In den meisten Fällen werden die veralteten Funktionen zwar direkt ersetzt, Sie sollten jedoch sicherstellen, dass Sie Ihre Anwendung testen und die Dokumentation auf Verhaltensänderungen überprüfen, die sich aus der Änderung ergeben.

routineSyntaxCheck

Stufe der Vorabprüfung: Fehler

MySQL-Syntaxprüfung für routinenähnliche Objekte

MySQL 8.0 hat reservierte Schlüsselwörter eingeführt, die zuvor nicht reserviert waren. Die Upgrade-Vorabprüfung wertet die Verwendung reservierter Schlüsselwörter in den Namen von Datenbankobjekten sowie in deren Definitionen und Körper aus. Wenn sie feststellt, dass reservierte Schlüsselwörter in Datenbankobjekten wie gespeicherten Prozeduren, Funktionen, Ereignissen und Triggern verwendet werden, schlägt das Upgrade fehl und ein Fehler wird in der Datei `upgrade-prechecks.log` veröffentlicht. Zum Beheben des Problems müssen Sie diese Objektdefinitionen aktualisieren und alle derartigen Referenzen vor dem Upgrade in einfache Anführungszeichen (') setzen. Weitere Informationen zum Maskieren reservierter Wörter in MySQL finden Sie unter Zeichenfolgenliterale in der MySQL-Dokumentation.

Alternativ können Sie den Namen in einen anderen Namen ändern, was möglicherweise Änderungen an der Anwendung erforderlich macht.

Beispielausgabe:

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "description": "The following objects did not pass a syntax check with the latest
 MySQL grammar. A common reason is that they reference names that conflict with new
 reserved keywords. You must update these routine definitions and `quote` any such
 references before upgrading.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
  "detectedProblems": [
      {
         "level": "Error",
         "dbObject": "test.select_res_word",
         "description": "at line 2,18: unexpected token 'except'",
         "dbObjectType": "Routine"
      }
  ]
}
```

Zum Beheben dieses Problems überprüfen Sie die Routinedefinition.

```
SHOW CREATE PROCEDURE test.select_res_word\G

*************************** 1. row ***************************
           Procedure: select_res_word
            sql_mode:
 ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_Z
    Create Procedure: CREATE PROCEDURE 'select_res_word'()
BEGIN
    SELECT * FROM except;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Die Prozedur verwendet eine Tabelle mit dem Namen `except`, was ein neues Schlüsselwort in MySQL 8.0 ist. Erstellen Sie die Prozedur neu, indem Sie das Zeichenfolgenliteral maskieren.

```
> drop procedure if exists select_res_word;
```

```
Query OK, 0 rows affected (0.00 sec)

> DELIMITER $$
 > CREATE PROCEDURE select_res_word()
    -> BEGIN
    ->     SELECT * FROM 'except';
    -> END$$
Query OK, 0 rows affected (0.00 sec)

 > DELIMITER ;
```

Die Vorabprüfung ist jetzt erfolgreich.

```
{
  "id": "routineSyntaxCheck",
  "title": "MySQL syntax check for routine-like objects",
  "status": "OK",
  "detectedProblems": []
}
```

schemaInconsistencyCheck

Stufe der Vorabprüfung: Fehler

Schemainkonsistenzen, die auf das Entfernen oder die Beschädigung von Dateien zurückzuführen sind

Wie bereits beschrieben, wurde mit MySQL 8.0 das Atomic Data Dictionary eingeführt, das alle Metadaten in einer Reihe interner InnoDB-Tabellen im `mysql`-Schema speichert. Diese neue Architektur bietet eine transaktionale, ACID-konforme Methode zur Verwaltung von Datenbankmetadaten und löst damit das „atomare DDL“-Problem des alten dateibasierten Ansatzes. Vor MySQL 8.0 konnten Schemaobjekte verwaisen, wenn eine DDL-Operation unerwartet unterbrochen wurde. Durch die Migration von dateibasierten Metadaten zu den neuen Tabellen des Atomic Data Dictionary während des Upgrades wird sichergestellt, dass es in der DB-Instance keine solchen verwaisten Schemaobjekte gibt. Wenn verwaiste Objekte gefunden werden, schlägt das Upgrade fehl.

Damit diese verwaisten Objekte vor dem Start des Upgrades leichter erkannt werden können, wird die Vorabprüfung `schemaInconsistencyCheck` durchgeführt, um sicherzustellen, dass alle Metadatenobjekte des Datenwörterbuchs synchron sind. Wenn verwaiste Metadatenobjekte

erkannt werden, wird das Upgrade nicht fortgesetzt. Bereinigen Sie diese verwaisten Metadatenobjekte, um mit dem Upgrade fortzufahren.

Wenn Sie bei dieser Vorabprüfung auf Fehler stoßen, wenden Sie sich an den AWS Support, um die Behebung der Metadaten-Inkonsistenz zu beantragen. Alternativ können Sie das Upgrade erneut versuchen, indem Sie einen logischen Dump durchführen und dann in einem neuen DB-Cluster von Aurora-MySQL-Version 3 wiederherstellen.

Beispielausgabe:

```
{
  "id": "schemaInconsistencyCheck",
  "title": "Schema inconsistencies resulting from file removal or corruption",
  "status": "OK",
  "description": "Error: Following tables show signs that either table datadir
 directory or frm file was removed/corrupted. Please check server logs, examine
 datadir to detect the issue and fix it before upgrade",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.schemaInconsistencyCheck_failure",
        "description": "present in INFORMATION_SCHEMA's INNODB_SYS_TABLES table but
 missing from TABLES table"
      }
  ]
}
```

Die Vorabprüfung meldet, dass die Tabelle `test.schemaInconsistencyCheck_failure` inkonsistente Metadaten enthält. In diesem Fall ist die Tabelle in den InnoDB-Speicher-Engine-Metadaten (`information_schema.INNODB_SYS_TABLES`) vorhanden, aber nicht in den MySQL-Metadaten (`information_schema.TABLES`).

Aurora-MySQL-Vorabprüfungen, die Fehler melden

Die folgenden Vorabprüfungen sind spezifisch für Aurora MySQL:

- auroraCheckDDLRecovery
- auroraCheckRdsUpgradePrechecksTable
- auroraFODUpgradeCheck
- auroraGetDanglingFulltextIndex

- auroraUpgradeCheckForDatafilePathInconsistency
- auroraUpgradeCheckForFtsSpaceIdZero
- auroraUpgradeCheckForIncompleteXATransactions
- auroraUpgradeCheckForInstanceLimit
- auroraUpgradeCheckForInternalUsers
- auroraUpgradeCheckForInvalidUtf8mb3CharacterStringInViews
- auroraUpgradeCheckForInvalidUtf8mb3ColumnComments
- auroraUpgradeCheckForInvalidUtf8mb3IndexComments
- auroraUpgradeCheckForInvalidUtf8mb3TableComments
- auroraUpgradeCheckForMasterUser
- auroraUpgradeCheckForPrefixIndexOnGeometryColumns
- auroraUpgradeCheckForSpecialCharactersInProcedures
- auroraUpgradeCheckForSysSchemaObjectTypeMismatch
- auroraUpgradeCheckForViewColumnNameLength
- auroraUpgradeCheckIndexLengthLimitOnTinyColumns
- auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable

auroraCheckDDLRecovery

Stufe der Vorabprüfung: Fehler

Prüfen auf Artefakte im Zusammenhang mit der Aurora-Funktion für die DDL-Wiederherstellung

Im Rahmen der Wiederherstellungsimplementierung der Data Definition Language (DDL) in Aurora MySQL werden Metadaten zu Inflight-DDL-Anweisungen in den Tabellen `ddl_log_md_table` und `ddl_log_table` im `mysql`-Schema verwaltet. Auroras Implementierung der DDL-Wiederherstellung wird ab Version 3 nicht unterstützt, da die Funktionalität Teil der neuen Implementierung des Atomic Data Dictionary in MySQL 8.0 ist. Wenn während der Kompatibilitätsprüfungen DDL-Anweisungen ausgeführt werden, schlägt diese Vorabprüfung möglicherweise fehl. Wir empfehlen, das Upgrade auszuprobieren, während keine DDL-Anweisungen ausgeführt werden.

Wenn diese Vorabprüfung fehlschlägt, ohne dass DDL-Anweisungen ausgeführt werden, wenden Sie sich an den AWS Support, um die Behebung der Metadaten-Inkonsistenz zu beantragen.

Alternativ können Sie das Upgrade erneut versuchen, indem Sie einen logischen Dump durchführen und dann in einem neuen DB-Cluster von Aurora-MySQL-Version 3 wiederherstellen.

Wenn DDL-Anweisungen ausgeführt werden, zeigt die Ausgabe der Vorabprüfung folgende Meldung an:

```
“There are DDL statements in process. Please allow DDL statements to finish before
 upgrading.”
```

Beispielausgabe:

```
{
  "id": "auroraCheckDDLRecovery",
  "title": "Check for artifacts related to Aurora DDL recovery feature",
  "status": "OK",
  "description": "Aurora implementation of DDL recovery is not supported from 3.x
 onwards. This check verifies that the database do not have artifacts realted to the
 feature",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_md_table",
        "description": "Table mysql.ddl_log_md_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "mysql.ddl_log_table",
        "description": "Table mysql.ddl_log_table is not empty. Your database has
 pending DDL recovery operations. Reachout to AWS support for assistance."
      },
      {
        "level": "Error",
        "dbObject": "information_schema.processlist",
        "description": "There are DDL statements in process. Please allow DDL
 statements to finish before upgrading."
      }
  ]
}
```

Die Vorabprüfung hat einen Fehler zurückgegeben, da eine Inflight-DDL gleichzeitig mit den Kompatibilitätsprüfungen ausgeführt wurde. Wir empfehlen, dass Sie das Upgrade erneut versuchen, ohne dass DDLs ausgeführt werden.

auroraCheckRdsUpgradePrechecksTable

Stufe der Vorabprüfung: Fehler

Prüfen der Existenz der Tabelle **`mysql.rds_upgrade_prechecks`**

Dies ist eine rein interne Vorabprüfung, die vom RDS-Service durchgeführt wird. Alle Fehler werden beim Upgrade automatisch behandelt und können problemlos ignoriert werden.

Wenn Sie bei dieser Vorabprüfung auf Fehler stoßen, wenden Sie sich an den AWS Support, um die Behebung der Metadaten-Inkonsistenz zu beantragen. Alternativ können Sie das Upgrade erneut versuchen, indem Sie einen logischen Dump durchführen und dann in einem neuen DB-Cluster von Aurora-MySQL-Version 3 wiederherstellen.

```
{
  "id": "auroraCheckRdsUpgradePrechecksTable",
  "title": "Check existence of mysql.rds_upgrade_prechecks table",
  "status": "OK",
  "detectedProblems": []
}
```

auroraFODUpgradeCheck

Stufe der Vorabprüfung: Fehler

Prüfen auf Artefakte im Zusammenhang mit der Aurora-Funktion für schnelle DDL

Die Optimierung für schnelle DDL wurde im Labormodus für Aurora-MySQL-Version 2 eingeführt, um die Effizienz einiger DDL-Operationen zu verbessern. In Aurora-MySQL-Version 3 wurde der Labormodus entfernt und die Implementierung schneller DDL wurde durch die MySQL-8.0-Funktion namens Sofortige DDL ersetzt.

Vor dem Upgrade auf Aurora-MySQL-Version 3 müssen alle Tabellen, die schnelle DDL im Labormodus verwenden, neu erstellt werden, indem der Befehl `OPTIMIZE TABLE` oder `ALTER TABLE ... ENGINE=InnoDB` ausgeführt wird, um die Kompatibilität mit Aurora-MySQL-Version 3 sicherzustellen.

Diese Vorabprüfung gibt eine Liste solcher Tabellen zurück. Nachdem die zurückgegebenen Tabellen neu erstellt wurden, können Sie das Upgrade erneut versuchen.

Beispielausgabe:

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "description": "Aurora fast DDL is not supported from 3.x onwards. This check
 verifies that the database does not have artifacts related to the feature",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/AuroraMySQL.Managing.FastDDL.html#AuroraMySQL.Managing.FastDDL-v2",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.test",
        "description": "Your table has pending Aurora fast DDL operations. Run
 'OPTIMIZE TABLE <table name>' for the table to apply all the pending DDL updates.
 Then try the upgrade again."
      }
  ]
}
```

Die Vorabprüfung meldet, dass in der Tabelle `test.test` ausstehende Operationen für schnelle DDL vorhanden sind.

Damit das Upgrade fortgesetzt werden kann, können Sie die Tabelle neu erstellen und dann das Upgrade erneut versuchen.

> **Note**
>
> Bevor Sie Tablespaces neu erstellen, finden Sie in der MySQL-Dokumentation unter Online-DDL-Operationen Informationen zu den Auswirkungen von Sperren und Datenverschiebungen auf Vordergrundtransaktionen.

```
mysql> optimize table test.test;
+-----------+----------+----------
+-------------------------------------------------------------------+
| Table     | Op       | Msg_type | Msg_text
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
```

```
| test.test | optimize | note     | Table does not support optimize, doing recreate
 + analyze instead |
| test.test | optimize | status   | OK
                  |
+-----------+----------+----------
+-------------------------------------------------------------------+
2 rows in set (0.04 sec)
```

Nach dem Neuerstellen der Tabelle ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraFODUpgradeCheck",
  "title": "Check for artifacts related to Aurora fast DDL feature",
  "status": "OK",
  "detectedProblems": []
}
```

auroraGetDanglingFulltextIndex

Stufe der Vorabprüfung: Fehler

Tabellen mit hängender **FULLTEXT**-Indexreferenz

Vor MySQL 5.6.26 war es möglich, dass die versteckten Spalten `FTS_DOC_ID` und `FTS_DOC_ID_INDEX` nach dem Löschen eines Volltextsuchindex verwaisen würden. Weitere Informationen finden Sie unter Fehler #76012.

Wenn Sie Tabellen haben, die in früheren Versionen von MySQL erstellt wurden, bei denen dies aufgetreten ist, kann dies dazu führen, dass Upgrades auf Aurora-MySQL-Version 3 fehlschlagen. Diese Vorabprüfung stellt vor dem Upgrade auf MySQL 8.0 sicher, dass in Ihrem DB-Cluster keine solchen verwaisten oder „hängenden“ Volltextindizes existieren. Wenn diese Vorabprüfung fehlschlägt, erstellen Sie alle Tabellen neu, die solche hängenden Volltextindizes enthalten.

Beispielausgabe:

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "description": "Error: The following tables contain dangling FULLTEXT index which
 is not supported. It is recommended to rebuild the table before upgrade.",
```

```
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.table_with_fts_index",
        "description": "Table `test.table_with_fts_index` contains dangling FULLTEXT
 index. Kindly recreate the table before upgrade."
      }
  ]
},
```

Die Vorabprüfung meldet einen Fehler für die Tabelle `test.table_with_fts_index`, da sie einen hängenden Volltextindex enthält. Damit das Upgrade fortgesetzt werden kann, müssen Sie die Tabelle neu erstellen, um die Hilfstabellen für den Volltextindex zu bereinigen. Verwenden Sie `OPTIMIZE TABLE test.table_with_fts_index` oder `ALTER TABLE test.table_with_fts_index, ENGINE=INNODB`.

Nach dem Neuerstellen der Tabelle ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraGetDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForDatafilePathInconsistency

Stufe der Vorabprüfung: Fehler

Prüfen auf Inkonsistenzen im Zusammenhang mit dem **`ibd`**-Dateipfad

Diese Vorabprüfung gilt nur für Aurora-MySQL-Version 3.03.0 und niedriger. Wenn Sie bei dieser Vorabprüfung auf einen Fehler stoßen, versuchen Sie ein Upgrade auf Aurora-MySQL-Version 3.04 oder höher.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForDatafilePathInconsistency",
  "title": "Check for inconsistency related to ibd file path.",
  "status": "OK",
```

```
  "detectedProblems": []
}
```

auroraUpgradeCheckForFtsSpaceIdZero

Stufe der Vorabprüfung: Fehler

Prüfen auf einen Volltextindex mit der Space-ID als Null

Wenn Sie in MySQL einer InnoDB-Tabelle einen Volltextindex hinzufügen, werden eine Reihe von zusätzlichen Index-Tablespaces erstellt. Aufgrund eines Fehlers in früheren Versionen von MySQL, der in Version 5.6.20 behoben wurde, war es möglich, dass diese zusätzlichen Indextabellen im System-Tablespace und nicht in ihrem eigenen InnoDB-Tablespace erstellt wurden.

Wenn solche zusätzlichen Tablespaces existieren, schlägt das Upgrade fehl. Erstellen Sie die im Fehler bei der Vorabprüfung genannten Volltextindizes neu und versuchen Sie dann das Upgrade erneut.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForFtsSpaceIdZero",
  "title": "Check for fulltext index with space id as zero",
  "status": "OK",
  "description": "The auxiliary tables of FTS indexes on the table are created
 in system table-space. Due to this DDL queries executed on MySQL8.0 shall cause
 database unavailability. To avoid that, drop and recreate all the FTS indexes on
 the table or rebuild the table using ALTER TABLE query before the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.fts_space_zero_check",
        "description": " The auxiliary tables of FTS indexes on the table
 'test.fts_space_zero_check' are created in system table-space due to https://
bugs.mysql.com/bug.php?id=72132. In MySQL8.0, DDL queries executed on this table
 shall cause database unavailability. To avoid that, drop and recreate all the
 FTS indexes on the table or rebuild the table using ALTER TABLE query before the
 upgrade."
      }
  ]
},
```

Die Vorabprüfung meldet einen Fehler für die Tabelle `test.fts_space_zero_check`, da sie zusätzliche Tabellen für Volltextsuchen (Full-Text Search, FTS) im System-Tablespace enthält.

Nachdem Sie die mit dieser Tabelle verknüpften FTS-Indizes gelöscht und neu erstellt haben, ist die Vorabprüfung erfolgreich.

> **Note**
>
> Bevor Sie Tablespaces neu erstellen, finden Sie in der MySQL-Dokumentation unter Partitionierungsoperationen Informationen zu den Auswirkungen von Sperren und Datenverschiebungen auf Vordergrundtransaktionen.

```
{
 "id": "auroraUpgradeCheckForFtsSpaceIdZero",
 "title": "Check for fulltext index with space id as zero",
 "status": "OK",
 "detectedProblems": []
}
```

auroraUpgradeCheckForIncompleteXATransactions

Stufe der Vorabprüfung: Fehler

Prüfen auf XA-Transaktionen im vorbereiteten Zustand

Während des Upgrade-Prozesses für die Hauptversion ist es wichtig, dass die DB-Instance von Aurora-MySQL-Version 2 ordnungsgemäß heruntergefahren wird. Dadurch wird sichergestellt, dass alle Transaktionen festgeschrieben oder rückgängig gemacht werden und dass InnoDB alle Undo-Protokolleinträge bereinigt hat. Da ein Transaktions-Rollback notwendig ist, kann das ordnungsgemäße Herunterfahren blockiert werden, wenn Ihre Datenbank XA-Transaktionen im vorbereiteten Zustand enthält. Aus diesem Grund kann das Upgrade nicht fortgesetzt werden, wenn vorbereitete XA-Transaktionen erkannt werden, bis Sie Maßnahmen ergreifen, um sie entweder festzuschreiben oder rückgängig zu machen.

Weitere Informationen zum Auffinden von XA-Transaktionen im vorbereiteten Zustand mithilfe von `XA RECOVER` finden Sie unter XA-Transaktions-SQL-Anweisungen in der MySQL-Dokumentation. Weitere Informationen zu XA-Transaktionszuständen finden Sie unter XA-Transaktionszustände in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "description": "Your cluster currently has XA transactions in the prepared state.
 To proceed with the upgrade, commit or rollback these transactions.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your cluster currently has XA transactions in the prepared
 state. To proceed with the upgrade, commit or rollback these transactions."
      }
  ]
}
```

Diese Vorabprüfung meldet einen Fehler, da sich Transaktionen im vorbereiteten Zustand befinden, die entweder festgeschrieben oder rückgängig gemacht werden sollten.

Nachdem Sie sich bei der Datenbank angemeldet haben, können Sie in der Tabelle information_schema.innodb_trx und in der `XA RECOVER`-Ausgabe nach weiteren Informationen suchen.

> **Important**
>
> Bevor Sie eine Transaktion festschreiben oder rückgängig machen, empfehlen wir Ihnen, die MySQL-Dokumentation und Ihre Anwendungsanforderungen zu lesen.

```
mysql> select trx_started,
    trx_mysql_thread_id,
    trx_id,trx_state,
    trx_operation_state,
    trx_rows_modified,
    trx_rows_locked
from
    information_schema.innodb_trx;
+--------------------+--------------------+---------+-----------
+--------------------+------------------+-----------------+
```

```
| trx_started         | trx_mysql_thread_id | trx_id  | trx_state |
 trx_operation_state | trx_rows_modified | trx_rows_locked |
+---------------------+---------------------+---------+-----------
+---------------------+-------------------+-----------------+
| 2024-08-12 01:09:39 |                   0 | 2849470 | RUNNING   | NULL
    |                 1 |                 0 |
+---------------------+---------------------+---------+-----------
+---------------------+-------------------+-----------------+
1 row in set (0.00 sec)

mysql> xa recover;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        1 |            6 |            0 | xatest |
+----------+--------------+--------------+--------+
1 row in set (0.00 sec)
```

In diesem Fall machen wir die vorbereitete Transaktion rückgängig.

```
mysql> XA ROLLBACK 'xatest';
Query OK, 0 rows affected (0.00 sec)
v
mysql> xa recover;
Empty set (0.00 sec)
```

Nachdem die XA-Transaktion rückgängig gemacht wurde, ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraUpgradeCheckForIncompleteXATransactions",
  "title": "Pre-checks for XA Transactions in prepared state.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForInstanceLimit

Stufe der Vorabprüfung: Fehler

Prüfen, ob das Upgrade für die aktuelle Instance-Klasse unterstützt wird

Die Ausführung eines direkten Upgrades von Aurora-MySQL-Version 2.12.0 oder 2.12.1, bei der die Schreiber-DB-Instance-Klasse db.r6i.32xlarge ist, wird derzeit nicht unterstützt. In diesem Fall

gibt die Vorabprüfung einen Fehler zurück. Damit das Upgrade fortgesetzt werden kann, können Sie Ihre DB-Instance-Klasse entweder in db.r6i.24xlarge oder kleiner ändern. Oder Sie können ein Upgrade auf Aurora-MySQL-Version 2.12.2 oder höher durchführen, wobei das direkte Upgrade auf Aurora-MySQL-Version 3 auf db.r6i.32xlarge unterstützt wird.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForInstanceLimit",
  "title": "Checks if upgrade is supported on the current instance class",
  "status": "OK",
  "description": "Upgrade from Aurora Version 2.12.0 and 2.12.1 may fail for 32.xl
 and above instance class.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Upgrade is not supported on this instance size for Aurora
 MySql Version 2.12.1. Before upgrading to Aurora MySql 3, please consider either:
 1. Changing the instance class to 24.xl or lower. -or- 2. Upgrading to patch
 version 2.12.2 or higher."
      }
  ]
},
```

Die Vorabprüfung gibt einen Fehler zurück, da die Schreiber-DB-Instance die Instance-Klasse db.r6i.32xlarge verwendet und auf Aurora-MySQL-Version 2.12.1 läuft.

auroraUpgradeCheckForInternalUsers

Stufe der Vorabprüfung: Fehler

Prüfen auf interne Benutzer von Version 8.0

Diese Vorabprüfung gilt nur für Aurora-MySQL-Version 3.03.0 und niedriger. Wenn Sie bei dieser Vorabprüfung auf einen Fehler stoßen, versuchen Sie ein Upgrade auf Aurora-MySQL-Version 3.04 oder höher.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForInternalUsers",
  "title": "Check for 8.0 internal users.",
```

```
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForInvalidUtf8mb3CharacterStringInViews

Stufe der Vorabprüfung: Fehler

Prüfen auf ungültige utf8mb3-Zeichen in einer Ansichtsdefinition

Diese Vorabprüfung identifiziert Ansichten, die Kommentare mit ungültiger `utf8mb3`-Zeichenkodierung enthalten. In MySQL 8.0 wird eine strengere Validierung auf die Zeichenkodierung in Metadaten angewendet, einschließlich Ansichtskommentaren. Wenn eine Ansichtsdefinition Zeichen enthält, die im `utf8mb3`-Zeichensatz nicht gültig sind, schlägt das Upgrade fehl.

Zum Beheben dieses Problems ändern Sie die Ansichtsdefinition so, dass alle Nicht-BMP-Zeichen entfernt oder ersetzt werden, bevor Sie versuchen, das Upgrade durchzuführen.

Beispielausgabe:

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3CharacterStringInViews",
"title": "Check for invalid utf8mb3 character string.",
"status": "OK",
"description": "Definition of following view(s) has/have invalid utf8mb3 character
 string.",
"detectedProblems": [
        {
            "level": "Error",
            "dbObject": "precheck.utf8mb3_invalid_char_view",
            "description": "Definition of view precheck.utf8mb3_invalid_char_view
 contains an invalid utf8mb3 character string. This is due to https://
bugs.mysql.com/bug.php?id=110177. To fix the inconsistency, we recommend you to
 modify the view definition to not use non-BMP characters and try the upgrade
 again."
        }
    ]
},
```

Die Vorabprüfung meldet, dass die Definition der Ansichtsdefinition `utf8mb3_invalid_char_view` ungültige `utf8mb3`-Zeichen enthält.

Zum Beheben dieses Problems identifizieren Sie die Ansicht, die die nicht unterstützten Zeichen enthält, und aktualisieren die Kommentare. Untersuchen Sie zunächst die Struktur der Ansicht und identifizieren Sie Kommentare.

```
MySQL> SHOW CREATE VIEW precheck.utf8mb3_invalid_char_view\G
*************************** 1. row ***************************
                View: utf8mb3_invalid_char_view
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`admin`@`%` SQL SECURITY
 DEFINER VIEW `utf8mb3_invalid_char_view` AS select 'This row contains a dolphin #'
 AS `message`
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set, 1 warning (0.00 sec)
```

Sobald Sie die Ansicht identifiziert haben, die den Fehler enthält, ersetzen Sie sie mithilfe der `CREATE OR REPLACE VIEW`-Anweisung.

```
MySQL> CREATE OR REPLACE VIEW precheck.utf8mb3_invalid_char_view AS select 'This
 view definition to not use non-BMP characters' AS message;
```

Nach der Aktualisierung aller Ansichtsdefinitionen, die nicht unterstützte Zeichen enthalten, ist die Vorabprüfung erfolgreich und das Upgrade kann fortgesetzt werden.

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
"title": "Check for invalid utf8mb3 column comments.",
"status": "OK",
"detectedProblems": []
}
```

auroraUpgradeCheckForInvalidUtf8mb3ColumnComments

Stufe der Vorabprüfung: Fehler

Prüfen auf ungültige utf8mb3-Spaltenkommentare

Diese Vorabprüfung identifiziert Tabellen, die Spaltenkommentare mit ungültiger `utf8mb3`-Zeichenkodierung enthalten. In MySQL 8.0 wird eine strengere Validierung auf die Zeichenkodierung in Metadaten angewendet, einschließlich Spaltenkommentaren. Wenn Spaltenkommentare Zeichen enthalten, die im utf8mb3-Zeichensatz nicht gültig sind, schlägt das Upgrade fehl.

Zum Beheben dieses Problems müssen Sie die Spaltenkommentare ändern, um alle Nicht-BMP-Zeichen zu entfernen oder zu ersetzen, bevor Sie das Upgrade durchführen. Sie können die `ALTER TABLE`-Anweisung verwenden, um die Spaltenkommentare zu aktualisieren.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "description": "Following table(s) has/have invalid utf8mb3 comments on the
 column/columns.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.t2",
        "description": "Table test.t2 has invalid utf8mb3 comments in it's column/
columns. This is due to non-BMP characters in the comment field. To fix the
 inconsistency, we recommend you to modify comment fields to not use non-BMP
 characters and try the upgrade again."
      }
  ]
}
```

Die Vorabprüfung meldet, dass die Tabelle `test.t2` ungültige `utf8mb3`-Zeichen in einem oder mehreren Spaltenkommentaren enthält, insbesondere aufgrund des Vorhandenseins von Nicht-BMP-Zeichen.

Zum Beheben dieses Problems können Sie die problematischen Spalten identifizieren und ihre Kommentare aktualisieren. Untersuchen Sie zunächst die Tabellenstruktur, um Spalten mit Kommentaren zu identifizieren:

```
mysql> SHOW CREATE TABLE test.t2\G
```

Sobald Sie die Spalten mit problematischen Kommentaren identifiziert haben, aktualisieren Sie sie mithilfe der `ALTER TABLE`-Anweisung. Zum Beispiel:

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT 'Updated
 comment without non-BMP characters';
```

Alternativ können Sie den Kommentar auch vollständig entfernen:

```
mysql> ALTER TABLE test.t2 MODIFY COLUMN column_name data_type COMMENT '';
```

Nach der Aktualisierung aller problematischen Spaltenkommentare ist die Vorabprüfung erfolgreich und das Upgrade kann fortgesetzt werden:

```
{
  "id": "auroraUpgradeCheckForInvalidUtf8mb3ColumnComments",
  "title": "Check for invalid utf8mb3 column comments.",
  "status": "OK",
  "detectedProblems": []
}
```

> **Note**
>
> Stellen Sie vor dem Ändern von Spaltenkommentaren sicher, dass Anwendungscodes oder Dokumentationen, die von diesen Kommentaren abhängen, entsprechend aktualisiert werden. Erwägen Sie für eine bessere Unicode-Unterstützung eine Migration zum utf8mb4-Zeichensatz, falls Ihre Anwendung Nicht-BMP-Zeichen benötigt.

auroraUpgradeCheckForInvalidUtf8mb3IndexComments

Stufe der Vorabprüfung: Fehler

Prüfen auf ungültige utf8mb3-Indexkommentare

Diese Vorabprüfung identifiziert Tabellen, die Indexkommentare mit ungültiger `utf8mb3`-Zeichenkodierung enthalten. In MySQL 8.0 wird eine strengere Validierung auf die Zeichenkodierung in Metadaten angewendet, einschließlich Indexkommentaren. Wenn ein Indexkommentar Zeichen enthält, die im `utf8mb3`-Zeichensatz nicht gültig sind, schlägt das Upgrade fehl.

Zum Beheben dieses Problems müssen Sie die Indexkommentare ändern, um alle Nicht-BMP-Zeichen zu entfernen oder zu ersetzen, bevor Sie das Upgrade durchführen.

Beispielausgabe:

```
{
```

```
    "id": "auroraUpgradeCheckForInvalidUtf8mb3IndexComments",
    "title": "Check for invalid utf8mb3 index comments.",
    "status": "OK",
    "description": "Following table(s) has/have invalid utf8mb3 comments on the
 index.",
    "detectedProblems": [
        {
            "level": "Error",
            "dbObject": "precheck.utf8mb3_tab_index_comment",
            "description": "Table precheck.utf8mb3_tab_index_comment has invalid
 utf8mb3 comments in it's index. This is due to https://bugs.mysql.com/bug.php?
id=110177. To fix the inconsistency, we recommend you to modify comment fields to
 not use non-BMP characters and try the upgrade again."
        }
    ]
},
```

Die Vorabprüfung meldet, dass die Tabelle `utf8mb3_tab_index_comment` ungültige `utf8mb3`-Zeichen in einem oder mehreren Spaltenkommentaren enthält, insbesondere aufgrund des Vorhandenseins von Nicht-BMP-Zeichen.

Zum Beheben dieses Problems untersuchen Sie zunächst die Tabellenstruktur, um den Index mit problematischen Kommentaren zu identifizieren.

```
MySQL> SHOW CREATE TABLE precheck.utf8mb3_tab_index_comment\G
*************************** 1. row ***************************
    Table: utf8mb3_tab_index_comment
Create Table: CREATE TABLE `utf8mb3_tab_index_comment` (
`id` int(11) DEFAULT NULL,
`name` varchar(100) DEFAULT NULL,
KEY `idx_name` (`name`) COMMENT 'Name index #'
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.01 sec)
```

Sobald Sie den Index identifiziert haben, der Kommentare enthält, die nicht unterstützte Zeichen verwenden, löschen Sie den Index und erstellen Sie ihn neu.

> **Note**
>
> Das Löschen und Neuerstellen eines Tabellenindexes kann zu Ausfallzeiten führen. Wir empfehlen, diesen Vorgang während der Wartung zu planen und durchzuführen.

```
MySQL> ALTER TABLE precheck.utf8mb3_tab_index_comment DROP INDEX idx_name;
MySQL> ALTER TABLE precheck.utf8mb3_tab_index_comment ADD INDEX idx_name(name);
```

Das folgende Beispiel zeigt eine andere Möglichkeit, den Index neu zu erstellen.

```
MySQL> ALTER TABLE utf8mb3_tab_index_comment DROP INDEX idx_name, ADD INDEX idx_name
 (name) COMMENT 'Updated comment without non-BMP characters';
```

Nachdem Sie alle nicht unterstützten Indexkommentare entfernt oder aktualisiert haben, ist die Vorabprüfung erfolgreich und das Upgrade kann fortgesetzt werden.

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3IndexComments",
"title": "Check for invalid utf8mb3 index comments.",
"status": "OK",
"detectedProblems": []
},
```

auroraUpgradeCheckForInvalidUtf8mb3TableComments

Stufe der Vorabprüfung: Fehler

Prüfen auf ungültige utf8mb3-Zeichen in einer Tabellendefinition

Diese Vorabprüfung identifiziert Tabellen, die Kommentare mit ungültiger `utf8mb3`-Zeichenkodierung enthalten. In MySQL 8.0 wird eine strengere Validierung auf die Zeichenkodierung in Metadaten angewendet, einschließlich Tabellenkommentaren. Wenn ein Tabellenkommentar Zeichen enthält, die im `utf8mb3`-Zeichensatz nicht gültig sind, schlägt das Upgrade fehl.

Zum Beheben dieses Problems müssen Sie die Tabellenkommentare ändern, um alle Nicht-BMP-Zeichen zu entfernen oder zu ersetzen, bevor Sie das Upgrade durchführen.

Beispielausgabe:

```
{
    "id": "auroraUpgradeCheckForInvalidUtf8mb3TableComments",
    "title": "Check for invalid utf8mb3 table comments.",
    "status": "OK",
```

```
    "description": "Following table(s) has/have invalid utf8mb3 comments.",
    "detectedProblems": [
        {
            "level": "Error",
            "dbObject": "precheck.utf8mb3_table_with_comment",
            "description": "Table precheck.utf8mb3_table_with_comment has invalid
 utf8mb3 comments. This is due to https://bugs.mysql.com/bug.php?id=110177. To fix
 the inconsistency, we recommend you to modify comment fields to not use non-BMP
 characters and try the upgrade again."
        }

    ]
},
```

Die Vorabprüfung meldet ungültige `utf8mb3`-Kommentare, die für die `utf8mb3_table_with_comment`-Tabellen in der Testdatenbank definiert wurden.

Zum Beheben dieses Problems identifizieren Sie die Tabelle, die nicht unterstützte Zeichen enthält, und aktualisieren die Kommentare. Untersuchen Sie zunächst die Struktur der Tabelle und identifizieren Sie die Kommentare.

```
MySQL> SHOW CREATE TABLE precheck.utf8mb3_table_with_comment\G
*************************** 1. row ***************************
    Table: utf8mb3_table_with_comment
Create Table: CREATE TABLE `utf8mb3_table_with_comment` (
`id` int(11) NOT NULL,
`name` varchar(100) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='This table comment contains flag ##'
1 row in set (0.00 sec)
```

Sobald Sie Tabellenkommentare identifiziert haben, die nicht unterstützte Zeichen enthalten, aktualisieren Sie die Kommentare mit der `ALTER TABLE`-Anweisung.

```
MySQL> ALTER TABLE precheck.utf8mb3_table_with_comment COMMENT='Updated comment
 without non-BMP characters';
```

Alternativ können Sie den Kommentar auch entfernen.

```
MySQL> ALTER TABLE precheck.utf8mb3_table_with_comment COMMENT='';
```

Nachdem Sie alle nicht unterstützten Zeichen aus allen Tabellenkommentaren entfernt haben, ist die Vorabprüfung erfolgreich.

```
{
"id": "auroraUpgradeCheckForInvalidUtf8mb3TableComments",
"title": "Check for invalid utf8mb3 table comments.",
"status": "OK",
"detectedProblems": []
},
```

auroraUpgradeCheckForMasterUser

Stufe der Vorabprüfung: Fehler

Prüfen, ob ein RDS-Master-Benutzer existiert

MySQL 8 hat ein neues Berechtigungsmodell mit Unterstützung für Rollen und dynamische Berechtigungen hinzugefügt, um die Verwaltung von Berechtigungen einfacher und feiner abgestuft zu gestalten. Im Rahmen dieser Änderung hat Aurora MySQL die neue `rds_superuser_role` eingeführt, die dem Master-Benutzer der Datenbank beim Upgrade von Aurora-MySQL-Version 2 auf Version 3 automatisch gewährt wird.

Weitere Informationen zu den Rollen und Berechtigungen, die dem Master-Benutzer in Aurora MySQL zugewiesen sind, finden Sie unter Berechtigungen von Hauptbenutzerkonten. Weitere Informationen zum rollenbasierten Berechtigungsmodell in Aurora-MySQL-Version 3 finden Sie unter Rollenbasiertes Berechtigungsmodell.

Diese Vorabprüfung stellt sicher, dass der Master-Benutzer in der Datenbank vorhanden ist. Wenn der Master-Benutzer nicht vorhanden ist, schlägt die Vorabprüfung fehl. Damit das Upgrade fortgesetzt werden kann, müssen Sie den Master-Benutzer neu erstellen, indem Sie das Master-Benutzerpasswort zurücksetzen oder den Benutzer manuell erstellen. Versuchen Sie anschließend das Upgrade erneut. Weitere Informationen zum Zurücksetzen des Master-Benutzerpassworts finden Sie unter Ändern des Passworts für den Master-Benutzer der Datenbank.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  "title": "Check if master user exists",
```

```
  "status": "OK",
  "description": "Throws error if master user has been dropped!",
  "documentationLink": "https://docs.aws.amazon.com/AmazonRDS/latest/
AuroraUserGuide/UsingWithRDS.MasterAccounts.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "all",
        "description": "Your Master User on host '%' has been dropped. To proceed
 with the upgrade, recreate the master user `reinvent` on default host '%'"
      }
  ]
}
```

Nachdem Sie Ihr Master-Benutzerpasswort zurückgesetzt haben, ist die Vorabprüfung erfolgreich und Sie können das Upgrade erneut versuchen.

Im folgenden Beispiel wird die AWS CLI zum Zurücksetzen des Passworts verwendet. Passwortänderungen werden sofort übernommen.

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --master-user-password my-new-password
```

Anschließend ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraUpgradeCheckForMasterUser",
  title": "Check if master user exists",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForPrefixIndexOnGeometryColumns

Stufe der Vorabprüfung: Fehler

Prüfen auf Geometriespalten in Präfixindizes

Ab MySQL 8.0.12 können Sie mit dem Datentyp GEOMETRY keinen Präfix-Index mehr für eine Spalte erstellen. Weitere Informationen finden Sie unter WL#11808.

Wenn solche Indizes existieren, schlägt Ihr Upgrade fehl. Zum Beheben des Problems löschen Sie die Präfix-GEOMETRY-Indizes in den Tabellen, die im Fehler bei der Vorabprüfung genannt wurden.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "description": "Consider dropping the prefix Indexes of geometry columns and
 restart the upgrade.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.geom_index_prefix",
        "description": "Table `test`.`geom_index_prefix` has an index `LatLon` on
 geometry column/s. Mysql 8.0 does not support this type of index on a geometry
 column https://dev.mysql.com/worklog/task/?id=11808. To upgrade to MySQL 8.0, Run
 'DROP INDEX `LatLon` ON `test`.`geom_index_prefix`;"
      }
  ]
}
```

Die Vorabprüfung meldet einen Fehler, da die Tabelle `test.geom_index_prefix` einen Index mit einem Präfix in einer GEOMETRY-Spalte enthält.

Nachdem Sie diesen Index gelöscht haben, ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraUpgradeCheckForPrefixIndexOnGeometryColumns",
  "title": "Check for geometry columns on prefix indexs",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForSpecialCharactersInProcedures

Stufe der Vorabprüfung: Fehler

Prüfen auf Inkonsistenzen im Zusammenhang mit Sonderzeichen in gespeicherten Prozeduren

Vor MySQL 8.0 entsprachen Datenbanknamen, Tabellennamen und andere Objekte Dateien im Datenverzeichnis, also dateibasierten Metadaten. Im Rahmen des Upgrades auf MySQL 8.0 werden alle Datenbankobjekte zu den neuen internen Datenwörterbuchtabellen migriert, die im `mysql`-Schema gespeichert sind, um das neu implementierte Atomic Data Dictionary zu unterstützen. Im Rahmen der Migration von gespeicherten Prozeduren werden Prozedurdefinition und -körper jeder Prozedur validiert, wenn sie in das neue Datenwörterbuch aufgenommen werden.

Vor MySQL 8 war es in einigen Fällen möglich, gespeicherte Routinen zu erstellen oder Prozeduren, die Sonderzeichen enthielten, direkt in die Tabelle `mysql.proc` einzufügen. Sie konnten beispielsweise eine gespeicherte Prozedur erstellen, die einen Kommentar mit dem nicht konformen, geschützten Leerzeichen `\xa0` enthielt. Wenn solche Prozeduren gefunden werden, schlägt das Upgrade fehl.

Diese Vorabprüfung stellt sicher, dass die Körper und Definitionen Ihrer gespeicherten Prozeduren keine derartigen Zeichen enthalten. Damit das Upgrade fortgesetzt werden kann, müssen Sie diese gespeicherten Prozeduren ohne versteckte Zeichen oder Sonderzeichen neu erstellen.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "description": "Following procedure(s) has special characters inconsistency.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "information_schema.routines",
        "description": "Data Dictionary Metadata is inconsistent for the procedure
 `get_version_proc` in the database `test` due to usage of special characters in
 procedure body. To avoid that, drop and recreate the procedure without any special
 characters before proceeding with the Upgrade."
      }
  ]
}
```

Die Vorabprüfung meldet, dass der DB-Cluster eine Prozedur namens `get_version_proc` in der `test`-Datenbank enthält, deren Prozedurkörper Sonderzeichen enthält.

Nach dem Löschen und Neuerstellen der gespeicherten Prozedur ist die Vorabprüfung erfolgreich, sodass das Upgrade fortgesetzt werden kann.

```
{
  "id": "auroraUpgradeCheckForSpecialCharactersInProcedures",
  "title": "Check for inconsistency related to special characters in stored
 procedures.",
  "status": "OK",
  "detectedProblems": []
},
```

auroraUpgradeCheckForSysSchemaObjectTypeMismatch

Stufe der Vorabprüfung: Fehler

Prüfen der Nichtübereinstimmung von Objekttypen im **`sys`**-Schema

Das sys-Schema besteht aus einer Reihe von Objekten und Ansichten in einer MySQL-Datenbank und unterstützt Benutzer bei der Fehleranalyse, Optimierung und Überwachung ihrer DB-Instances. Wenn ein Hauptversions-Upgrade von Aurora-MySQL-Version 2 auf Version 3 durchgeführt wird, werden die `sys`-Schemaansichten neu erstellt und auf die neuen Definitionen von Aurora-MySQL-Version 3 aktualisiert.

Im Rahmen des Upgrades schlägt dieses fehl, wenn Objekte im `sys`-Schema mit Speicher-Engines (`sys_config/BASE TABLE` in INFORMATION_SCHEMA.TABLES) statt mit Ansichten definiert sind. Solche Tabellen finden Sie in der Tabelle `information_schema.tables`. Dies ist kein erwartetes Verhalten, kann aber in einigen Fällen aufgrund eines Benutzerfehlers auftreten.

Diese Vorabprüfung validiert alle `sys`-Schemaobjekte, um sicherzustellen, dass sie die richtigen Tabellendefinitionen verwenden und dass Ansichten nicht fälschlicherweise als InnoDB- oder MyISAM-Tabellen definiert sind. Zum Beheben des Problems korrigieren Sie alle zurückgegebenen Objekte manuell, indem Sie sie umbenennen oder löschen. Versuchen Sie anschließend das Upgrade erneut.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
```

```
  "description": "Database contains objects with type mismatch for sys schema.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "sys.waits_global_by_latency",
        "description": "Your object sys.waits_global_by_latency has a type mismatch.
 To fix the inconsistency we recommend to rename or remove the object before
 upgrading (use RENAME TABLE command). "
      }
  ]
}
```

Die Vorabprüfung meldet, dass die Ansicht sys.waits_global_by_latency im `sys`-Schema einen Typkonflikt aufweist, der die Fortsetzung des Upgrades verhindert.

Nachdem Sie sich bei der DB-Instance angemeldet haben, können Sie sehen, dass dieses Objekt als InnoDB-Tabelle definiert ist, obwohl es eine Ansicht sein sollte.

```
mysql> show create table sys.waits_global_by_latency\G
*************************** 1. row ***************************
       Table: waits_global_by_latency
Create Table: CREATE TABLE `waits_global_by_latency` (
  `events` varchar(128) DEFAULT NULL,
  `total` bigint(20) unsigned DEFAULT NULL,
  `total_latency` text,
  `avg_latency` text,
  `max_latency` text
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

Zum Beheben dieses Problems können wir die Ansicht entweder löschen und mit der richtigen Definition neu erstellen oder sie umbenennen. Während des Upgrade-Vorgangs wird sie automatisch mit der richtigen Tabellendefinition erstellt.

```
mysql> RENAME TABLE sys.waits_global_by_latency to sys.waits_global_by_latency_old;
Query OK, 0 rows affected (0.01 sec)
```

Nachdem dies durchgeführt wurde, ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraUpgradeCheckForSysSchemaObjectTypeMismatch",
```

```
  "title": "Check object type mismatch for sys schema.",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckForViewColumnNameLength

Stufe der Vorabprüfung: Fehler

Prüfen der Obergrenze für einen Spaltennamen in einer Ansicht

Die maximal zulässige Länge eines Spaltennamens in MySQL beträgt 64 Zeichen. Vor MySQL 8.0 war es in einigen Fällen möglich, eine Ansicht mit einem Spaltennamen mit mehr als 64 Zeichen zu erstellen. Wenn solche Ansichten in Ihrer Datenbank-Instance existieren, wird ein Fehler bei der Vorabprüfung zurückgegeben und das Upgrade schlägt fehl. Damit das Upgrade fortgesetzt werden kann, müssen Sie die fraglichen Ansichten neu erstellen und dabei sicherstellen, dass ihre Spaltenlänge weniger als 64 Zeichen beträgt. Versuchen Sie anschließend das Upgrade erneut.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
  "description": "Following view(s) has column(s) with length greater than 64.",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject":
 "test.colname_view_test.col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pa
        "description": "View `test`.`colname_view_test`has column
 `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad` with
 invalid column name length. To avoid Upgrade errors, view should be altered by
 renaming the column name so that its length is not 0 and doesn't exceed 64."
      }
  ]
}
```

Die Vorabprüfung meldet, dass die Ansicht `test.colname_view_test` eine `col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad`-Spalte enthält, die die maximal zulässige Spaltenlänge von 64 Zeichen überschreitet.

Wenn wir uns die Ansichtsdefinition ansehen, können wir die problematische Spalte sehen.

```
mysql> desc `test`.`colname_view_test`;
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| Field                                                             | Type        |
 Null | Key | Default | Extra |
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
| col1                                                              | varchar(20) |
 YES  |     | NULL    |       |
| col2_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad_pad | int(11)     |
 YES  |     | NULL    |       |
+-------------------------------------------------------------------+-------------
+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Damit das Upgrade fortgesetzt werden kann, müssen Sie die Ansicht neu erstellen und dabei sicherstellen, dass die Spaltenlänge 64 Zeichen nicht überschreitet.

```
mysql> drop view `test`.`colname_view_test`;
Query OK, 0 rows affected (0.01 sec)

mysql> create view `test`.`colname_view_test`(col1, col2_nopad) as select inf,
 fodcol from test;
Query OK, 0 rows affected (0.01 sec)

mysql> desc `test`.`colname_view_test`;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| col1       | varchar(20) | YES  |     | NULL    |       |
| col2_nopad | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Nachdem dies durchgeführt wurde, ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraUpgradeCheckForViewColumnNameLength",
  "title": "Check for upperbound limit related to column name in view.",
  "status": "OK",
```

```
  "detectedProblems": []
}
```

auroraUpgradeCheckIndexLengthLimitOnTinyColumns

Stufe der Vorabprüfung: Fehler

Prüfen auf Tabellen mit Indizes, die für kleine Spalten mit einer Präfixlänge von mehr als 255 Byte definiert sind

Wenn Sie einen Index für eine Spalte mit einem binären Datentyp in MySQL erstellen, müssen Sie der Indexdefinition eine Präfixlänge hinzufügen. Vor MySQL 8.0 war es in einigen Fällen möglich, eine Präfixlänge anzugeben, die größer als die maximal zulässige Größe solcher Datentypen war. Ein Beispiel dafür sind `TINYTEXT`- und `TINYBLOB`-Spalten, bei denen die maximal zulässige Datengröße 255 Byte beträgt, aber auch größere Indexpräfixe zulässig waren. Weitere Informationen finden Sie unter Limits für InnoDB in der MySQL-Dokumentation.

Wenn diese Vorabprüfung fehlschlägt, löschen Sie den problematischen Index oder reduzieren Sie die Präfixlänge von `TINYTEXT`- und `TINYBLOB`-Spalten des Indexes auf weniger als 255 Byte. Versuchen Sie anschließend das Upgrade erneut.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "description": "Prefix length of the indexes defined on tiny columns cannot
 exceed 255 bytes. With utf8mb4 char set, this limits the prefix length supported
 upto 63 characters only. A larger prefix length was allowed in MySQL5.7 using
 innodb_large_prefix parameter. This parameter is deprecated in MySQL 8.0.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/innodb-limits.html,
 https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.tintxt_prefixed_index.col1",
        "description": "Index 'PRIMARY' on tinytext/tinyblob column `col1` of table
 `test.tintxt_prefixed_index` is defined with prefix length exceeding 255 bytes.
 Reduce the prefix length to <=255 bytes depending on character set used. For
 utf8mb4, it should be <=63."
```

```
      }
  ]
}
```

Die Vorabprüfung meldet einen Fehler für die Tabelle `test.tintxt_prefixed_index`, da sie über einen `PRIMARY`-Index verfügt, dessen Präfix für eine TINYTEXT- oder TINYBLOB-Spalte größer als 255 Byte ist.

Wenn wir uns die Definition für diese Tabelle ansehen, können wir sehen, dass der Primärschlüssel einen Präfix von 65 für die `TINYTEXT`-Spalte `col1` aufweist. Da die Tabelle mit dem `utf8mb4`-Zeichensatz definiert wird, der 4 Byte pro Zeichen speichert, überschreitet das Präfix die 255-Byte-Grenze.

```
mysql> show create table `test`.`tintxt_prefixed_index`\G
*************************** 1. row ***************************
       Table: tintxt_prefixed_index
Create Table: CREATE TABLE `tintxt_prefixed_index` (
  `col1` tinytext NOT NULL,
  `col2` tinytext,
  `col_id` tinytext,
  PRIMARY KEY (`col1`(65))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC
1 row in set (0.00 sec)
```

Wenn Sie das Indexpräfix auf 63 ändern, während Sie den `utf8mb4`-Zeichensatz verwenden, kann das Upgrade fortgesetzt werden.

```
mysql> alter table `test`.`tintxt_prefixed_index` drop PRIMARY KEY, ADD  PRIMARY KEY
 (`col1`(63));
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Nachdem dies durchgeführt wurde, ist die Vorabprüfung erfolgreich.

```
{
  "id": "auroraUpgradeCheckIndexLengthLimitOnTinyColumns",
  "title": "Check for the tables with indexes defined with prefix length greater
 than 255 bytes on tiny columns",
  "status": "OK",
  "detectedProblems": []
}
```

auroraUpgradeCheckMissingInnodbMetadataForMysqlHostTable

Stufe der Vorabprüfung: Fehler

Prüfen der Inkonsistenz fehlender InnoDB-Metadaten für die Tabelle **`mysql.host`**

Dies ist eine rein interne Vorabprüfung, die vom RDS-Service durchgeführt wird. Alle Fehler werden beim Upgrade automatisch behandelt und können problemlos ignoriert werden.

Wenn Sie bei dieser Vorabprüfung auf Fehler stoßen, wenden Sie sich an den AWS Support, um die Behebung der Metadaten-Inkonsistenz zu beantragen. Alternativ können Sie das Upgrade erneut versuchen, indem Sie einen logischen Dump durchführen und dann in einem neuen DB-Cluster von Aurora-MySQL-Version 3 wiederherstellen.

Warnungen

Die folgenden Vorabprüfungen generieren Warnungen, wenn die Vorabprüfung fehlschlägt, aber das Upgrade kann fortgesetzt werden.

Themen

- MySQL-Vorabprüfungen, die Warnungen melden
- Aurora-MySQL-Vorabprüfungen, die Warnungen melden

MySQL-Vorabprüfungen, die Warnungen melden

Die folgenden Vorabprüfungen stammen von Community MySQL:

- defaultAuthenticationPlugin
- maxdbFlagCheck
- mysqlDollarSignNameCheck
- reservedKeywordsCheck
- utf8mb3Check
- zeroDatesCheck

defaultAuthenticationPlugin

Stufe der Vorabprüfung: Warnung

Überlegungen zum neuen Standard-Authentifizierungs-Plugin

In MySQL 8.0 wurde das `caching_sha2_password`-Authentifizierungs-Plugin eingeführt, das eine sicherere Passwortverschlüsselung und eine bessere Leistung als das veraltete `mysql_native_password`-Plugin bietet. Für Aurora-MySQL-Version 3 ist das für Datenbankbenutzer verwendete Standard-Authentifizierungs-Plugin das `mysql_native_password`-Plugin.

Diese Vorabprüfung warnt davor, dass dieses Plugin entfernt und die Standardeinstellung in einer künftigen Hauptversion geändert wird. Erwägen Sie, vor dieser Änderung die Kompatibilität Ihrer Anwendungsclients und Benutzer zu überprüfen.

Weitere Informationen finden Sie unter Kompatibilitätsprobleme mit caching_sha2_password und deren Lösungen in der MySQL-Dokumentation.

Beispielausgabe:


```
{
  "id": "defaultAuthenticationPlugin",
  "title": "New default authentication plugin considerations",
  "description": "Warning: The new default authentication plugin
 'caching_sha2_password' offers more secure password hashing than previously
 used 'mysql_native_password' (and consequent improved client connection
 authentication). However, it also has compatibility implications that
 may affect existing MySQL installations. If your MySQL installation
 must serve pre-8.0 clients and you encounter compatibility issues after
 upgrading, the simplest way to address those issues is to reconfigure
 the server to revert to the previous default authentication plugin
 (mysql_native_password). For example, use these lines in the server option file:
\n\n[mysqld]\ndefault_authentication_plugin=mysql_native_password\n\nHowever, the
 setting should be viewed as temporary, not as a long term or permanent solution,
 because it causes new accounts created with the setting in effect to forego the
 improved authentication security.\nIf you are using replication please take time to
 understand how the authentication plugin changes may impact you.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-caching-sha2-password-compatibility-issues\nhttps://
dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-series.html#upgrade-caching-
sha2-password-replication"
},
```


maxdbFlagCheck

Stufe der Vorabprüfung: Warnung

Verwendung eines veralteten **`MAXDB`**-**`sql_mode`**-Flags

In MySQL 8.0 wurden eine Reihe von veralteten sql_mode-Systemvariablenoptionen entfernt, darunter `MAXDB`. Diese Vorabprüfung untersucht alle derzeit verbundenen Sitzungen sowie Routinen und Trigger, um sicherzustellen, dass keine davon `sql_mode` auf eine Kombination gesetzt haben, die `MAXDB` enthält.

Beispielausgabe:

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "description": "Warning: The following DB objects have the obsolete MAXDB
 option persisted for sql_mode, which will be cleared during the upgrade. It can
 potentially change the datatype DATETIME into TIMESTAMP if it is used inside
 object's definition, and this in turn can change the behavior in case of dates
 earlier than 1970 or later than 2037. If this is a concern, please redefine these
 objects so that they do not rely on the MAXDB flag before running the upgrade.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/mysql-
nutshell.html#mysql-nutshell-removals",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.maxdb_stored_routine",
        "description": "PROCEDURE uses obsolete MAXDB sql_mode",
        "dbObjectType": "Routine"
      }
  ]
}
```

Die Vorabprüfung meldet, dass die Routine `test.maxdb_stored_routine` eine nicht unterstützte `sql_mode`-Option enthält.

Nachdem Sie sich bei der Datenbank angemeldet haben, können Sie in der Routinedefinition sehen, dass `sql_mode` `MAXDB` enthält.

```
 > SHOW CREATE PROCEDURE test.maxdb_stored_routine\G

*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode:
 PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,MAXDB,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPT
```

```
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Zum Beheben des Problems erstellen Sie die Prozedur neu, nachdem Sie den richtigen `sql_mode` auf dem Client festgelegt haben.

> **Note**
>
> Gemäß der MySQL-Dokumentation speichert MySQL die `sql_mode`-Einstellung, die gültig ist, wenn eine Routine erstellt oder geändert wird. Die Routine wird stets mit dieser Einstellung ausgeführt, unabhängig von der `sql_mode`-Einstellung beim Start der Routine.
>
> Lesen Sie vor der Änderung von `sql_mode` den Abschnitt Server-SQL-Modi in der MySQL-Dokumentation. Prüfen Sie sorgfältig, welche möglichen Auswirkungen dies auf Ihre Anwendung haben könnte.

Erstellen Sie die Prozedur ohne den nicht unterstützten `sql_mode` neu.

```
mysql > set sql_mode='PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql > DROP PROCEDURE test.maxdb_stored_routine\G
Query OK, 0 rows affected (0.00 sec)

mysql >
mysql > DELIMITER $$
mysql >
mysql > CREATE PROCEDURE test.maxdb_stored_routine()
    -> SQL SECURITY DEFINER
    -> BEGIN
    ->     SELECT * FROM test;
    -> END$$
Query OK, 0 rows affected (0.00 sec)
```

```
mysql >
mysql > DELIMITER ;
mysql > show create procedure test.maxdb_stored_routine\G
*************************** 1. row ***************************
           Procedure: maxdb_stored_routine
            sql_mode: PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE
    Create Procedure: CREATE DEFINER="msandbox"@"localhost" PROCEDURE
 "maxdb_stored_routine"()
BEGIN
    SELECT * FROM test;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

Die Vorabprüfung ist erfolgreich.

```
{
  "id": "maxdbFlagCheck",
  "title": "Usage of obsolete MAXDB sql_mode flag",
  "status": "OK",
  "detectedProblems": []
}
```

mysqlDollarSignNameCheck

Stufe der Vorabprüfung: Warnung

Prüfen auf die veraltete Verwendung einzelner Dollarzeichen in Objektnamen

Ab MySQL 8.0.32 ist die Verwendung des Dollarzeichens ($) als erstes Zeichen eines Bezeichners ohne Anführungszeichen veraltet. Wenn Sie über Schemas, Tabellen, Ansichten, Spalten oder Routinen verfügen, die ein $ als erstes Zeichen enthalten, gibt diese Vorabprüfung eine Warnung zurück. Diese Warnung verhindert zwar nicht, dass das Upgrade fortgesetzt wird, wir empfehlen Ihnen jedoch, bald Maßnahmen zu ergreifen, um dieses Problem zu beheben. Ab MySQL 8.4 geben solche Bezeichner eher einen Syntaxfehler als eine Warnung zurück.

Beispielausgabe:

```
{
```

```
  "id": "mysqlDollarSignNameCheck",
  "title": "Check for deprecated usage of single dollar signs in object names",
  "status": "OK",
  "description": "Warning: The following objects have names with deprecated usage
 of dollar sign ($) at the begining of the identifier. To correct this warning,
 ensure, that names starting with dollar sign, also end with it, similary to quotes
 ($example$). ",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.$deprecated_syntx",
        "description": " name starts with $ sign."
      }
  ]
},
```

Die Vorabprüfung gibt eine Warnung aus, da die Tabelle `$deprecated_syntx` im `test`-Schema ein $ als erstes Zeichen enthält.

reservedKeywordsCheck

Stufe der Vorabprüfung: Warnung

Verwendung von Datenbankobjekten, deren Namen mit neuen reservierten Schlüsselwörtern kollidieren

Diese Prüfung ähnelt routineSyntaxCheck, da sie die Verwendung von Datenbankobjekten prüft, deren Namen mit neuen reservierten Schlüsselwörtern kollidieren. Obwohl sie Upgrades nicht blockieren, müssen Warnungen sorgfältig bewertet werden.

Beispielausgabe:

Bei Verwendung des vorherigen Beispiels mit der Tabelle namens `except` gibt die Vorabprüfung eine Warnung zurück:

```
{
  "id": "reservedKeywordsCheck",
  "title": "Usage of db objects with names conflicting with new reserved keywords",
  "status": "OK",
  "description": "Warning: The following objects have names that conflict with
 new reserved keywords. Ensure queries sent by your applications use `quotes` when
 referring to them or they will result in errors.",
  "documentationLink": "https://dev.mysql.com/doc/refman/en/keywords.html",
```

```
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.except",
        "description": "Table name",
        "dbObjectType": "Table"
      }
  ]
}
```

Diese Warnung informiert Sie darüber, dass möglicherweise einige Anwendungsabfragen überprüft werden müssen. Wenn Ihre Anwendungsabfragen Zeichenfolgenliterale nicht korrekt maskieren, kann es nach dem Upgrade auf MySQL 8.0 zu Fehlern kommen. Überprüfen Sie Ihre Anwendungen zur Bestätigung und testen Sie sie anhand eines Klons oder Snapshots Ihres DB-Clusters von Aurora MySQL, der auf Version 3 ausgeführt wird.

Beispiel für eine nicht maskierte Anwendungsabfrage, die nach dem Upgrade fehlschlägt:

```
SELECT * FROM escape;
```

Beispiel für eine korrekt maskierte Anwendungsabfrage, die nach dem Upgrade erfolgreich ist:

```
SELECT * FROM 'escape';
```

utf8mb3Check

Stufe der Vorabprüfung: Warnung

Verwendung des **`utf8mb3`**-Zeichensatzes

In MySQL 8.0 ist der `utf8mb3`-Zeichensatz veraltet und wird in einer künftigen MySQL-Hauptversion entfernt. Diese Vorabprüfung ist implementiert, um eine Warnung auszulösen, wenn Datenbankobjekte mit diesem Zeichensatz erkannt werden. Dies verhindert zwar nicht, dass ein Upgrade fortgesetzt wird, wir empfehlen Ihnen jedoch dringend, darüber nachzudenken, Tabellen zum `utf8mb4`-Zeichensatz zu migrieren, der ab MySQL 8.0 die Standardeinstellung ist. Weitere Informationen zu utf8mb3 und utf8mb4 finden Sie unter Konvertierung zwischen 3-Byte- und 4-Byte-Unicode-Zeichensätzen in der MySQL-Dokumentation.

Beispielausgabe:

```
{
```

```
  "id": "utf8mb3",
  "title": "Usage of utf8mb3 charset",
  "status": "OK",
  "description": "Warning: The following objects use the deprecated utf8mb3
 character set. It is recommended to convert them to use utf8mb4 instead, for
 improved Unicode support. The utf8mb3 character is subject to removal in the
 future.",
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-
utf8mb3.html",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.t1.col1",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
      },
      {
        "level": "Warning",
        "dbObject": "test.t1.col2",
        "description": "column's default character set: utf8",
        "dbObjectType": "Column"
      }
  ]
}
```

Zum Beheben dieses Problems erstellen Sie die Objekte und Tabellen neu, auf die verwiesen wird. Weitere Informationen und Voraussetzungen dafür finden Sie unter Konvertierung zwischen 3-Byte- und 4-Byte-Unicode-Zeichensätzen in der MySQL-Dokumentation.

zeroDatesCheck

Stufe der Vorabprüfung: Warnung

Nullwerte für Date, Datetime und Timestamp

MySQL erzwingt jetzt strengere Regeln für die Verwendung von Nullwerten in Date-, Datetime- und Timestamp-Spalten. Wir empfehlen, die Modi `NO_ZERO_IN_DATE` und `NO_ZERO_DATE SQL` zusammen mit dem Modus `strict` zu verwenden, da sie in einer künftigen MySQL-Version mit dem Modus `strict` zusammengeführt werden.

Wenn die `sql_mode`-Einstellung für eine Ihrer Datenbankverbindungen zum Zeitpunkt der Ausführung der Vorabprüfung diese Modi nicht enthält, wird bei der Vorabprüfung eine Warnung ausgegeben. Benutzer können möglicherweise weiterhin Werte für Date, Datetime und

Timestamp einfügen, die Nullwerte enthalten. Wir empfehlen jedoch dringend, alle Nullwerte durch gültige Werte zu ersetzen, da sich ihr Verhalten in Zukunft ändern könnte und sie möglicherweise nicht richtig funktionieren. Da es sich um eine Warnung handelt, werden Upgrades dadurch nicht blockiert. Wir empfehlen Ihnen jedoch, mit der Planung von Maßnahmen zu beginnen.

Beispielausgabe:

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no
 longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included
 in SQL_MODE by default. These modes should be used with strict mode as they will be
 merged with strict mode in a future release. If you do not include these modes in
 your SQL_MODE setting, you are able to insert date/datetime/timestamp values that
 contain zeros. It is strongly advised to replace zero values with valid ones, as
 they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "global.sql_mode",
        "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
 which allows insertion of zero dates"
      },
      {
        "level": "Warning",
        "dbObject": "session.sql_mode",
        "description": " of 10 session(s) does not contain either NO_ZERO_DATE or
 NO_ZERO_IN_DATE which allows insertion of zero dates"
      }
  ]
}
```

Aurora-MySQL-Vorabprüfungen, die Warnungen melden

Die folgenden Vorabprüfungen sind spezifisch für Aurora MySQL:

- auroraUpgradeCheckForRollbackSegmentHistoryLength
- auroraUpgradeCheckForUncommittedRowModifications

auroraUpgradeCheckForRollbackSegmentHistoryLength

Stufe der Vorabprüfung: Warnung

Prüft, ob die Länge der Liste des Rollback-Segmentverlaufs für den Cluster hoch ist

Wie in auroraUpgradeCheckForIncompleteXATransactions erwähnt, ist es wichtig, dass die DB-Instance von Aurora-MySQL-Version 2 während des Upgrade-Prozesses für die Hauptversion ordnungsgemäß heruntergefahren wird. Dadurch wird sichergestellt, dass alle Transaktionen festgeschrieben oder rückgängig gemacht werden und dass InnoDB alle Undo-Protokolleinträge bereinigt hat.

Wenn Ihr DB-Cluster eine hohe Länge der Liste des Rollback-Segmentverlaufs (History List Length, HLL) aufweist, kann das die Zeit verlängern, die InnoDB benötigt, um die Undo-Protokolleinträge zu bereinigen. Dies führt zu längeren Ausfallzeiten während des Upgrade-Prozesses für die Hauptversion. Wenn die Vorabprüfung feststellt, dass der HLL-Wert in Ihrem DB-Cluster hoch ist, wird eine Warnung ausgegeben. Dies verhindert zwar nicht, dass Ihr Upgrade fortgesetzt wird, wir empfehlen Ihnen jedoch, die HLL in Ihrem DB-Cluster genau zu überwachen. Ein möglichst niedriger Wert reduziert die Ausfallzeit, die während des Hauptversions-Upgrade erforderlich ist. Weitere Informationen zur Überwachung der HLL finden Sie unter Die Länge der InnoDB-Verlaufsliste wurde deutlich erhöht.

Beispielausgabe:

```
{
  "id": "auroraUpgradeCheckForRollbackSegmentHistoryLength",
  "title": "Checks if the rollback segment history length for the cluster is high",
  "status": "OK",
  "description": "Rollback Segment History length is greater than 1M. Upgrade may
 take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_metrics",
        "description": "The InnoDB undo history list length('trx_rseg_history_len')
 is 82989114. Upgrade may take longer due to purging of undo information for old row
 versions."
      }
  ]
}
```

Die Vorabprüfung gibt eine Warnung zurück, da festgestellt wurde, dass die InnoDB-Undo-HLL im Datenbank-Cluster hoch war (82989114). Obwohl das Upgrade fortgesetzt wird, kann sich die erforderliche Ausfallzeit während des Upgrade-Vorgangs je nach Umfang der zu bereinigenden Undo-Daten verlängern.

Wir empfehlen Ihnen, offene Transaktionen in Ihrem DB-Cluster zu untersuchen, bevor Sie das Upgrade in der Produktion ausführen, um sicherzustellen, dass Ihre HLL eine überschaubare Größe hat.

auroraUpgradeCheckForUncommittedRowModifications

Stufe der Vorabprüfung: Warnung

Prüft, ob es viele nicht festgeschriebene Zeilenänderungen gibt

Wie in auroraUpgradeCheckForIncompleteXATransactions erwähnt, ist es wichtig, dass die DB-Instance von Aurora-MySQL-Version 2 während des Upgrade-Prozesses für die Hauptversion ordnungsgemäß heruntergefahren wird. Dadurch wird sichergestellt, dass alle Transaktionen festgeschrieben oder rückgängig gemacht werden und dass InnoDB alle Undo-Protokolleinträge bereinigt hat.

Wenn Ihr DB-Cluster über Transaktionen verfügt, die eine große Anzahl von Zeilen geändert haben, kann dies die Zeit verlängern, die InnoDB benötigt, um ein Rollback dieser Transaktion als Teil des ordnungsgemäßen Herunterfahrens abzuschließen. Wenn bei der Vorabprüfung lang laufende Transaktionen mit einer großen Anzahl von geänderten Zeilen in Ihrem DB-Cluster festgestellt werden, wird eine Warnung ausgegeben. Dies verhindert zwar nicht, dass Ihr Upgrade fortgesetzt wird, wir empfehlen Ihnen jedoch, die Größe der aktiven Transaktionen in Ihrem DB-Cluster genau zu überwachen. Ein möglichst niedriger Wert reduziert die Ausfallzeit, die während des Hauptversions-Upgrade erforderlich ist.

Beispielausgabe:


```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "description": "Database contains uncommitted row changes greater than 10M.
 Upgrade may take longer time.",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "information_schema.innodb_trx",
```

```
        "description": "The database contains 11000000 uncommitted row change(s) in
 1 transaction(s). Upgrade may take longer due to transaction rollback."
      }
  ]
},
```

Die Vorabprüfung meldet, dass der DB-Cluster eine Transaktion mit 11 000 000 nicht festgeschriebenen Zeilenänderungen enthält, die während des ordnungsgemäßen Herunterfahrens rückgängig gemacht werden müssen. Das Upgrade wird fortgesetzt, aber um die Ausfallzeiten während des Upgrade-Prozesses zu reduzieren, empfehlen wir Ihnen, dies zu überwachen und zu untersuchen, bevor Sie das Upgrade in Ihren Produktions-Clustern ausführen.

Wenn Sie aktive Transaktionen in Ihrer Schreiber-DB-Instance anzeigen möchten, können Sie die Tabelle information_schema.innodb_trx verwenden. Die folgende Abfrage der Schreiber-DB-Instance zeigt aktuelle Transaktionen, die Laufzeit, den Status und geänderte Zeilen für den DB-Cluster.

```
# Example of uncommitted transaction
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state | rows_modified |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
| 2024-08-12 18:32:52 |                           1592 |
 20041 | 52866130 | RUNNING   |      11000000 |
+---------------------+-----------------------------
+---------------------------------+----------+-----------+---------------+
1 row in set (0.01 sec)

# Example of transaction rolling back
mysql> SELECT trx_started,
       TIME_TO_SEC(TIMEDIFF(now(), trx_started)) AS seconds_trx_has_been_running,
       trx_mysql_thread_id AS show_processlist_connection_id,
```

```
       trx_id,
       trx_state,
       trx_rows_modified AS rows_modified
FROM information_schema.innodb_trx;
+--------------------+------------------------------
+--------------------------------+----------+-------------+---------------+
| trx_started         | seconds_trx_has_been_running |
 show_processlist_connection_id | trx_id   | trx_state    | rows_modified |
+--------------------+------------------------------
+--------------------------------+----------+-------------+---------------+
| 2024-08-12 18:32:52 |                          1719 |
 20041 | 52866130 | ROLLING BACK |      10680479 |
+--------------------+------------------------------
+--------------------------------+----------+-------------+---------------+
1 row in set (0.01 sec)
```

Nachdem die Transaktion festgeschrieben oder rückgängig gemacht wurde, gibt die Vorabprüfung keine Warnung mehr zurück. Konsultieren Sie die MySQL-Dokumentation und Ihr Anwendungsteam, bevor Sie große Transaktionen rückgängig machen, da das Rollback je nach Transaktionsgröße einige Zeit in Anspruch nehmen kann.

```
{
  "id": "auroraUpgradeCheckForUncommittedRowModifications",
  "title": "Checks if there are many uncommitted modifications to rows",
  "status": "OK",
  "detectedProblems": []
},
```

Weitere Informationen zur Optimierung der InnoDB-Transaktionsverwaltung und zu den möglichen Auswirkungen der Ausführung und des Rollbacks großer Transaktionen in MySQL-Datenbank-Instances finden Sie unter Optimieren der InnoDB-Transaktionsverwaltung in der MySQL-Dokumentation.

Hinweise

Die folgende Vorabprüfung generiert einen Hinweis, wenn die Vorabprüfung fehlschlägt, aber das Upgrade kann fortgesetzt werden.

sqlModeFlagCheck

Stufe der Vorabprüfung: Hinweis

Verwendung von veralteten **`sql_mode`**-Flags

Zusätzlich zu `MAXDB` wurden eine Reihe weiterer `sql_mode`-Optionen entfernt: `DB2`, `MSSQL`, `MYSQL323`, `MYSQL40`, `ORACLE`, `POSTGRESQL`, `NO_FIELD_OPTIONS`, `NO_KEY_OPTIONS` und `NO_TABLE_OPTIONS`. Ab MySQL 8.0 kann keiner dieser Werte der Systemvariablen `sql_mode` zugewiesen werden. Wenn diese Vorabprüfung offene Sitzungen findet, die diese `sql_mode`-Einstellungen verwenden, stellen Sie sicher, dass Ihre DB-Instance- und DB-Cluster-Parametergruppen sowie Client-Anwendungen und -Konfigurationen aktualisiert werden, um sie zu deaktivieren. Weitere Informationen finden Sie in der MySQL-Dokumentation.

Beispielausgabe:

```
{
  "id": "sqlModeFlagCheck",
  "title": "Usage of obsolete sql_mode flags",
  "status": "OK",
  "detectedProblems": []
}
```

Informationen zum Beheben dieser Fehler bei der Vorabprüfung finden Sie unter maxdbFlagCheck.

Fehler, Warnungen oder Hinweise

Die folgende Vorabprüfung kann je nach Ausgabe der Vorabprüfung einen Fehler, eine Warnung oder einen Hinweis zurückgeben.

checkTableOutput

Stufe der Vorabprüfung: Fehler, Warnung oder Hinweis

Probleme, die vom Befehl **`check table x for upgrade`** gemeldet wurden

Bevor das Upgrade auf Aurora-MySQL-Version 3 gestartet wird, wird `check table for upgrade` für jede Tabelle in den Benutzerschemas in Ihrem DB-Cluster ausgeführt. Diese Vorabprüfung entspricht nicht checkTableMysqlSchema.

Der Befehl `check table for upgrade` untersucht Tabellen auf mögliche Probleme, die bei einem Upgrade auf eine neuere Version von MySQL auftreten könnten. Das Ausführen dieses Befehls vor dem Upgrade kann helfen, etwaige Inkompatibilitäten im Voraus zu identifizieren und zu beheben, sodass der eigentliche Upgrade-Prozess reibungsloser abläuft.

Mit diesem Befehl werden verschiedene Prüfungen für jede Tabelle durchgeführt, z. B. die folgenden:

- Überprüfung, ob die Tabellenstruktur und die Metadaten mit der Ziel-MySQL-Version kompatibel sind
- Prüfung auf veraltete oder entfernte Funktionen, die von der Tabelle verwendet werden
- Sicherstellung, dass die Tabelle ordnungsgemäß und ohne Datenverlust aktualisiert werden kann

Im Gegensatz zu anderen Vorabprüfungen kann sie je nach `check table`-Ausgabe einen Fehler, eine Warnung oder einen Hinweis zurückgeben. Wenn diese Vorabprüfung Tabellen zurückgibt, überprüfen Sie diese zusammen mit dem Rückgabecode und der Meldung sorgfältig, bevor Sie das Upgrade starten. Weitere Informationen finden Sie unter CHECK-TABLE-Anweisung in der MySQL-Dokumentation.

Hier finden Sie ein Beispiel für einen Fehler und eine Warnung.

Beispiel für einen Fehler:

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Error",
        "dbObject": "test.parent",
        "description": "Table 'test.parent' doesn't exist"
      }
  ]
},
```

Die Vorabprüfung meldet einen Fehler, der besagt, dass die Tabelle `test.parent` nicht existiert.

Die `mysql-error.log`-Datei für die Schreiber-DB-Instance zeigt, dass ein Fremdschlüsselfehler vorliegt.

```
2024-08-13T15:32:10.676893Z 62 [Warning] InnoDB: Load table `test`.`parent` failed,
 the table has missing foreign key indexes. Turn off 'foreign_key_checks' and try
 again.
2024-08-13T15:32:10.676905Z 62 [Warning] InnoDB: Cannot open table test/parent from
 the internal data dictionary of InnoDB though the .frm file for the table exists.
```

```
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-troubleshooting.html
 for how to resolve the issue.
```

Melden Sie sich bei der Schreiber-DB-Instance an und führen Sie `show engine innodb status\G` aus, um weitere Informationen zum Fremdschlüsselfehler zu erhalten.

```
mysql> show engine innodb status\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2024-08-13 15:33:33 0x14ef7b8a1700 INNODB MONITOR OUTPUT
=====================================
.
.
.
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2024-08-13 15:32:10 0x14ef6dbbb700 Error in foreign key constraint of table test/
child:
there is no index in referenced table which would contain
the columns as the first columns, or the data types in the
referenced table do not match the ones in table. Constraint:
,
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
The index in the foreign key in table is p_name_idx
Please refer to http://dev.mysql.com/doc/refman/5.7/en/innodb-foreign-key-
constraints.html for correct foreign key definition.
.
.
```

Die `LATEST FOREIGN KEY ERROR`-Meldung gibt an, dass die `fk_pname`-Fremdschlüsseleinschränkung in der Tabelle `test.child`, die auf die Tabelle `test.parent` verweist, entweder einen fehlenden Index oder einen Datentypkonflikt aufweist. Die MySQL-Dokumentation zu Fremdschlüsseleinschränkungen besagt, dass Spalten, auf die in einem Fremdschlüssel verwiesen wird, einen zugehörigen Index haben müssen und dass die übergeordneten/untergeordneten Spalten denselben Datentyp verwenden müssen.

Wenn Sie überprüfen möchten, ob dies mit einem fehlenden Index oder einer Nichtübereinstimmung des Datentyps zusammenhängt, melden Sie sich bei der Datenbank an

und überprüfen Sie die Tabellendefinitionen, indem Sie vorübergehend die Sitzungsvariable foreign_key_checks deaktivieren. Danach können wir sehen, dass die fragliche untergeordnete Einschränkung (`fk_pname`) `p_name varchar(20) CHARACTER SET latin1 DEFAULT NULL` verwendet, um auf die übergeordnete Tabelle `name varchar(20) NOT NULL` zu verweisen. Die übergeordnete Tabelle verwendet `DEFAULT CHARSET=utf8`, aber die Spalte `p_name` der untergeordneten Tabelle verwendet `latin1`, sodass der Fehler wegen fehlender Datentypübereinstimmung ausgelöst wird.

```
mysql> show create table parent\G
ERROR 1146 (42S02): Table 'test.parent' doesn't exist

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> set foreign_key_checks=0;
Query OK, 0 rows affected (0.00 sec)

mysql> show create table parent\G
*************************** 1. row ***************************
       Table: parent
Create Table: CREATE TABLE `parent` (
  `name` varchar(20) NOT NULL,
  PRIMARY KEY (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> show create table child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) NOT NULL,
  `p_name` varchar(20) CHARACTER SET latin1 DEFAULT NULL,
  PRIMARY KEY (`id`),
```

```
  KEY `p_name_idx` (`p_name`),
  CONSTRAINT `fk_pname` FOREIGN KEY (`p_name`) REFERENCES `parent` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

Zum Beheben dieses Problems können wir entweder die untergeordnete Tabelle so ändern, dass sie denselben Zeichensatz wie die übergeordnete Tabelle verwendet, oder die übergeordnete Tabelle so ändern, dass sie denselben Zeichensatz wie die untergeordnete Tabelle verwendet. Da die untergeordnete Tabelle hier explizit `latin1` in der `p_name`-Spaltendefinition verwendet, führen wir `ALTER TABLE` aus, um den Zeichensatz in `utf8` zu ändern.

```
mysql> alter table child modify p_name varchar(20) character set utf8 DEFAULT NULL;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> flush tables;
Query OK, 0 rows affected (0.01 sec)
```

Danach ist die Vorabprüfung erfolgreich und das Upgrade kann fortgesetzt werden.

Beispiel für eine Warnung:

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": [
      {
        "level": "Warning",
        "dbObject": "test.orders",
        "description": "Trigger test.orders.delete_audit_trigg does not have CREATED
 attribute."
      }
  ]
}
```

Die Vorabprüfung meldet eine Warnung für den Trigger `delete_audit_trigg` in der Tabelle `test.orders`, da er kein `CREATED`-Attribut aufweist. Laut Überprüfen der Versionskompatibilität in der MySQL-Dokumentation ist diese Meldung informativ und wird für Trigger angezeigt, die vor MySQL 5.7.2 erstellt wurden.

Da es sich um eine Warnung handelt, verhindert sie nicht, dass das Upgrade fortgesetzt wird. Wenn Sie das Problem jedoch beheben möchten, können Sie den fraglichen Trigger neu erstellen und die Vorabprüfung ist ohne Warnungen erfolgreich.

```
{
  "id": "checkTableOutput",
  "title": "Issues reported by 'check table x for upgrade' command",
  "status": "OK",
  "detectedProblems": []
},
```

Erläuterung der Durchführung eines direkten Upgrades

Sehen Sie sich die Hintergrundinformationen unter Funktionsweise des Aurora MySQL direkten Upgrade der Hauptversion an.

Führen Sie alle Planungen und Tests vor dem Upgrade durch, wie unter Planen eines Hauptversionsupgrades für einen Aurora MySQL-Cluster beschrieben.

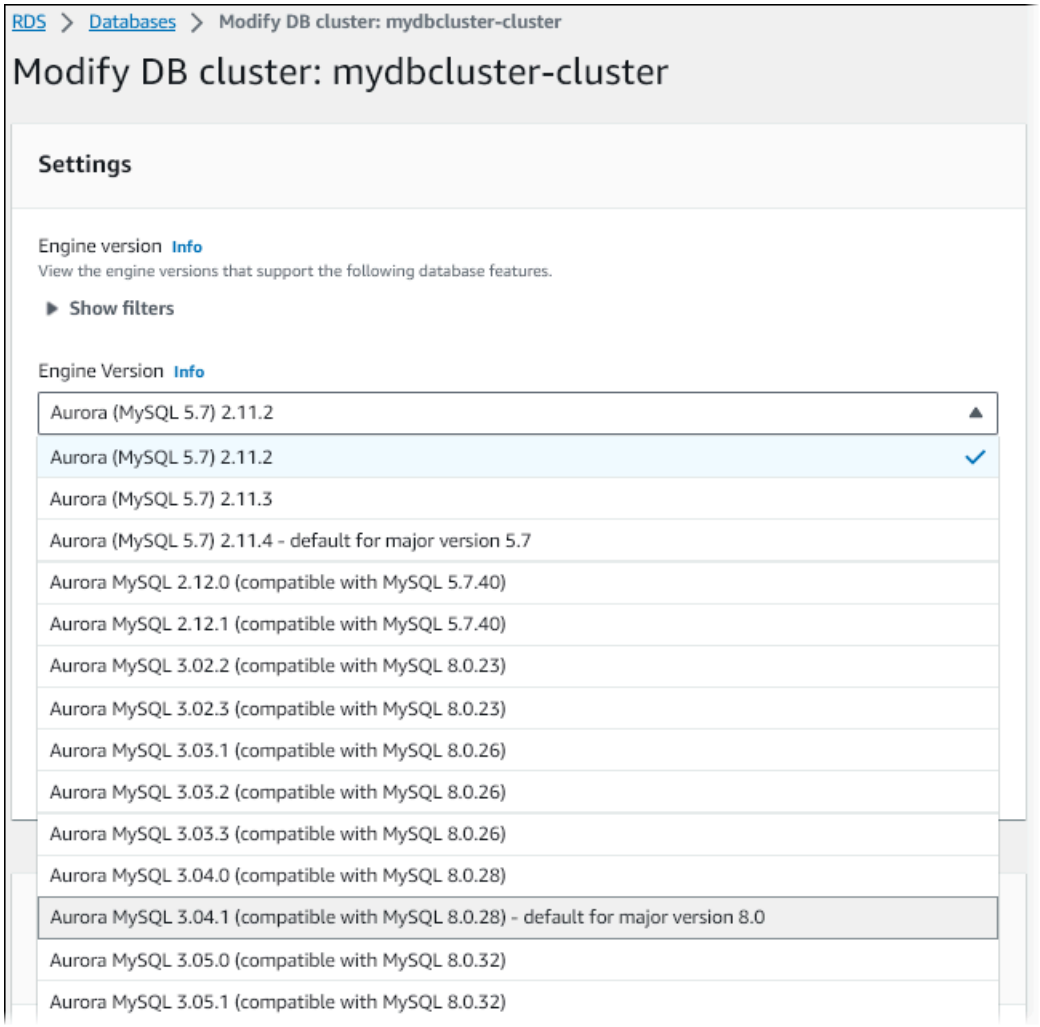
Konsole

Im folgenden Beispiel wird der `mydbcluster-cluster`-DB-Cluster auf Aurora-MySQL-Version 3.04.1 aktualisiert.

Aktualisieren der Hauptversion eines Aurora MySQL-DB-Clusters

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wenn Sie eine benutzerdefinierte Parametergruppe für den ursprünglichen DB-Cluster verwendet haben, erstellen Sie eine entsprechende Parametergruppe, die mit der neuen Hauptversion kompatibel ist. Nehmen Sie alle erforderlichen Anpassungen an den Konfigurationsparametern in dieser neuen Parametergruppe vor. Weitere Informationen finden Sie unter Wie sich direkte Upgrades auf die Parametergruppen für einen Cluster auswirken.
3. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
4. Wählen Sie den DB-Cluster aus, den Sie ändern möchten.
5. Wählen Sie Ändern aus.
6. Wählen Sie unter Version eine neue Aurora-MySQL-Hauptversion aus.

   Wir empfehlen im Allgemeinen, die neueste Nebenversion der Hauptversion zu verwenden. Hier wählen wir die aktuelle Standardversion.



7. Klicken Sie auf Weiter.
8. Geben Sie auf der nächsten Seite an, wann das Upgrade durchgeführt werden soll. Wählen Sie During the next scheduled maintenance window (Während des nächsten geplanten Wartungsfensters) oder Sofort aus.
9. (Optional) Überprüfen Sie während des Upgrades regelmäßig die Seite Ereignisse in der RDS-Konsole. Auf diese Weise können Sie den Fortschritt des Upgrades überwachen und etwaige Probleme erkennen. Wenn das Upgrade auf Probleme stößt, lesen Sie Problembehandlung für Aurora MySQL direkte Upgrades für zu ergreifende Schritte.
10. Wenn Sie zu Beginn dieses Vorgangs eine neue Parametergruppe erstellt haben, ordnen Sie die benutzerdefinierte Parametergruppe Ihrem aktualisierten Cluster zu. Weitere Informationen finden Sie unter Wie sich direkte Upgrades auf die Parametergruppen für einen Cluster auswirken.

> **Note**
>
> Wenn Sie diesen Schritt ausführen, müssen Sie den Cluster erneut neu starten, um die neue Parametergruppe anzuwenden.

11. (Optional) Löschen Sie nach dem Upgrade den manuellen Snapshot, den Aurora zu Beginn des Upgrades erstellt hat.

AWS CLI

Um die Hauptversion eines Aurora-MySQL-DB-Clusters zu aktualisieren, verwenden Sie den AWS CLI-Befehl modify-db-cluster mit den folgenden erforderlichen Parametern:

- `--db-cluster-identifier`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` oder `--no-apply-immediately`

Wenn Ihr Cluster benutzerdefinierte Parametergruppen verwendet, schließen Sie auch eine oder beide der folgenden Optionen ein:

- `--db-cluster-parameter-group-name`, wenn der Cluster eine benutzerdefinierte Cluster-Parametergruppe verwendet
- `--db-instance-parameter-group-name`, falls Instances im Cluster eine benutzerdefinierte DB-Parametergruppe verwenden

Im folgenden Beispiel wird der `sample-cluster`-DB-Cluster auf Aurora-MySQL-Version 3.04.1 aktualisiert. Das Upgrade erfolgt sofort, anstatt auf das nächste Wartungsfenster zu warten.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --engine-version 8.0.mysql_aurora.3.04.1 \
          --allow-major-version-upgrade \
```

```
          --apply-immediately
```

Für Windows:

```
aws rds modify-db-cluster ^
          --db-cluster-identifier sample-cluster ^
          --engine-version 8.0.mysql_aurora.3.04.1 ^
          --allow-major-version-upgrade ^
          --apply-immediately
```

Sie können andere CLI-Befehle mit `modify-db-cluster` kombinieren, um einen automatisierten End-to-End-Prozess zum Durchführen und Überprüfen von Upgrades zu erstellen. Weitere Informationen und Beispiele finden Sie unter Aurora MySQL direktes Upgrade.

> **Note**
>
> Wenn Ihr Cluster Teil einer Aurora globalen Datenbank ist, unterscheidet sich das Verfahren des direkten Upgrades geringfügig. Sie rufen die Befehlsoperation modify-global-cluster statt `modify-db-cluster` auf. Weitere Informationen finden Sie unter In-Situ-Hauptversions-Upgrades für globale Datenbanken.

RDS-API

Verwenden Sie zum Aktualisieren der Hauptversion eines DB-Clusters von Aurora MySQL die RDS-API-Operation ModifyDBCluster mit den folgenden erforderlichen Parametern:

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately` (festgelegt auf `true` oder `false`)

> **Note**
>
> Wenn Ihr Cluster Teil einer Aurora globalen Datenbank ist, unterscheidet sich das Verfahren des direkten Upgrades geringfügig. Sie rufen den Vorgang ModifyGlobalCluster statt

`ModifyDBCluster` auf. Weitere Informationen finden Sie unter In-Situ-Hauptversions-Upgrades für globale Datenbanken.

Wie sich direkte Upgrades auf die Parametergruppen für einen Cluster auswirken

Aurora-Parametergruppen haben verschiedene Sätze von Konfigurationseinstellungen für Cluster, die mit MySQL 5.7 oder 8.0 kompatibel sind. Wenn Sie ein direktes Upgrade durchführen, müssen der aktualisierte Cluster und alle seine Instances die entsprechenden Cluster- und Instance-Parametergruppen verwenden:

Ihr Cluster und Ihre Instances verwenden möglicherweise die standardmäßigen 5.7-kompatiblen Parametergruppen. In diesem Fall beginnen der aktualisierte Cluster und die Instance mit den standardmäßigen 8.0-kompatiblen Parametergruppen. Wenn Ihr Cluster und Ihre Instances benutzerdefinierte Parametergruppen verwenden, müssen Sie entsprechende oder 8.0-kompatible Parametergruppen erstellen. Diese müssen während des Upgrade-Vorgangs angegeben werden.

Note

Für die meisten Parametereinstellungen können Sie die benutzerdefinierte Parametergruppe an zwei Stellen auswählen. Und zwar können Sie sie auswählen, wenn Sie den Cluster erstellen oder wenn Sie die Parametergruppe später dem Cluster zuordnen.
Wenn Sie jedoch eine nicht standardmäßige Einstellung für den Parameter `lower_case_table_names` verwenden, müssen Sie die benutzerdefinierte Parametergruppe mit dieser Einstellung im Voraus einrichten. Geben Sie dann die Parametergruppe an, wenn Sie die Snapshot-Wiederherstellung zum Erstellen des Clusters durchführen. Änderungen des `lower_case_table_names`-Parameters haben keine Auswirkung, nachdem der Cluster erstellt wurde.
Wir empfehlen Ihnen, die gleiche Einstellung für `lower_case_table_names` zu verwenden, wenn Sie ein Upgrade von Aurora MySQL Version 2 auf Version 3 durchführen.
Bei einer globalen Aurora-Datenbank, die auf Aurora MySQL basiert, können Sie ein direktes Upgrade von Aurora-MySQL-Version 2 auf Version 3 nur durchführen, wenn Sie den `lower_case_table_names`-Parameter auf den Standardwert setzen und Ihre globale Datenbank neu starten. Weitere Informationen zu den möglichen Verfahren finden Sie unter Hauptversions-Upgrades.

Änderungen an Cluster-Eigenschaften zwischen Aurora-MySQL-Versionen

Wenn Sie von Aurora-MySQL-Version 2 zu Version 3 aktualisieren, stellen Sie sicher, dass Sie alle Anwendungen oder Skripts ändern, die Sie zum Einrichten oder Verwalten von Clustern und DB-Instances von Aurora MySQL verwenden.

Ändern Sie außerdem Ihren Code, der Parametergruppen manipuliert, um der Tatsache Rechnung zu tragen, dass die Standardnamen der Parametergruppen für 5.7- und 8.0-kompatible Cluster unterschiedlich sind. Die entsprechenden Parametergruppennamen für Aurora-MySQL-Version 2 und 3-Cluster sind `default.aurora-mysql5.7` und `default.aurora-mysql8.0`.

Beispielsweise haben Sie möglicherweise Code wie den folgenden, der vor einem Upgrade für Ihren Cluster gilt.

```
# Check the default parameter values for MySQL 5.7-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --
region us-east-1
```

Ändern Sie nach dem Upgrade der Hauptversion des Clusters diesen Code wie folgt.

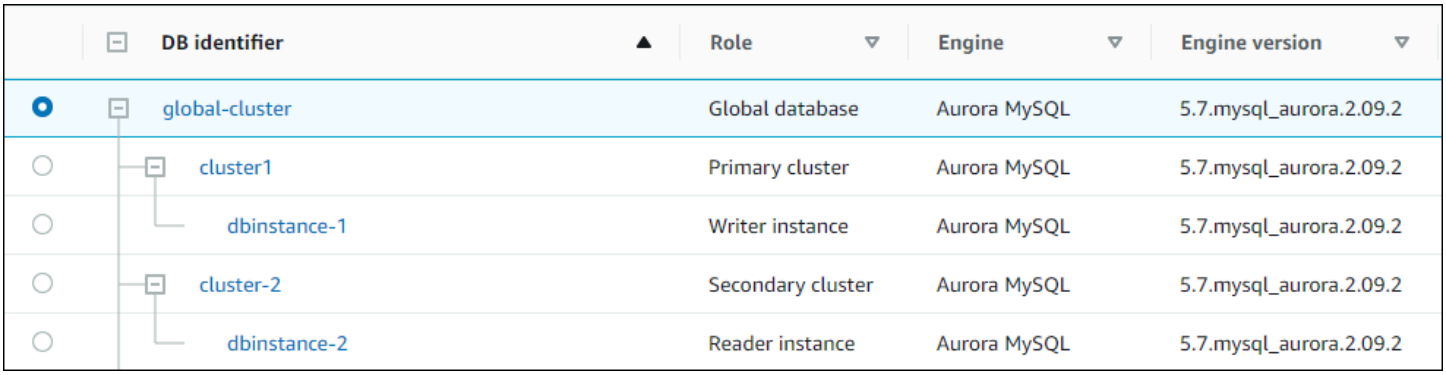
```
# Check the default parameter values for MySQL 8.0-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql8.0 --
region us-east-1
```

In-Situ-Hauptversions-Upgrades für globale Datenbanken

Bei einer globalen Aurora-Datenbank aktualisieren Sie den globalen Datenbank-Cluster. Aurora aktualisiert automatisch alle Cluster gleichzeitig und stellt sicher, dass sie alle dieselbe Engine-Version ausführen. Diese Anforderung liegt darin begründet, dass Änderungen an Systemtabellen, Datendateiformaten usw. automatisch auf alle sekundären Cluster repliziert werden.

Folgen Sie den Anweisungen in Funktionsweise des Aurora MySQL direkten Upgrade der Hauptversion. Wenn Sie angeben, was aktualisiert werden soll, stellen Sie sicher, dass Sie den globalen Datenbank-Cluster anstelle eines der darin enthaltenen Cluster auswählen.

Wenn Sie die AWS-Managementkonsole verwenden, wählen Sie das Element mit der Rolle Global database (Globale Datenbank) aus.



DB identifier	Role	Engine	Engine version
global-cluster	Global database	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster1	Primary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-1	Writer instance	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster-2	Secondary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-2	Reader instance	Aurora MySQL	5.7.mysql_aurora.2.09.2

Wenn Sie die AWS CLI oder die RDS-API verwenden, starten Sie den Upgrade-Prozess, indem Sie den Befehl modify-global-cluster oder die Operation ModifyGlobalCluster aufrufen. Verwenden Sie diese anstelle von `modify-db-cluster` oder `ModifyDBCluster`.

> **Note**
>
> Sie können keine benutzerdefinierte Parametergruppe für den globalen Datenbank-Cluster angeben, während Sie ein größeres Versions-Upgrade dieser globalen Aurora-Datenbank durchführen. Erstellen Sie Ihre benutzerdefinierten Parametergruppen in jeder Region des globalen Clusters. Wenden Sie sie nach dem Upgrade manuell auf die regionalen Cluster an.

Um die Hauptversion eines globalen Datenbank-Clusters von Aurora MySQL mithilfe der AWS CLI zu aktualisieren, verwenden Sie den Befehl modify-gobal-cluster mit den folgenden erforderlichen Parametern:

- `--global-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`

Im folgenden Beispiel wird der globale Datenbank-Cluster auf Aurora-MySQL-Version 3.04.2 aktualisiert.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-global-cluster \
```

```
          --global-cluster-identifier global_cluster_identifier \
          --engine aurora-mysql \
          --engine-version 8.0.mysql_aurora.3.04.2 \
          --allow-major-version-upgrade
```

Für Windows:

```
aws rds modify-global-cluster ^
          --global-cluster-identifier global_cluster_identifier ^
          --engine aurora-mysql ^
          --engine-version 8.0.mysql_aurora.3.04.2 ^
          --allow-major-version-upgrade
```

Direkte Upgrades für DB-Cluster mit regionsübergreifenden Lesereplikaten

Sie können einen Aurora-DB-Cluster, der über ein regionsübergreifendes Lesereplikat verfügt, mithilfe des direkten Upgrades aktualisieren, es gibt jedoch einige Überlegungen:

- Sie müssen zuerst den Lesereplikat-DB-Cluster aktualisieren. Wenn Sie zuerst versuchen, den primären Cluster zu aktualisieren, erhalten Sie eine Fehlermeldung wie die folgende:

  Unable to upgrade DB cluster test-xr-primary-cluster because the associated Aurora cross-Region replica test-xr-replica-cluster isn't patched yet. Upgrade the Aurora cross-Region replica and try again.

  Das bedeutet, dass der primäre DB-Cluster keine höhere DB-Engine-Version als der Replikat-Cluster haben kann.
- Bevor Sie den primären DB-Cluster aktualisieren, halten Sie den Schreib-Workload an und deaktivieren Sie alle neuen Verbindungsanfragen zur Schreiber-DB-Instance des primären Clusters.
- Wenn Sie den primären Cluster aktualisieren, wählen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe aus, bei der der `binlog_format`-Parameter auf einen Wert gesetzt ist, der die Replikation von Binärprotokollen unterstützt, z. B. `MIXED`.

  Weitere Informationen zur Verwendung der binären Protokollierung mit Aurora MySQL finden Sie unter Replizieren zwischen Aurora und MySQL oder zwischen Aurora und einem anderen Aurora-DB-Cluster (binäre Protokollreplikation). Weitere Informationen zum Ändern der Aurora MySQL-Konfigurationsparameter finden Sie unter Aurora MySQL Konfigurationsparameter und Parametergruppen für Amazon Aurora.

- Warten Sie nicht lange mit dem Upgrade des primären DB-Clusters, nachdem Sie den Replikat-Cluster aktualisiert haben. Wir empfehlen, nicht länger als bis zum nächsten Wartungszeitraum zu warten.
- Starten Sie nach dem Upgrade des primären DB-Clusters dessen Schreiber-DB-Instance neu. Die benutzerdefinierte DB-Cluster-Parametergruppe, die die Binlog-Replikation aktiviert, wird erst wirksam, wenn die Schreiber-DB-Instance neu gestartet wird.
- Nehmen Sie den Schreib-Workload erst wieder auf oder aktivieren Sie Verbindungen zur Schreiber-DB-Instance, wenn Sie bestätigt haben, dass die regionsübergreifende Replikation neu gestartet wurde und dass die Replikatverzögerung in der sekundären AWS-Region 0 ist.

Aurora MySQL direktes Upgrade

Die folgenden Linux-Beispiele zeigen, wie Sie die allgemeinen Schritte des Verfahrens für ein direktes Upgrade mit dem durchführen können AWS CLI.

In diesem ersten Beispiel wird ein Aurora-DB-Cluster erstellt, auf dem eine 2.x-Version von Aurora MySQL ausgeführt wird. Der Cluster enthält eine Writer-DB-Instance und eine Reader-DB-Instance Der Befehl `wait db-instance-available` pausiert, bis die Writer-DB-Instance verfügbar ist. Das ist der Punkt, an dem der Cluster für ein Upgrade bereit ist.

```
aws rds create-db-cluster --db-cluster-identifier mynewdbcluster --engine aurora-mysql
 \
  --db-cluster-version 5.7.mysql_aurora.2.11.2
...
aws rds create-db-instance --db-instance-identifier mynewdbcluster-instance1 \
  --db-cluster-identifier mynewdbcluster --db-instance-class db.t4g.medium --engine
 aurora-mysql
...
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

Die Aurora-MySQL-3.x-Versionen, auf die Sie den Cluster aktualisieren können, hängen von der 2.x-Version ab, auf der der Cluster derzeit läuft, sowie von der AWS-Region-Region, in der sich der Cluster befindet. Der erste Befehl mit `--output text` zeigt nur die verfügbare Zielversion an. Der zweite Befehl zeigt die vollständige JSON-Ausgabe der Antwort an. Diese Antwort enthält Details wie den `aurora-mysql`-Wert, den Sie für den `engine`-Parameter verwenden. Sie können auch sehen, dass ein Upgrade auf 3.04.0 ein Hauptversion-Upgrade darstellt.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
```

```
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.11.2 \
  --query '*[].[ValidUpgradeTarget]'
...
{
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.04.0",
    "Description": "Aurora MySQL 3.04.0 (compatible with MySQL 8.0.28)",
    "AutoUpgrade": false,
    "IsMajorVersionUpgrade": true,
    "SupportedEngineModes": [
        "provisioned"
    ],
    "SupportsParallelQuery": true,
    "SupportsGlobalDatabases": true,
    "SupportsBabelfish": false,
    "SupportsIntegrations": false
},
...
```

Dieses Beispiel zeigt, dass Aurora das Upgrade nicht durchführt, wenn Sie eine Zielversionsnummer eingeben, die kein gültiges Upgrade-Ziel für den Cluster ist. Aurora führt auch kein Hauptversions-Upgrade durch, es sei denn, Sie geben den Parameter `--allow-major-version-upgrade` an. Auf diese Weise können Sie nicht versehentlich ein Upgrade durchführen, das umfangreiche Tests und Änderungen an Ihrem Anwendungscode erfordert.

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 5.7.mysql_aurora.2.09.2 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: Cannot find upgrade target from 5.7.mysql_aurora.2.11.2 with requested
 version 5.7.mysql_aurora.2.09.2.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: The AllowMajorVersionUpgrade flag must be present when upgrading to a new
 major version.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --apply-immediately --allow-major-version-
upgrade
```

```
{
  "DBClusterIdentifier": "mynewdbcluster",
  "Status": "available",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.11.2"
}
```

Es dauert einige Augenblicke, bis sich der Status des Clusters und der zugehörigen DB-Instances auf änder `upgrading`. Die Versionsnummern für die Cluster- und DB-Instances ändern sich erst, wenn das Upgrade abgeschlossen ist. Auch hier können Sie den `wait db-instance-available`-Befehl verwenden, damit die Writer-DB-Instance wartet, bis das Upgrade abgeschlossen ist, bevor Sie fortfahren.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[Status,EngineVersion]' --output text
upgrading 5.7.mysql_aurora.2.11.2

aws rds describe-db-instances --db-instance-identifier mynewdbcluster-instance1 \
  --query '*[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceStatus} | [0]'
{
    "DBInstanceIdentifier": "mynewdbcluster-instance1",
    "DBInstanceStatus": "upgrading"
}

aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

Zu diesem Zeitpunkt entspricht die Versionsnummer für den Cluster der Nummer, die für das Upgrade angegeben wurde.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[EngineVersion]' --output text

8.0.mysql_aurora.3.04.0
```

Das vorhergehende Beispiel führte ein sofortiges Upgrade durch Angabe des `--apply-immediately`-Parameters durch. Damit das Upgrade zu einem geeigneten Zeitpunkt durchgeführt wird, zu dem nicht zu erwarten ist, dass der Cluster ausgelastet ist, können Sie den Parameter `--no-apply-immediately` festlegen. Dadurch wird das Upgrade während des nächsten Wartungsfensters für den Cluster gestartet. Das Wartungsfenster definiert den Zeitraum, in

dem Wartungsvorgänge beginnen können. Ein lang andauernder Vorgang kann während des Wartungsfensters möglicherweise nicht beendet werden. Daher müssen Sie kein größeres Wartungsfenster definieren, selbst wenn Sie davon ausgehen, dass das Upgrade sehr lange dauern kann.

Im folgenden Beispiel wird ein Cluster aktualisiert, auf dem ursprünglich Aurora-MySQL-Version 2.11.2 ausgeführt wurde. In der Ausgabe `describe-db-engine-versions` stellen die Werte `False` und `True` die Eigenschaft `IsMajorVersionUpgrade` dar. Ab Version 2.11.2 können Sie auf andere 2.*-Versionen aktualisieren. Diese Upgrades gelten nicht als Hauptversions-Upgrades und erfordern daher kein direktes Upgrade. Ein direktes Upgrade ist nur für die 3.*-Versionen verfügbar, die in der Liste aufgeführt sind.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.11.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget]|[0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]'
 --output text

5.7.mysql_aurora.2.11.3 False
5.7.mysql_aurora.2.11.4 False
5.7.mysql_aurora.2.11.5 False
5.7.mysql_aurora.2.11.6 False
5.7.mysql_aurora.2.12.0 False
5.7.mysql_aurora.2.12.1 False
5.7.mysql_aurora.2.12.2 False
5.7.mysql_aurora.2.12.3 False
8.0.mysql_aurora.3.04.0 True
8.0.mysql_aurora.3.04.1 True
8.0.mysql_aurora.3.04.2 True
8.0.mysql_aurora.3.04.3 True
8.0.mysql_aurora.3.05.2 True
8.0.mysql_aurora.3.06.0 True
8.0.mysql_aurora.3.06.1 True
8.0.mysql_aurora.3.07.1 True

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.04.0 --no-apply-immediately --allow-major-
version-upgrade
...
```

Wenn ein Cluster ohne ein festgelegtes Wartungsfenster erstellt wird, wählt Aurora einen zufälligen Wochentag aus. In diesem Fall wird der Befehl `modify-db-cluster` an einem Montag gesendet. Daher ändern wir das Wartungsfenster auf Dienstagmorgen. Alle Zeiten sind in der UTC-Zeitzone dargestellt. Das Fenster `tue:10:00-tue:10:30` entspricht 2:00 - 2:30 Uhr Pacific Time. Die Änderung des Wartungsfensters wird sofort wirksam.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster --preferred-
maintenance-window tue:10:00-tue:10:30"
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
]
```

Das folgende Beispiel zeigt, wie ein Bericht über die durch das Upgrade generierten Ereignisse abgerufen wird. Das Argument `--duration` gibt die Anzahl der Minuten an, die für den Abruf der Ereignisinformationen erforderlich sind. Dieses Argument ist erforderlich, da `describe-events` standardmäßig nur Ereignisse der letzten Stunde ausgibt.

```
aws rds describe-events --source-type db-cluster --source-identifier mynewdbcluster --
duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
            "EventCategories": [
                "creation"
            ],
            "Date": "2022-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
```

```
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:08.450000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-
mynewdbcluster-5-7-mysql-aurora-2-10-2-to-8-0-mysql-aurora-3-02-0-2022-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:01:45.428000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
```

```
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:02:25.141000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:23.036000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Upgrading database objects.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:48.208000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster major version has been upgraded",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:10:28.999000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        }
    ]
}
```

Finden der Gründe für Fehler bei Hauptversion-Upgrades von Aurora MySQL

Im Tutorial wurde das Upgrade von Aurora-MySQL-Version 2 auf Version 3 erfolgreich abgeschlossen. Wenn das Upgrade jedoch fehlgeschlagen wäre, würden Sie wissen wollen, warum.

Sie können damit beginnen, den AWS CLI-Befehl `describe-events` zu verwenden, um sich die DB-Cluster-Ereignisse anzusehen. Dieses Beispiel zeigt die Ereignisse für `mydbcluster` der letzten 10 Stunden.

```
aws rds describe-events \
    --source-type db-cluster \
    --source-identifier mydbcluster \
    --duration 600
```

In diesem Fall ist bei der Upgrade-Vorabprüfung ein Fehler aufgetreten.

```
{
    "Events": [
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:22.846000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        },
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster is in a state that cannot be upgraded: Upgrade
 prechecks failed. For more details, see the
             upgrade-prechecks.log file. For more information on troubleshooting the
 cause of the upgrade failure, see
             https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
AuroraMySQL.Upgrading.Troubleshooting.html",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:24.373000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        }
```

```
    ]
}
```

Um die genaue Ursache des Problems zu diagnostizieren, untersuchen Sie die Datenbankprotokolle für die Schreiber-DB-Instance. Wenn ein Upgrade auf Aurora-MySQL-Version 3 fehlschlägt, enthält die Schreiber-Instance eine Protokolldatei namens `upgrade-prechecks.log`. Dieses Beispiel zeigt, wie Sie das Vorhandensein dieses Protokolls erkennen und es dann zur Untersuchung in eine lokale Datei herunterladen.

```
aws rds describe-db-log-files --db-instance-identifier mydbcluster-instance \
    --query '*[].[LogFileName]' --output text

error/mysql-error-running.log
error/mysql-error-running.log.2024-04-11.20
error/mysql-error-running.log.2024-04-11.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

aws rds download-db-log-file-portion --db-instance-identifier mydbcluster-instance \
    --log-file-name upgrade-prechecks.log \
    --starting-token 0 \
    --output text >upgrade_prechecks.log
```

Die `upgrade-prechecks.log`-Datei befindet sich im JSON-Format. Wir laden sie mit der Option `--output text` herunter, um JSON-Output nicht in einem anderen JSON-Wrapper zu kodieren. Für Upgrades auf Aurora-MySQL-Version 3 enthält dieses Protokoll immer bestimmte Informations- und Warnmeldungen. Es enthält nur Fehlermeldungen, wenn das Upgrade fehlschlägt. Wenn das Upgrade erfolgreich ist, wird die Protokolldatei überhaupt nicht erstellt.

Um alle Fehler zusammenzufassen und die zugehörigen Objekt- und Beschreibungsfelder anzuzeigen, können Sie den Befehl `grep -A 2 '"level": "Error"'` für den Inhalt der `upgrade-prechecks.log`-Datei ausführen. Dadurch werden jede Fehlerzeile und die beiden Zeilen danach angezeigt. Diese enthalten den Namen des entsprechenden Datenbankobjekts und Anleitungen zur Behebung des Problems.

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'

"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
```

```
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains dangling
 FULLTEXT index. Kindly recreate the table before upgrade."
```

In diesem Beispiel können Sie den folgenden SQL-Befehl für die problematische Tabelle ausführen, um das Problem zu beheben, oder Sie können die Tabelle ohne den hängenden Index neu erstellen.

```
OPTIMIZE TABLE problematic_upgrade.dangling_fulltext_index;
```

Versuchen Sie anschließend das Upgrade erneut.

Problembehandlung für Aurora MySQL direkte Upgrades

Verwenden Sie die folgenden Tipps, um Probleme mit direkten Aurora-MySQL-Upgrades zu beheben. Diese Tipps gelten nicht für Aurora Serverless-DB-Cluster.

Grund für das abgebrochene oder langsame direkte Upgrade	Auswirkung	Lösung, die den Abschluss eines direkten Upgrades im Wartungsfenster ermöglicht
Das zugehörige regionsübergreifende Aurora-Replikat wurde noch nicht gepatcht.	Aurora bricht das Upgrade ab.	Aktualisieren Sie das regionsübergreifende Aurora-Replikat und versuchen Sie es erneut.
Der Cluster hat XA-Transaktionen im vorbereiteten Zustand	Aurora bricht das Upgrade ab.	Bestätigen Sie alle vorbereiteten XA-Transaktionen oder setzen Sie sie zurück.
Der Cluster verarbeitet alle DDL-Anweisungen (Data Definition Language)	Aurora bricht das Upgrade ab.	Erwägen Sie, zu warten und das Upgrade durchzuführen, nachdem alle DDL-Anweisungen abgeschlossen sind.
Der Cluster hat für viele Zeilen nicht festgeschriebene Änderungen	Das Upgrade könnte eine lange Zeit dauern.	Der Upgrade-Prozess setzt die nicht festgeschriebenen Änderungen zurück. Der Indikator für diese Bedingung ist der Wert von `TRX_ROWS_MODIFIED` in der `INFORMATION_SCHEMA.INNODB_TRX` -Tabelle.

Grund für das abgebrochene oder langsame direkte Upgrade	Auswirkung	Lösung, die den Abschluss eines direkten Upgrades im Wartungsfenster ermöglicht
		Erwägen Sie, das Upgrade erst durchzuführen, nachdem alle großen Transaktionen festgeschrieben oder zurückgesetzt wurden.
Der Cluster hat eine hohe Anzahl von Undo-Datensätzen	Das Upgrade könnte eine lange Zeit dauern.	Selbst wenn sich die nicht festgeschriebenen Transaktionen nicht auf eine große Anzahl von Zeilen auswirken, können sie eine große Datenmenge beinhalten. Beispielsweise könnten Sie große BLOBs einfügen. Aurora erkennt oder generiert ein Ereignis für diese Art von Transaktionsaktivität nicht automatisch. Der Indikator für diese Bedingung ist die Länge der Verlaufsliste (History List Length, HLL). Der Upgrade-Prozess setzt die nicht festgeschriebenen Änderungen zurück. Sie können die HLL in der Ausgabe des SQL-Befehls `SHOW ENGINE INNODB STATUS` oder direkt mit der folgenden SQL-Abfrage überprüfen: `SELECT count FROM information_schema.innodb_metrics WHERE name = 'trx_rseg_history_len';` Sie können auch die Metrik `RollbackSegmentHistoryListLength` in Amazon CloudWatch überwachen. Erwägen Sie, das Upgrade erst durchzuführen, wenn die HLL kleiner ist.

Grund für das abgebrochene oder langsame direkte Upgrade	Auswirkung	Lösung, die den Abschluss eines direkten Upgrades im Wartungsfenster ermöglicht
Der Cluster ist dabei, eine große binäre Protokoll-Transaktion festzuschreiben	Das Upgrade könnte eine lange Zeit dauern.	Der Upgrade-Prozess wartet, bis die binären Protokolländerungen angewendet werden. In diesem Zeitraum können mehr Transaktionen oder DDL-Anweisungen gestartet werden, was den Upgrade-Prozess weiter verlangsamt. Planen Sie den Upgrade-Prozess, wenn der Cluster nicht mit der Generierung von Änderungen der binären Protokollreplikation beschäftigt Aurora erkennt oder generiert ein Ereignis für diese Bedingung nicht automatisch.

Grund für das abgebrochene oder langsame direkte Upgrade	Auswirkung	Lösung, die den Abschluss eines direkten Upgrades im Wartungsfenster ermöglicht
Schemainkonsistenzen, die auf das Entfernen oder die Beschädigung von Dateien zurückzuführen sind	Aurora bricht das Upgrade ab.	Ändern Sie die Standard-Speicher-Engine für temporäre Tabellen von MyISAM auf InnoDB. Führen Sie die folgenden Schritte aus: 1. Ändern Sie den DB-Parameter `default_tmp_storage_engine` in `InnoDB`. 2. Starten Sie den DB-Cluster neu. 3. Vergewissern Sie sich nach dem Neustart, dass der DB-Parameter `default_tmp_storage_engine` auf `InnoDB` festgelegt ist. Verwenden Sie den folgenden Befehl: `show global variables like 'default_tmp_storage_engine';` 4. Stellen Sie sicher, dass Sie keine temporären Tabellen erstellen, die die MyISAM-Speicher-Engine verwenden. Wir empfehlen, dass Sie jeden Datenbank-Workload unterbrechen und keine neuen Datenbankverbindungen herstellen, da Sie bald ein Upgrade durchführen werden. 5. Versuchen Sie erneut, das direkte Upgrade durchzuführen.

Grund für das abgebrochene oder langsame direkte Upgrade	Auswirkung	Lösung, die den Abschluss eines direkten Upgrades im Wartungsfenster ermöglicht

Der Master-Benutzer wurde gelöscht.	Aurora bricht das Upgrade ab.	**Important**
		Löschen Sie den Master-Benutzer nicht.
		
		Sollten Sie jedoch aus irgendeinem Grund den Master-Benutzer löschen, stellen Sie ihn mithilfe der folgenden SQL-Befehle wieder her:
		
		`CREATE USER 'master_username '@'% IDENTIFIED BY ' master_user_password ' REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK;`
		
		`GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT INTO S3, INVOKE LAMBDA, INVOKE SAGEMAKER, INVOKE COMPREHEND ON *.* TO 'master_username '@'% WITH GRANT OPTION;`

Weitere Informationen zur Behebung von Problemen, die dazu führen, dass Upgrade-Vorabprüfungen fehlschlagen, finden Sie in den folgenden Blogs:

- Checkliste für das Upgrade von Amazon-Aurora-MySQL-Version 2 (mit MySQL-5.7-Kompatibilität) auf Version 3 (mit MySQL-8.0-Kompatibilität), Teil 1

- Checkliste für das Upgrade von Amazon-Aurora-MySQL-Version 2 (mit MySQL-5.7-Kompatibilität) auf Version 3 (mit MySQL-8.0-Kompatibilität), Teil 2

Sie können die folgenden Schritte verwenden, um eigene Überprüfungen für einige der Bedingungen in der vorherigen Tabelle durchzuführen. Auf diese Weise können Sie das Upgrade für einen Zeitpunkt planen, an dem Sie wissen, dass sich die Datenbank in einem Zustand befindet, in dem das Upgrade erfolgreich und schnell abgeschlossen werden kann.

- Sie können nach offenen XA-Transaktionen suchen, indem Sie die `XA RECOVER`-Anweisung ausführen. Sie können dann die XA-Transaktionen festschreiben oder zurücksetzen, bevor Sie mit dem Upgrade beginnen.
- Sie können nach DDL-Anweisungen suchen, indem Sie eine `SHOW PROCESSLIST` Anweisung ausführen und in der Ausgabe nach `CREATE`, `DROP`, `ALTER`, `RENAME` und `TRUNCATE` Anweisungen suchen. Warten Sie bis alle DDL-Anweisungen fertig sind, bevor Sie mit dem Upgrade beginnen.
- Sie können die Gesamtzahl der nicht festgeschriebenen Zeilen überprüfen, indem Sie die `INFORMATION_SCHEMA.INNODB_TRX`-Tabelle abfragen. Die Tabelle enthält eine Zeile für jede Transaktion. Die `TRX_ROWS_MODIFIED` Spalte enthält die Anzahl der Zeilen, die von der Transaktion geändert oder eingefügt wurden.
- Sie können die Länge der InnoDB-Verlaufsliste überprüfen, indem Sie die `SHOW ENGINE INNODB STATUS SQL`-Anweisung ausführen und nach `History list length` in der Ausgabe suchen. Sie können den Wert auch direkt überprüfen, indem Sie die folgende Abfrage ausführen:

  ```
  SELECT count FROM information_schema.innodb_metrics WHERE name =
   'trx_rseg_history_len';
  ```

  Die Länge der Verlaufsliste entspricht der Menge der Undo-Daten, die von der Datenbank gespeichert werden, um die Multi-Versions-Concurrency Control (MVCC) zu implementieren.

Bereinigung nach dem Upgrade für Aurora MySQL Version 3

Nachdem Sie ein Upgrade von Aurora-MySQL-Clustern der 2 auf Aurora-MySQL-Version 3 abgeschlossen haben, können Sie diese anderen Bereinigungsaktionen ausführen:

- Erstellen Sie neue MySQL 8.0-kompatible Versionen beliebiger benutzerdefinierter Parametergruppen. Wenden Sie alle erforderlichen benutzerdefinierten Parameterwerte auf die neuen Parametergruppen an.

- Aktualisieren Sie alle CloudWatch-Alarme, Setup-Skripte usw., um die neuen Namen für alle Metriken zu verwenden, deren Namen von inklusiven Sprachänderungen betroffen waren. Eine Liste der -Metriken finden Sie unter Inklusive Sprachänderungen für Aurora MySQL Version 3.
- Aktualisieren Sie alleCloudFormationVorlagen, um die neuen Namen für alle Konfigurationsparameter zu verwenden, deren Namen von inklusiven Sprachänderungen betroffen waren. Eine vollständige Liste der Parameter finden Sie unter Inklusive Sprachänderungen für Aurora MySQL Version 3.

SPATIAL-Index

Überprüfen Sie nach dem Upgrade auf Aurora MySQL Version 3, ob Sie Objekte und Indizes im Zusammenhang mit räumlichen Indizes löschen oder neu erstellen müssen. Vor MySQL 8.0 konnte Aurora räumliche Abfragen mithilfe von Indizes optimieren, die keinen räumlichen Ressourcenbezeichner (SRID) enthielten. Aurora MySQL Version 3 verwendet nur räumliche Indizes, die SRIDs enthalten. Während eines Upgrades legt Aurora automatisch alle räumlichen Indizes ohne SRIDs ab und gibt Warnmeldungen im Datenbankprotokoll aus. Wenn Sie solche Warnmeldungen beobachten, erstellen Sie nach dem Upgrade neue räumliche Indizes mit SRIDs. Weitere Informationen zu Änderungen an räumlichen Funktionen und Datentypen in MySQL 8.0 finden Sie unter Änderungen in MySQL 8.0 im MySQL-Referenzhandbuch.

Datenbank-Engine-Updates und -Korrekturen für Amazon Aurora MySQL

In den Versionshinweisen für Amazon Aurora MySQL-Compatible Edition finden Sie folgende Informationen:

- Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 3
- Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 2
- Aktualisierungen der Datenbank-Engine für Amazon-Aurora-MySQL-Version 1 (veraltet)
- MySQL-Bugfixes durch Updates der Datenbank-Engine von Aurora MySQL
- In Amazon Aurora MySQL behobene Sicherheitsschwachstellen

Arbeiten mit Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL ist eine vollständig verwaltete, PostgreSQL-kompatible und ACID-kompatible relationale Datenbank-Engine, die die Geschwindigkeit, Zuverlässigkeit und Verwaltbarkeit von Amazon Aurora mit der Einfachheit und Kosteneffizienz von Open-Source-Datenbanken kombiniert. Aurora PostgreSQL ist ein Drop-In-Ersatz für PostgreSQL und macht es einfach und kostengünstig, Ihre neuen und bestehenden PostgreSQL-Implementierungen einzurichten, zu betreiben und zu skalieren, sodass Sie sich auf Ihr Unternehmen und Ihre Anwendungen konzentrieren können. Weitere Informationen über Aurora finden Sie unter Was ist Amazon Aurora?.

Zusätzlich zu den Vorteilen von Aurora bietet Aurora PostgreSQL einen bequemen Migrationspfad von Amazon RDS zu Aurora mit Migrationstools auf Knopfdruck, die Ihre vorhandenen Anwendungen von RDS für PostgreSQL in Aurora PostgreSQL konvertieren. Routinemäßige Datenbankaufgaben wie Bereitstellung, Patching, Backup, Wiederherstellung, Fehlererkennung und Reparatur sind mit Aurora PostgreSQL ebenfalls einfach zu verwalten.

Aurora PostgreSQL kann mit vielen Industriestandards arbeiten. Sie können beispielsweise Aurora PostgreSQL-Datenbanken verwenden, um HIPAA-konforme Anwendungen zu erstellen und Gesundheitsdaten einschließlich geschützter Gesundheitsdaten (Protected Health Information, PHI) entsprechend einem mit AWS abgeschlossenen Business Associate Agreement (BAA) zu speichern.

Aurora PostgreSQL ist mit FedRAMP HIGH konform. Einzelheiten zu den Compliance-Maßnahmen AWS und den Bemühungen zur Einhaltung der Vorschriften finden Sie unter AWS Services im Umfang der einzelnen Compliance-Programme.

Important

Wenn Sie auf ein Problem mit Ihrem Aurora PostgreSQL-DB-Cluster stoßen, benötigt Ihr AWS Support-Mitarbeiter möglicherweise weitere Informationen zum Zustand Ihrer Datenbanken. Ziel ist es, sicherzustellen, dass der AWS Support in kürzester Zeit so viele der erforderlichen Informationen wie möglich erhält.
Sie können PG Collector verwenden, um wertvolle Datenbankinformationen in einer konsolidierten HTML-Datei zu sammeln. Weitere Informationen zu PG Collector, zur Ausführung und zum Herunterladen des HTML-Berichts finden Sie unter PG Collector.
Nach erfolgreichem Abschluss und sofern nicht anders angegeben, gibt das Skript die Ausgabe in einem lesbaren HTML-Format zurück. Das Skript ist so konzipiert, dass alle

Daten oder Sicherheitsdetails, die Ihr Unternehmen gefährden könnten, aus dem HTML-Code ausgeschlossen werden. Es nimmt auch keine Änderungen an Ihrer Datenbank oder ihrer Umgebung vor. Wenn Sie jedoch Informationen im HTML-Code finden, die Sie nicht weitergeben möchten, können Sie diese entfernen, bevor Sie den HTML-Code hochladen. Wenn der HTML-Code akzeptabel ist, laden Sie ihn im Anlagen-Abschnitt der Falldetails Ihres Support-Falls hoch.

Themen

- Arbeiten mit Database Preview Environment
- Sicherheit in Amazon Aurora PostgreSQL
- Aktualisieren von Anwendungen für die Verbindung zu Aurora-PostgreSQL-DB-Clustern mit neuen SSL/TLS-Zertifikaten
- Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL
- Migrieren von Daten nach Amazon Aurora mit PostgreSQL-Kompatibilität
- Optimieren der Abfrageleistung in Aurora PostgreSQL
- Arbeiten mit Tabellen ohne Protokollierung in Aurora PostgreSQL
- Arbeiten mit der PostgreSQL-Selbstbereinigung in Amazon Aurora PostgreSQL
- Verwaltung von TOAST-OID-Konflikten in
- Verwenden von Babelfish for Aurora PostgreSQL
- Leistung und Skalierung für Amazon Aurora PostgreSQL
- Optimierung mit Warteereignissen für Aurora PostgreSQL
- Optimierung von Aurora PostgreSQL mit proaktiven Einblicken von Amazon DevOps Guru
- Bewährte Methoden mit Amazon Aurora PostgreSQL
- Replikation mit Amazon Aurora PostgreSQL
- Lokale Schreibweiterleitung in Aurora PostgreSQL
- Verwenden von Aurora PostgreSQL als Wissensdatenbank für Amazon Bedrock
- Integrieren von Amazon Aurora PostgreSQL in anderen AWS-Services
- Überwachen der Ausführungspläne von Abfragen und deren maximale Speichernutzung für Aurora PostgreSQL
- Verwalten von Abfrageausführungsplänen für Aurora PostgreSQL
- Arbeiten mit Erweiterungen und Fremddaten-Wrappern

- Arbeiten mit Trusted Language Extensions für PostgreSQL
- Amazon-Aurora-PostgreSQL-Referenz
- Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL

Arbeiten mit Database Preview Environment

Die PostgreSQL-Community veröffentlicht jährlich eine neue Hauptversion von PostgreSQL. Entsprechend stellt Amazon Aurora PostgreSQL-Hauptversionen als Vorschauversionen zur Verfügung. Auf diese Weise können Sie DB-Cluster mit der Vorschauversion erstellen und ihre Features in der Datenbank-Vorschauumgebung testen.

Aurora-PostgreSQL-DB-Cluster in der Datenbank-Vorschauumgebung funktionieren ähnlich wie andere Aurora-PostgreSQL-DB-Cluster. Sie können eine Vorschauversion jedoch nicht für die Produktion einsetzen.

Beachten Sie folgende wichtige Einschränkungen:

- Alle DB-Instances und DB-Cluster werden 60 Tage nach Erstellung zusammen mit allen Sicherungen und Snapshots gelöscht.
- Sie können eine DB-Instance nur in einer virtuellen privaten Cloud (VPC) erstellen, die auf dem Service Amazon VPC basiert.
- Sie können einen Snapshot einer DB-Instance nicht in eine Produktionsumgebung kopieren.

Die folgenden Optionen werden von der Vorschauversion unterstützt.

- Sie können nur DB-Instances mit den Instance-Typen r5, r6g, r6i, r7g, r7i, r8g, x2g, t3 und t4g erstellen. Weitere Informationen zu Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
- Sie können Single-AZ- und Multi-AZ-Bereitstellungen verwenden.
- Sie können die standardmäßigen PostgreSQL-Dump- und -Ladefunktionen verwenden, um Datenbanken aus der Database Vorschauumgebung zu exportieren oder in diese zu importieren.

Unterstützte Typen von DB-Instance-Klassen

Amazon Aurora PostgreSQL unterstützt die folgenden DB-Instance-Klassen in der Preview-Region:

Speicheroptimierte Klassen

- db.r5 – arbeitsspeicheroptimierte Instance-Klassen
- db.r6g – arbeitsspeicheroptimierte Instance-Klassen mit AWS-Graviton2-Prozessoren
- db.r6i – arbeitsspeicheroptimierte Instance-Klassen
- db.r7g – arbeitsspeicheroptimierte Instance-Klassen mit AWS-Graviton3-Prozessoren
- db.r7i – arbeitsspeicheroptimierte Instance-Klassen
- db.r8g – arbeitsspeicheroptimierte Instance-Klassen mit AWS-Graviton4-Prozessoren
- db.x2g - speicheroptimierte Instance-Klassen, die von AWS Graviton2-Prozessoren unterstützt werden

> **Note**
>
> Weitere Informationen zur Liste der Instance-Klassen finden Sie unter DB-Instance-Klassenarten.

Burst-fähige Klassen

- db.t3.medium
- db.t3.large
- db.t4g.medium
- db.t4g.large

Nicht in der Vorschauumgebung unterstützte Features

Die folgenden Funktionen sind in der Vorschauumgebung nicht verfügbar:

- Aurora Serverless v1 und v2
- Upgrades der Hauptversion (MVU)
- In der Vorschauregion werden keine neuen Nebenversionen veröffentlicht
- Eingehende Replikation von RDS für PostgreSQL nach Aurora PostgreSQL
- Blau/Grün-Bereitstellungen von Amazon RDS
- Regionsübergreifende Snapshot-Kopie

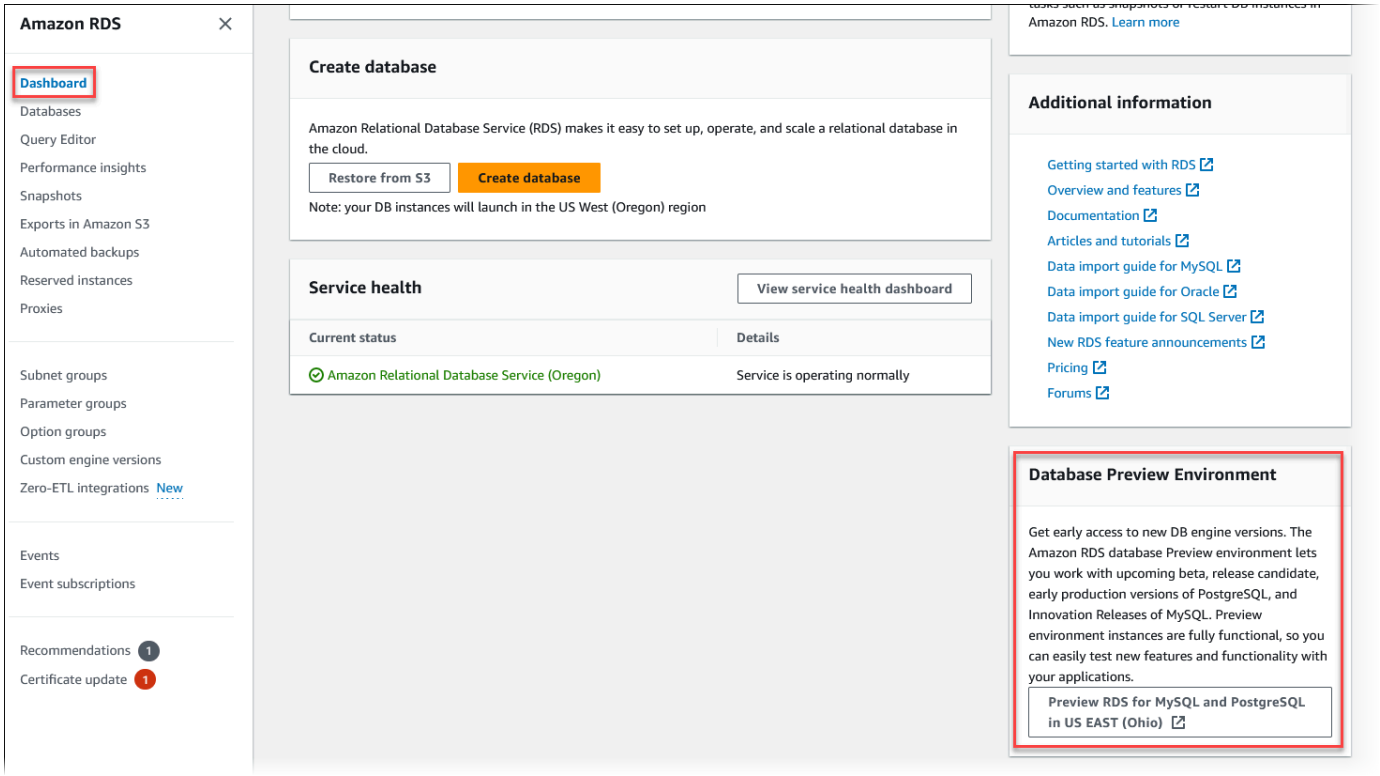
- Globale Aurora-Datenbank.
- Datenbankaktivitäts-Streams (DAS), RDS-Proxy und AWS DMS
- Auto Scaling von Lesereplikaten
- AWSBedrock
- Export von RDS
- Performance Insights
- Benutzerdefinierte Endpunkte
- Snapshot kopieren
- Null-ETL
- Babelfish
- Alle Erweiterungen

Erstellen eines neuen DB-Clusters in der Vorschauumgebung

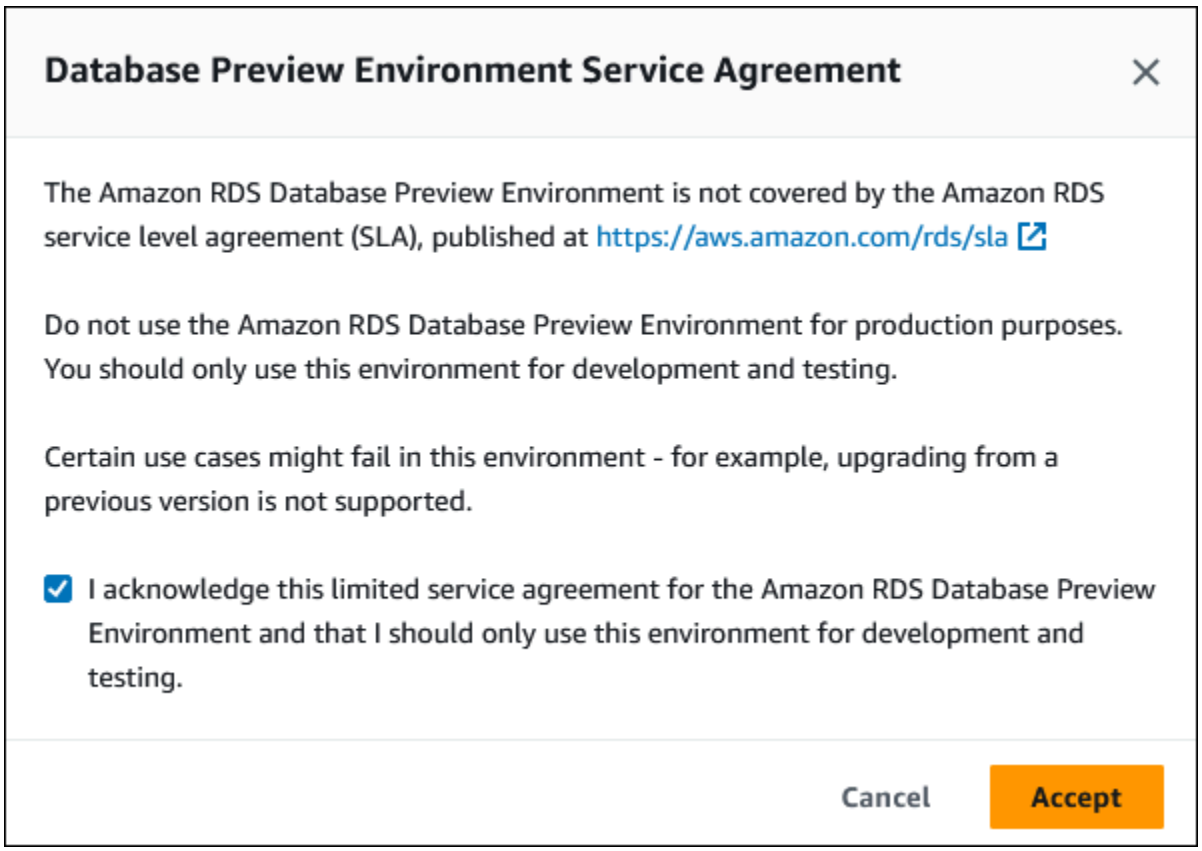
Gehen Sie zum Erstellen eines DB-Clusters in der Vorschauumgebung wie nachfolgend beschrieben vor.

So erstellen Sie einen DB-Cluster in der Vorschauumgebung

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Dashboard aus.
3. Suchen Sie auf Dashboard-Seite nach Datenbank-Vorschauumgebung, wie in der folgenden Abbildung gezeigt.



Sie können auch direkt zu Datenbank-Vorschauumgebung navigieren. Bevor Sie fortfahren können, müssen Sie die Einschränkungen bestätigen und akzeptieren.



4. Gehen Sie zum Erstellen des Aurora-PostgreSQL-DB-Clusters genauso vor wie bei der Erstellung eines Aurora-DB-Clusters. Weitere Informationen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Verwenden Sie den folgenden Endpunkt, um mit der Aurora-API oder der AWS CLI eine Instance in der Datenbank-Preview-Umgebung zu erstellen.

```
rds-preview.us-east-2.amazonaws.com
```

PostgreSQL Version 17 in der Datenbank-Vorschauumgebung

⚠ Dies ist eine Preview-Dokumentation für Aurora PostgreSQL Version 17. Änderungen sind vorbehalten.

PostgreSQL Version 17.0 ist jetzt in der Datenbank-Vorschauumgebung in Amazon RDS verfügbar. PostgreSQL Version 17 enthält mehrere Verbesserungen, die in der folgenden PostgreSQL-Dokumentation beschrieben werden:

- PostgreSQL 17 Released

Weitere Informationen zur Datenbank-Vorschauumgebung finden Sie unter Arbeiten mit Database Preview Environment. Wählen Sie https://console.aws.amazon.com/rds-preview/ aus, um von der Konsole aus auf die Vorschauumgebung zuzugreifen.

Sicherheit in Amazon Aurora PostgreSQL

Eine allgemeine Übersicht über die Aurora-Sicherheit finden Sie unter Sicherheit in Amazon Aurora. Sie können die Sicherheit für Amazon Aurora PostgreSQL auf mehreren Ebenen verwalten:

- Um zu kontrollieren, wer Amazon RDS-Managementaktionen auf Aurora PostgreSQL-DB-Clustern und DB-Instances ausführen kann, verwenden Sie AWS Identity and Access Management (IAM). IAM übernimmt die Authentifizierung der Benutzeridentität, bevor der Benutzer auf den Service zugreifen kann. Außerdem übernimmt IAM auch die Autorisierung, d. h., ob der Benutzer für die Vorgänge berechtigt ist, die er versucht auszuführen. Die IAM-Datenbankauthentifizierung ist eine zusätzliche Authentifizierungsmethode, die Sie beim Erstellen Ihres Aurora-PostgreSQL-DB-Clusters auswählen können. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora.

  Wenn Sie IAM mit Ihrem Aurora PostgreSQL-DB-Cluster verwenden, melden Sie sich zuerst AWS-Managementkonsole mit Ihren IAM-Anmeldeinformationen an, bevor Sie die Amazon RDS-Konsole unter öffnen. https://console.aws.amazon.com/rds/
- Stellen Sie sicher, dass Sie Aurora-DB-Cluster in einer Virtual Private Cloud (VPC) basierend auf dem Amazon-VPC-Service erstellen. Verwenden Sie eine VPC-Sicherheitsgruppe, um zu steuern, welche Geräte und EC2 Amazon-Instances Verbindungen zum Endpunkt und Port der DB-Instance für Aurora-DB-Cluster in einer VPC öffnen können. Sie können diese Endpunkt- und Portverbindungen mithilfe von Secure Sockets Layer (SSL) herstellen. Zusätzlich können Firewall-Regeln in Ihrem Unternehmen steuern, ob in Ihrem Unternehmen verwendete Geräte Verbindungen mit einer DB-Instance herstellen dürfen. Weitere Informationen dazu finden Sie unter VPCs. Amazon VPC und Amazon Aurora

Die unterstützte VPC-Tenancy hängt von der DB-Instance-Klasse ab, die von Ihren Aurora-PostgreSQL-DB-Clustern verwendet wird. Mit `default`-VPC-Tenancy, läuft der DB-Cluster auf gemeinsam genutzter Hardware. Mit `dedicated`-VPC-Tenancy läuft der DB-Cluster auf einer dedizierten Hardware-Instance. Die DB-Instance-Klassen mit Spitzenlastleistung unterstützen nur die Standard-VPC-Tenancy. Die DB-Instance-Klassen mit Spitzenlastleistung umfassen die DB-Instance-Klassen db.t3 und db.t4g. Alle anderen Aurora-PostgreSQL-DB-Instance-Klassen unterstützen sowohl die Standard- als auch die dedizierte VPC-Tenancy.

Weitere Informationen zu Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora. Weitere Informationen über die `default`- und `dedicated`-VPC-Tenancy finden Sie unter Dedicated Instances im Amazon Elastic Compute Cloud-Benutzerhandbuch.

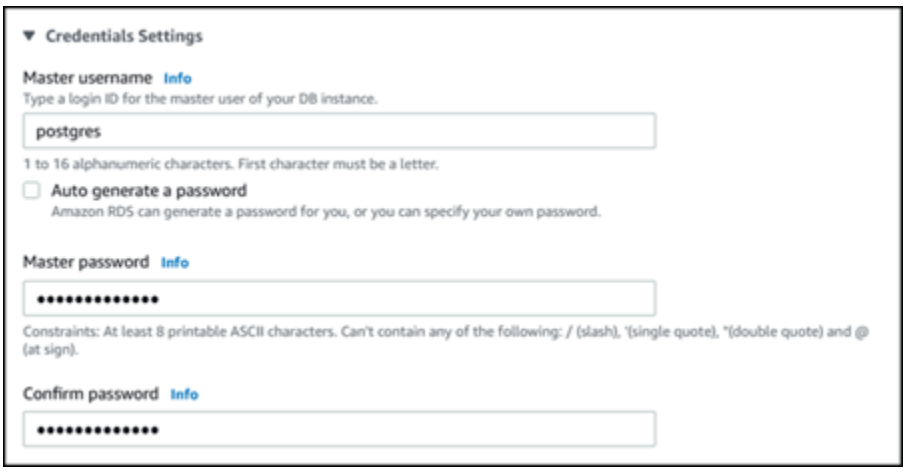
- Wenn Sie den PostgreSQL-Datenbanken, die auf Ihrem Amazon-Aurora-DB-Cluster ausgeführt werden, Berechtigungen erteilen möchten, können Sie den gleichen allgemeinen Ansatz wie bei eigenständigen Instances von PostgreSQL verwenden. Befehle wie `CREATE ROLE`, `ALTER ROLE`, `GRANT` und `REVOKE` funktionieren genau wie auf On-Premises-Datenbanken. Gleiches gilt für das direkte Ändern von Datenbanken, Schemas und Tabellen.

  PostgreSQL verwaltet Berechtigungen mithilfe von Rollen. Die Rolle `rds_superuser` ist die Rolle mit den meisten Berechtigungen in einem Aurora-PostgreSQL-DB-Cluster. Diese Rolle wird automatisch erstellt und dem Benutzer gewährt, der den DB-Cluster erstellt (das Hauptbenutzerkonto, standardmäßig `postgres`). Weitere Informationen hierzu finden Sie unter Grundlegendes zu PostgreSQL-Rollen und -Berechtigungen.

Alle verfügbaren Aurora PostgreSQL-Versionen, einschließlich der Versionen 10, 11, 12, 13, 14 und höher, unterstützen den Salted Challenge Response Authentication Mechanism (SCRAM) für Passwörter als Alternative zu Message Digest (). MD5 Wir empfehlen die Verwendung von SCRAM, da es sicherer ist als. MD5 Weitere Informationen, einschließlich der Migration von Datenbankbenutzerkennwörtern von MD5 zu SCRAM, finden Sie unter. Verwenden von SCRAM für die PostgreSQL-Passwortverschlüsselung

Grundlegendes zu PostgreSQL-Rollen und -Berechtigungen

Wenn Sie einen Aurora-PostgreSQL-DB-Cluster mit AWS-Managementkonsole erstellen, wird gleichzeitig ein Administratorkonto angelegt. Der Name lautet standardmäßig `postgres`, wie im folgenden Screenshot gezeigt:



Sie können einen anderen Namen auswählen, anstatt den Standard (`postgres`) beizubehalten. In diesem Fall muss der von Ihnen gewählte Name mit einem Buchstaben beginnen und zwischen 1 und 16 alphanumerische Zeichen umfassen. Der Einfachheit halber verwenden wir für das Hauptbenutzerkonto den Standardwert (`postgres`) in diesem Handbuch.

Wenn Sie die `create-db-cluster` AWS CLI statt der AWS-Managementkonsole verwenden, erstellen Sie den Namen, indem Sie ihn mit dem Parameter `master-username` weitergeben. Weitere Informationen finden Sie unter Schritt 2: Erstellen eines DB-Clusters von Aurora PostgreSQL.

Unabhängig davon, ob Sie die AWS-Managementkonsole, die AWS CLI oder die Amazon RDS-API verwenden und ob Sie den `postgres`-Standardnamen nutzen oder einen anderen Namen auswählen, ist dieses erste Datenbankbenutzerkonto Mitglied der `rds_superuser`-Gruppe und verfügt über `rds_superuser`-Berechtigungen.

Themen

- Die Rolle „rds_superuser“ verstehen
- Steuern des Benutzerzugriffs auf die PostgreSQL−Datenbank
- Delegieren und Steuern der Benutzerpasswortverwaltung
- Verwenden von SCRAM für die PostgreSQL-Passwortverschlüsselung

Die Rolle „rds_superuser“ verstehen

In PostgreSQL kann eine Rolle einen Benutzer, eine Gruppe oder einen Satz bestimmter Berechtigungen definieren, die einer Gruppe oder einem Benutzer für verschiedene Objekte in der Datenbank gewährt werden. PostgreSQL-Befehle für `CREATE USER` und `CREATE GROUP` wurden durch den allgemeineren Befehl `CREATE ROLE` mit bestimmten Eigenschaften zur Unterscheidung

von Datenbankbenutzern ersetzt. Einen Datenbankbenutzer kann man sich als Rolle mit der LOGIN-Berechtigung vorstellen.

> **Note**
>
> Die Befehle `CREATE USER` und `CREATE GROUP` können weiterhin verwendet werden. Weitere Informationen dazu finden Sie im Abschnitt Datenbankrollen der PostgreSQL-Dokumentation.

Der `postgres`-Benutzer ist der Datenbankbenutzer mit den meisten Berechtigungen auf Ihrem Aurora-PostgreSQL-DB-Cluster. Er verfügt über die Eigenschaften, die durch die folgende `CREATE ROLE`-Anweisung definiert sind.

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

Die Eigenschaften `NOSUPERUSER`, `NOREPLICATION`, `INHERIT` und `VALID UNTIL 'infinity'` sind die Standardoptionen für CREATE ROLE, sofern nicht anders angegeben.

Standardmäßig verfügt `postgres` über Berechtigungen, die der `rds_superuser`-Rolle gewährt wurden, und über Berechtigungen zum Erstellen von Rollen und Datenbanken. Die `rds_superuser`-Rolle erlaubt dem `postgres`-Benutzer, folgende Aktionen auszuführen:

- Erweiterungen für die Verwendung mit Aurora PostgreSQL. Weitere Informationen finden Sie unter Arbeiten mit Erweiterungen und Fremddaten-Wrappern.
- Rollen für Benutzer erstellen und Benutzern Berechtigungen gewähren. Weitere Informationen dazu finden Sie im Abschnitt CREATE ROLE und GRANT der PostgreSQL-Dokumentation.
- Datenbanken erstellen. Weitere Informationen finden Sie im Abschnitt CREATE DATABASE der PostgreSQL-Dokumentation.
- Gewähren Sie `rds_superuser`-Berechtigungen anderen Benutzerrollen, die nicht über diese Berechtigungen verfügen, und widerrufen Sie diese Berechtigungen bei Bedarf. Es wird empfohlen, diese Rolle nur denjenigen Benutzern zu gewähren, die Superuser-Aufgaben ausführen. Mit anderen Worten, Sie können diese Rolle Datenbankadministratoren (DBAs) oder Systemadministratoren erteilen.
- Die `rds_replication`-Rolle Datenbankbenutzern gewähren (oder entziehen), die nicht über die `rds_superuser`-Rolle verfügen.

- Die `rds_password`-Rolle Datenbankbenutzern gewähren (oder entziehen), die nicht über die `rds_superuser`-Rolle verfügen.
- Statusinformationen über alle Datenbankverbindungen über die Ansicht `pg_stat_activity` abrufen. Bei Bedarf kann `rds_superuser` alle Verbindungen mit `pg_terminate_backend` oder `pg_cancel_backend` stoppen.

In der `CREATE ROLE postgres...`-Anweisung können Sie sehen, dass die `postgres`-Benutzerrolle PostgreSQL ausdrücklich `superuser`-Berechtigungen verweigert. Aurora PostgreSQL ist ein verwalteter Service, sodass Sie nicht auf das Host-Betriebssystem zugreifen können und keine Verbindung mit dem `superuser`-PostgreSQL-Konto herstellen können. Viele der Aufgaben, die `superuser`-Zugriff auf einem eigenständigem PostgreSQL erfordern, werden von Aurora automatisch verwaltet.

Weitere Informationen zum Gewähren von Berechtigungen finden Sie unter GRANT in der PostgreSQL-Dokumentation.

Die `rds_superuser`-Rolle ist eine von mehreren vordefinierten Rollen in einem Aurora-PostgreSQL-DB-Cluster.

> **Note**
>
> In PostgreSQL 13 und früheren Versionen werden vordefinierte Rollen als Standardrollen bezeichnet.

In der folgenden Liste finden Sie einige der anderen vordefinierten Rollen, die automatisch für einen neuen Aurora-PostgreSQL-DB-Cluster erstellt werden. Vordefinierte Rollen und ihre Berechtigungen können nicht geändert werden. Sie können Berechtigungen für diese vordefinierten Rollen nicht löschen, umbenennen oder ändern. Jeder entsprechende Versuch führt zu einem Fehler.

- rds_passwort – Eine Rolle, die Passwörter ändern und Passwortbeschränkungen für Datenbankbenutzer einrichten kann. Die `rds_superuser`-Rolle erhält diese Rolle standardmäßig und kann die Rolle Datenbankbenutzern gewähren. Weitere Informationen finden Sie unter Steuern des Benutzerzugriffs auf die PostgreSQL−Datenbank.
  - In Versionen von RDS für PostgreSQL, die höher sind als 14, kann die `rds_password`-Rolle Passwörter ändern und Passwortbeschränkungen für Datenbankbenutzer und Benutzer mit der `rds_superuser`-Rolle einrichten. Ab Version 14 und höher von RDS für PostgreSQL

kann die `rds_password`-Rolle Passwörter ändern und Passwortbeschränkungen nur für Datenbankbenutzer einrichten. Nur Benutzer mit der `rds_superuser`-Rolle können diese Aktionen für andere Benutzer mit der `rds_superuser`-Rolle ausführen.

- rdsadmin – Eine Rolle, die erstellt wurde, um viele der Verwaltungsaufgaben zu erledigen, die der Administrator mit `superuser`-Berechtigungen für eine eigenständige PostgreSQL-Datenbank ausführt. Diese Rolle wird intern von Aurora PostgreSQL für viele Verwaltungsaufgaben verwendet.

Anzeigen von Rollen und deren Berechtigungen

Sie können vordefinierte Rollen und deren Berechtigungen in Ihrer DB-Instance von RDS für PostgreSQL je nach PostgreSQL-Version mit unterschiedlichen Befehlen anzeigen. Wenn Sie alle vordefinierten Rollen anzeigen möchten, können Sie eine Verbindung zu Ihrer DB-Instance von RDS für PostgreSQL herstellen und die folgenden Befehle mit `psql` ausführen.

Für **`psql`** 15 und frühere Versionen

Stellen Sie eine Verbindung zu Ihrer DB-Instance von RDS für PostgreSQL her und verwenden Sie den \du-Befehl in psql:

```
postgres=> \du
                                                          List of roles
    Role name    |                         Attributes                          |
                  Member of
-----------------+-----------------------------------------------------------------
+-------------------------------------------------
 postgres        | Create role, Create DB                                      +|
 {rds_superuser}
                 | Password valid until infinity                                |
 rds_ad          | Cannot login                                                 | {}
 rds_iam         | Cannot login                                                 | {}
 rds_password    | Cannot login                                                 | {}
 rds_replication | Cannot login                                                 | {}
 rds_superuser   | Cannot login                                                 |
 {pg_monitor,pg_signal_backend,rds_password,rds_replication}
 rdsadmin        | Superuser, Create role, Create DB, Replication, Bypass RLS+| {}
                 | Password valid until infinity                                |
```

Für **`psql`** 16 und höhere Versionen

```
postgres=> \drg+
```

```
                            List of role grants
   Role name   |           Member of           |       Options       | Grantor
---------------+-------------------------------+---------------------+----------
 postgres      | rds_superuser                 | INHERIT, SET        | rdsadmin
 rds_superuser | pg_checkpoint                 | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_monitor                    | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_signal_backend             | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_use_reserved_connections | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_password                  | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_replication               | ADMIN, INHERIT, SET | rdsadmin
```

Wenn Sie die Rollenmitgliedschaft ohne Versionsabhängigkeit überprüfen möchten, können Sie die folgende SQL-Abfrage verwenden:

```
SELECT m.rolname AS "Role name", r.rolname AS "Member of"
FROM pg_catalog.pg_roles m
JOIN pg_catalog.pg_auth_members pam ON (pam.member = m.oid)
LEFT JOIN pg_catalog.pg_roles r ON (pam.roleid = r.oid)
LEFT JOIN pg_catalog.pg_roles g ON (pam.grantor = g.oid)
WHERE m.rolname !~ '^pg_'
ORDER BY 1, 2;
```

In der Ausgabe sehen Sie, dass `rds_superuser` keine Datenbankbenutzerrolle ist (sie kann sich nicht anmelden), aber über die Berechtigungen vieler anderer Rollen verfügt. Sie können auch sehen, dass dieser Datenbankbenutzer `postgres` Mitglied der `rds_superuser`-Rolle ist. Wie bereits erwähnt, ist `postgres` der Standardwert auf der Seite Create database (Datenbank erstellen) der Amazon-RDS-Konsole. Wenn Sie einen anderen Namen gewählt haben, wird dieser Name stattdessen in der Rollenliste angezeigt.

> **Note**
>
> Aurora-PostgreSQL-Versionen 15.2 und 14.7 führten ein restriktives Verhalten der `rds_superuser`-Rolle ein. Einem Aurora PostgreSQL-Benutzer muss die `CONNECT`-Berechtigung für die entsprechende Datenbank zugewiesen werden, um eine Verbindung herzustellen, auch wenn dem Benutzer die `rds_superuser`-Rolle zugewiesen ist. Vor den Versionen 14.7 und 15.2 von Aurora PostgreSQL konnte ein Benutzer eine Verbindung zu jeder Datenbank und Systemtabelle herstellen, wenn dem Benutzer die `rds_superuser`-Rolle zugewiesen war. Dieses restriktive Verhalten passt zu den Verpflichtungen von AWS und Amazon Aurora zur kontinuierlichen Verbesserung der Sicherheit.

Bitte aktualisieren Sie die entsprechende Logik in Ihren Anwendungen, falls die oben genannte Erweiterung Auswirkungen hat.

Steuern des Benutzerzugriffs auf die PostgreSQL−Datenbank

Neue Datenbanken in PostgreSQL werden immer mit Standardberechtigungen im `public`-Schema der Datenbank erstellt, mit dem alle Datenbankbenutzer und -rollen Objekte erstellen können. Diese Berechtigungen ermöglichen es Datenbankbenutzern, eine Verbindung mit der Datenbank herzustellen und während der Verbindung temporäre Tabellen zu erstellen.

Es wird empfohlen, diese `public`-Standardberechtigungen zu widerrufen, um den Benutzerzugriff auf die Datenbank-Instances, die Sie auf Ihrem Primärknoten des Aurora-PostgreSQL-DB-Clusters erstellen, besser kontrollieren können. Danach erteilen Sie Datenbankbenutzern auf einer detaillierteren Basis spezifische Berechtigungen, wie im Folgenden gezeigt.

So richten Sie Rollen und Berechtigungen für eine neue Datenbank-Instance ein

Angenommen, Sie richten eine Datenbank für einen neu erstellten Aurora-PostgreSQL-DB-Cluster ein, die von mehreren Forschenden verwendet wird, die alle Lese-/Schreibzugriff auf die Datenbank benötigen.

1. Verwenden Sie `psql` (oder pgAdmin) zum Herstellen einer Verbindung mit dem primären Aurora-PostgreSQL-DB-Cluster:

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   Geben Sie bei der Aufforderung Ihr Passwort ein. Der `psql`-Client verbindet und zeigt die standardmäßige administrative Verbindungsdatenbank `postgres=>` als Eingabeaufforderung an.

2. Gehen Sie wie folgt vor, um zu verhindern, dass Datenbankbenutzer Objekte im `public`-Schema erstellen:

   ```
   postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
   REVOKE
   ```

3. Als Nächstes erstellen Sie eine neue Datenbank-Instance:

```
postgres=> CREATE DATABASE lab_db;
CREATE DATABASE
```

4. Widerrufen Sie alle Berechtigungen aus dem PUBLIC-Schema in dieser neuen Datenbank.

```
postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
REVOKE
```

5. Erstellen Sie eine Rolle für Datenbankbenutzer.

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. Geben Sie Datenbankbenutzern mit dieser Rolle die Möglichkeit, eine Verbindung mit der Datenbank herzustellen.

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. Gewähren Sie allen Benutzern mit der `lab_tech`-Rolle alle Berechtigungen für diese Datenbank.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. Erstellen Sie Datenbankbenutzer wie folgt:

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. Gewähren Sie diesen beiden Benutzern die Berechtigungen, die mit der lab_tech-Rolle verknüpft sind:

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

An dieser Stelle können `lab_user1` und `lab_user2` eine Verbindung mit der `lab_db`-Datenbank herstellen. Dieses Beispiel folgt nicht den bewährten Methoden für den Unternehmensgebrauch, darunter das Erstellen mehrerer Datenbank-Instances, verschiedener Schemas und das Erteilen eingeschränkter Berechtigungen. Umfassende Informationen und zusätzliche Szenarien finden Sie unter Verwalten von PostgreSQL-Benutzern und -Rollen.

Weitere Informationen zu Berechtigungen in PostgreSQL-Datenbanken finden Sie unter dem Befehl GRANT in der PostgreSQL-Dokumentation.

Delegieren und Steuern der Benutzerpasswortverwaltung

Als DBA sollten Sie ggf. die Verwaltung von Benutzerpasswörtern delegieren. Oder Sie möchten verhindern, dass Datenbankbenutzer ihre Passwörter ändern oder Passwortbeschränkungen wie die Lebensdauer des Passworts neu konfigurieren. Um sicherzustellen, dass nur die von Ihnen ausgewählten Datenbankbenutzer Passworteinstellungen ändern können, können Sie die Funktion zur eingeschränkten Passwortverwaltung aktivieren. Wenn Sie diese Funktion aktivieren, können nur die Datenbankbenutzer, denen die `rds_password`-Rolle gewährt wurde, Passwörter verwalten.

> **Note**
>
> Um die eingeschränkte Passwortverwaltung nutzen zu können, muss Ihr DB-Cluster von Aurora PostgreSQL Amazon Aurora PostgreSQL 10.6 oder höher ausführen.

Standardmäßig lautet diese Funktion `off`, wie im Folgenden gezeigt:

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

Zum Aktivieren dieser Funktion verwenden Sie eine benutzerdefinierte Parametergruppe und ändern die Einstellung für `rds.restrict_password_commands` in 1. Stellen Sie sicher, dass Sie Ihre primäre DB-Instance von Aurora PostgreSQL neu starten, damit die Einstellung wirksam wird.

Wenn diese Funktion aktiv ist, werden für die folgenden SQL-Befehle `rds_password`-Berechtigungen benötigt:

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
```

```
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

Das Umbenennen einer Rolle (`ALTER ROLE myrole RENAME TO newname`) ist auch eingeschränkt, wenn das Passwort den MD5-Hashing-Algorithmus verwendet.

Wenn diese Funktion aktiv ist, generiert jeder Versuch, einen dieser SQL-Befehle ohne die `rds_password`-Rollenberechtigungen auszuführen, den folgenden Fehler:

```
ERROR: must be a member of rds_password to alter passwords
```

Wir empfehlen, dass Sie die `rds_password`-Berechtigung nur wenigen Rollen zuweisen, die Sie ausschließlich für die Passwortverwaltung verwenden. Wenn Sie `rds_password`-Berechtigungen für Datenbankbenutzer erteilen, die keine `rds_superuser`-Berechtigungen haben, müssen Sie ihnen auch das `CREATEROLE`-Attribut erteilen.

Stellen Sie sicher, dass Sie die Passwortanforderungen wie Ablaufdatum und erforderliche Komplexität auf Kundenseite überprüfen. Wenn Sie Ihr eigenes clientseitiges Dienstprogramm für passwortbezogene Änderungen verwenden, muss das Dienstprogramm Mitglied von `rds_password` sein und über `CREATE ROLE`-Berechtigungen verfügen.

Verwenden von SCRAM für die PostgreSQL-Passwortverschlüsselung

Der Salted Challenge Response Authentication Mechanism (SCRAM) ist eine Alternative zum standardmäßigen Message Digest (MD5)-Algorithmus von PostgreSQL zum Verschlüsseln von Passwörtern. Der SCRAM-Authentifizierungsmechanismus gilt als sicherer als MD5. Weitere Informationen zu diesen beiden verschiedenen Ansätzen zur Sicherung von Passwörtern finden Sie unter Passwortauthentifizierung in der PostgreSQL-Dokumentation.

Wir empfehlen, SCRAM anstelle von MD5 als Passwortverschlüsselungsschema für den Aurora-PostgreSQL-DB-Cluster zu verwenden. SCRAM wird in Aurora-PostgreSQL-Version 10 und allen höheren Haupt- und Nebenversionen unterstützt. Es ist ein kryptografischer Challenge-Response-Mechanismus, der den scram–sha-256-Algorithmus zur Passwortauthentifizierung und -verschlüsselung nutzt.

Möglicherweise müssen Sie Bibliotheken aktualisieren, damit Ihre Clientanwendungen SCRAM unterstützen können. JDBC-Versionen vor 42.2.0 unterstützen SCRAM beispielsweise nicht. Weitere

Informationen finden Sie unter PostgreSQL-JDBC-Treiber in der Dokumentation zu PostgreSQL-JDBC-Treibern. Für eine Liste anderer PostgreSQL-Treiber und SCRAM-Unterstützung siehe die Treiberliste in der PostgreSQL-Dokumentation.

Aurora PostgreSQL Version 14 und höhere Versionen unterstützen scram-sha-256 standardmäßig zur Passwortverschlüsselung für neue DB-Cluster. Für diese Versionen ist der `password_encryption`-Wert der standardmäßigen DB-Cluster-Parametergruppe (`default.aurora-postgresql14`) auf scram-sha-256 festgelegt. Für Aurora Serverless v1 wird SCRAM nicht unterstützt.

Einrichten des Aurora-PostgreSQL-DB-Clusters, sodass SCRAM erforderlich ist

Bei Aurora PostgreSQL 14.3 und höheren Versionen können Sie verlangen, dass der Aurora-PostgreSQL-DB-Cluster nur Passwörter akzeptiert, die den scram-sha-256-Algorithmus verwenden.

> **Important**
>
> Wenn Sie bei vorhandenen RDS-Proxys mit PostgreSQL-Datenbanken die Datenbankauthentifizierung so ändern, dass nur `SCRAM` verwendet wird, ist der Proxy für bis zu 60 Sekunden nicht verfügbar. Um das Problem zu vermeiden, führen Sie einen der folgenden Schritte aus:
>
> - Stellen Sie sicher, dass die Datenbank sowohl die `SCRAM`- als auch die `MD5`-Authentifizierung zulässt.
> - Wenn Sie nur die `SCRAM`-Authentifizierung verwenden möchten, erstellen Sie einen neuen Proxy, migrieren Sie Ihren Anwendungsdatenverkehr auf den neuen Proxy und löschen Sie dann den zuvor mit der Datenbank verknüpften Proxy.

Bevor Sie Änderungen an Ihrem System vornehmen, vergewissern Sie sich, dass Sie den folgenden Prozess komplett verstehen:

- Sammeln Sie Informationen über alle Rollen und Passwortverschlüsselung für alle Datenbankbenutzer.
- Überprüfen Sie die Parametereinstellungen für Ihren Aurora-PostgreSQL-DB-Cluster für die Parameter, die die Passwortverschlüsselung steuern.
- Wenn Ihr Aurora-PostgreSQL-DB-Cluster eine Standardparametergruppe verwendet, müssen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe erstellen und sie auf Ihren Aurora-

PostgreSQL-DB-Cluster anwenden, damit Sie bei Bedarf Parameter ändern können. Wenn Ihr Aurora-PostgreSQL-DB-Cluster eine benutzerdefinierte Parametergruppe verwendet, können Sie die erforderlichen Parameter bei Bedarf später im Prozess ändern.

- Ändern Sie den Parameter `password_encryption` in `scram-sha-256`.
- Informieren Sie alle Datenbankbenutzer, dass sie ihre Passwörter aktualisieren müssen. Wiederholen Sie diesen Schritt für Ihr `postgres`-Konto. Die neuen Passwörter werden mit dem scram-sha-256-Algorithmus verschlüsselt und gespeichert.
- Stellen Sie sicher, dass alle Passwörter mit diesem Verschlüsselungstyp verschlüsselt sind.
- Wenn alle Passwörter scram-sha-256 verwenden, können Sie den `rds.accepted_password_auth_method`-Parameter von `md5+scram` in `scram-sha-256` ändern.

> **Warning**
>
> Nachdem Sie `rds.accepted_password_auth_method` nur in scram-sha-256 geändert haben, können Benutzer (Rollen) mit `md5`-verschlüsselten Passwörtern keine Verbindung herstellen.

Vorbereiten der SCRAM-Anforderung für Ihren Aurora-PostgreSQL-DB-Cluster

Bevor Sie Änderungen an Ihrem Aurora-PostgreSQL-DB-Cluster, vornehmen, überprüfen Sie alle vorhandenen Datenbankbenutzerkonten. Überprüfen Sie auch die Art der Verschlüsselung, die für Passwörter verwendet wird. Sie können für diese Aufgaben die `rds_tools`-Erweiterung verwenden. Weitere Informationen darüber, welche PostgreSQL-Versionen `rds_tools` unterstützen, finden Sie unter Erweiterungsversionen für Amazon RDS für PostgreSQL.

So erhalten Sie eine Liste der Datenbankbenutzer (Rollen) und Passwortverschlüsselungsmethoden

1. Verwenden Sie `psql` zum Herstellen einer Verbindung mit der primären Instance Ihres Aurora-PostgreSQL-DB-Clusters , wie im Folgenden gezeigt.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. Installieren Sie die `rds_tools`-Erweiterung.

```
postgres=> CREATE EXTENSION rds_tools;
CREATE EXTENSION
```

3. Rufen Sie eine Auflistung der Rollen und Verschlüsselungsmethoden ab.

```
postgres=> SELECT * FROM
      rds_tools.role_password_encryption_type();
```

Die Ausgabe entspricht weitgehend der folgenden.

```
       rolname        | encryption_type
----------------------+-----------------
 pg_monitor           |
 pg_read_all_settings |
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | md5
 user_465             | md5
 postgres             | md5
(8 rows)
```

Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe

> **Note**
>
> Wenn Ihr Aurora-PostgreSQL-DB-Cluster bereits eine benutzerdefinierte Parametergruppe verwendet, müssen Sie keine neue erstellen.

Eine Übersicht über Parametergruppen für Aurora finden Sie unter Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora.

Der für Passwörter verwendete Passwortverschlüsselungstyp wird in einem Parameter, `password_encryption`, festgelegt. Die Verschlüsselung, die der Aurora-PostgreSQL-DB-Cluster zulässt, wird in einem anderen Parameter, `rds.accepted_password_auth_method`, festgelegt. Wenn Sie den Standardwert eines dieser Parameter ändern, müssen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe erstellen und auf Ihren Cluster anwenden.

Sie können auch die AWS-Managementkonsole oder die RDS-API zum Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe verwenden. Weitere Informationen finden Sie unter Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora.

Sie können jetzt die benutzerdefinierte Parametergruppe Ihrer DB-Instance zuordnen.

So erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe

1. Verwenden Sie den CLI-Befehl `create-db-cluster-parameter-group` zum Erstellen der benutzerdefinierten Parametergruppe für den Cluster. Folgendes Beispiel verwendet `aurora-postgresql13` als Quelle für diese benutzerdefinierte Parametergruppe.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
     --db-parameter-group-family aurora-postgresql13  --description 'Custom DB cluster parameter group for SCRAM'
   ```

   Für Windows:

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
     --db-parameter-group-family aurora-postgresql13  --description "Custom DB cluster parameter group for SCRAM"
   ```

   Sie können jetzt die benutzerdefinierte Parametergruppe Ihrem Cluster zuordnen.

2. Verwenden Sie den CLI-Befehl `modify-db-cluster` zum Anwenden dieser benutzerdefinierten Parametergruppe auf Ihren Aurora-PostgreSQL-DB-Cluster.

   Für Linux, macOS oder Unix:

   ```
   aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

   Für Windows:

   ```
   aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

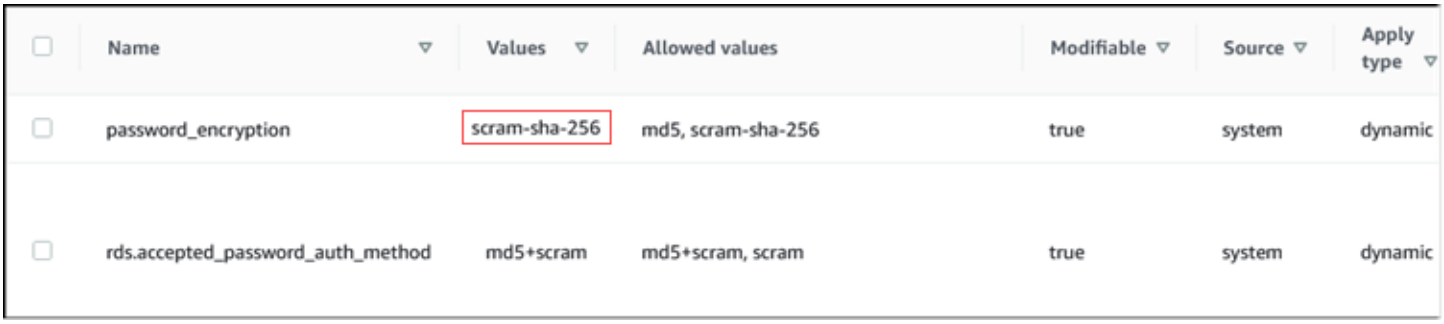
Zum erneuten Synchronisieren Ihres Aurora-PostgreSQL-DB-Clusters mit Ihrer benutzerdefinierten DB-Cluster-Parametergruppe starten Sie die primäre und alle anderen Instances des Clusters neu.

Konfigurieren der Passwortverschlüsselung für die Verwendung von SCRAM

Der Passwortverschlüsselungsmechanismus, der von einem Aurora-PostgreSQL-DB-Cluster verwendet wird, ist in der DB-Cluster-Parametergruppe auf den Parameter `password_encryption` festgelegt. Zulässige Werte sind keine Angabe, `md5` oder `scram-sha-256`. Der Standardwert hängt von der Version von Aurora PostgreSQL wie folgt ab:

- Aurora PostgreSQL 14 – Der Standardwert ist `scram-sha-256`.
- Aurora PostgreSQL 13 – Der Standardwert ist `md5`.

Mit einer benutzerdefinierten DB-Cluster-Parametergruppe , die Ihrem Aurora-PostgreSQL-DB-Cluster angefügt ist, können Sie die Werte für den Passwortverschlüsselungsparameter ändern.



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

So ändern Sie die Passwortverschlüsselungseinstellung in scram-sha-256

- Ändern Sie den Wert der Passwortverschlüsselung in scram-sha-256, wie nachfolgend gezeigt. Die Änderung kann sofort angewendet werden, da der Parameter dynamisch ist. Daher ist kein Neustart erforderlich, damit die Änderung wirksam wird.

  Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name \
  'docs-lab-scram-passwords' --parameters
 'ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate'
```

Für Windows:

```
aws rds modify-db-parameter-group --db-parameter-group-name ^
  "docs-lab-scram-passwords" --parameters
 "ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate"
```

Migrieren von Passwörtern für Benutzerrollen zu SCRAM

Sie können Passwörter für Benutzerrollen wie folgt zu SCRAM migrieren.

So migrieren Sie Passwörter für Datenbankbenutzer (Rolle) von MD5 zu SCRAM

1. Melden Sie sich als Administratorbenutzer an (Standardbenutzer `postgres`), wie nachfolgend gezeigt.

   ```
   psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. Überprüfen Sie die Einstellung des `password_encryption`-Parameters auf der DB-Instance von RDS für PostgreSQL mithilfe des folgenden Befehls.

   ```
   postgres=> SHOW password_encryption;
    password_encryption
   ---------------------
    md5
    (1 row)
   ```

3. Ändern Sie den Wert dieses Parameters in scram-sha-256. Weitere Informationen finden Sie unter Konfigurieren der Passwortverschlüsselung für die Verwendung von SCRAM.

4. Überprüfen Sie den Wert erneut, um sicherzustellen, dass er jetzt auf `scram-sha-256` festgelegt ist. Gehen Sie dazu wie folgt vor.

   ```
   postgres=> SHOW password_encryption;
    password_encryption
   ---------------------
    scram-sha-256
    (1 row)
   ```

5. Bitten Sie alle Datenbankbenutzer, ihre Passwörter zu ändern. Stellen Sie sicher, dass Sie auch Ihr eigenes Passwort für das `postgres`-Konto ändern (der Datenbankbenutzer mit `rds_superuser`-Berechtigungen).

```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

6. Wiederholen Sie den Vorgang für alle Datenbanken auf dem Aurora-PostgreSQL-DB-Cluster.

Ändern des Parameters, um SCRAM zu erfordern

Dies ist der letzte Schritt in diesem Prozess. Nachdem Sie die Änderung im folgenden Verfahren vorgenommen haben, können sich Benutzerkonten (Rollen), die nach wie vor die `md5`-Verschlüsselung für Passwörter verwenden, nicht beim Aurora-PostgreSQL-DB-Cluster anmelden.

Die `rds.accepted_password_auth_method` gibt die Verschlüsselungsmethode an, die der Aurora-PostgreSQL-DB-Cluster als Benutzerpasswort beim Anmeldevorgang akzeptiert. Der Standardwert ist `md5+scram`, was bedeutet, dass eine der beiden Methoden akzeptiert wird. Im folgenden Bild finden Sie die Standardeinstellung für diesen Parameter.

Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

Die zulässigen Werte für diesen Parameter sind `md5+scram` oder nur `scram`. Durch Ändern dieses Parameterwerts in `scram` wird er erforderlich.

So ändern Sie den Parameterwert, sodass eine SCRAM-Authentifizierung für Passwörter erforderlich ist

1. Stellen Sie sicher, dass alle Datenbank-Benutzerpasswörter für alle Datenbanken auf Ihrem Aurora-PostgreSQL-DB-Cluster `scram-sha-256` für die Passwortverschlüsselung verwenden. Fragen Sie hierzu `rds_tools` für die Rolle (Benutzer) und den Verschlüsselungstyp wie folgt ab.

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
```

```
 rolname        | encryption_type
 ---------------------+-----------------
 pg_monitor           |
 pg_read_all_settings |
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | scram-sha-256
 user_465             | scram-sha-256
 postgres             | scram-sha-256
 ( rows)
```

2. Wiederholen Sie die Abfrage auf allen DB-Instances Ihres Aurora-PostgreSQL-DB-Clusters.

   Wenn alle Passwörter scram-sha-256 verwenden, können Sie fortfahren.

3. Ändern Sie den Wert der akzeptierten Passwortauthentifizierung in scram-sha-256 wie folgt.

   Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-
lab-scram-passwords' \
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

   Für Windows:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-
lab-scram-passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

Sicherung von Aurora-PostgreSQL-Daten mit SSL/TLS

Amazon RDS unterstützt Secure Socket Layer (SSL) und Transport Layer Security (TLS) Verschlüsselung für Aurora PostgreSQL DB-Cluster. Verwenden von SSL/TLS, you can encrypt a connection between your applications and your Aurora PostgreSQL DB clusters. You can also force all connections to your Aurora PostgreSQL DB cluster to use SSL/TLS Amazon Aurora PostgreSQL unterstützt jetzt Transport Layer Security (TLS) in den Versionen 1.1 und 1.2. Wir empfehlen die Verwendung von TLS 1.2 für verschlüsselte Verbindungen. Wir haben Unterstützung für TLSv1 .3 aus den folgenden Versionen von Aurora PostgreSQL hinzugefügt:

- 15.3 und alle höheren Versionen
- 14.8 und höhere 14-Versionen
- 13.11 und höhere 13-Versionen
- 12.15 und höhere 12-Versionen
- 11.20 und höhere 11-Versionen

Allgemeine Informationen zur SSL/TLS Unterstützung und zu PostgreSQL-Datenbanken finden Sie unter SSL-Unterstützung in der PostgreSQL-Dokumentation. Informationen zur Verwendung einer SSL/TLS Verbindung über JDBC finden Sie unter Konfiguration des Clients in der PostgreSQL-Dokumentation.

Themen

- Erfordernis einer SSL/TLS-Verbindung zu einem Aurora PostgreSQL DB-Cluster
- Ermitteln des SSL/TLS-Verbindungsstatus
- Konfigurieren von Chiffrier-Suiten für Verbindungen mit Aurora-PostgreSQL-DB-Clustern

SSL/TLS-Unterstützung ist in allen AWS Regionen für Aurora PostgreSQL verfügbar. Amazon RDS erstellt ein SSL/TLS Zertifikat für Ihren Aurora PostgreSQL-DB-Cluster, wenn der DB-Cluster erstellt wird. Wenn Sie die SSL/TLS Zertifikatsüberprüfung aktivieren, enthält das SSL/TLS Zertifikat den DB-Cluster-Endpunkt als Common Name (CN) für das SSL/TLS Zertifikat zum Schutz vor Spoofing-Angriffen.

Um sich mit einem Aurora PostgreSQL DB-Cluster über SSL/TLS zu verbinden

1. Laden Sie das Zertifikat herunter.

   Informationen zum Herunterladen von Zertifikaten finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.
2. Importieren Sie das Zertifikat in Ihr Betriebssystem.
3. Verbinden Sie sich mit Ihrem Aurora PostgreSQL DB-Cluster über SSL/TLS.

   Wenn Sie eine Verbindung über SSL/TLS herstellen, kann Ihr Client wählen, ob er die Zertifikatskette überprüfen möchte oder nicht. Wenn Ihre Verbindungsparameter `sslmode=verify-ca` oder `sslmode=verify-full` angeben, verlangt Ihr Client, dass sich die RDS CA-Zertifikate im Trust Store befinden oder von der Verbindungs-URL referenziert

werden. Diese Anforderung dient zur Prüfung der Zertifikatskette, die Ihr Datenbankzertifikat signiert.

Wenn ein Client, z. B. psql oder JDBC, mit SSL/TLS Support konfiguriert ist, versucht der Client standardmäßig zunächst, eine Verbindung mit der Datenbank herzustellen. SSL/TLS Wenn der Client keine Verbindung herstellen kann mit. SSL/TLS, it reverts to connecting without SSL/TLS Standardmäßig ist die `sslmode`-Option für JDBC- und libpq-basierte Clients auf `prefer` festgelegt.

Verwenden Sie den Parameter `sslrootcert`, um auf das Zertifikat zu verweisen, beispielsweise `sslrootcert=rds-ssl-ca-cert.pem`.

Das folgende Beispiel zeigt psql zur Herstellung einer Verbindung mit einem Aurora-PostgreSQL-DB-Cluster:

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
    "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

Erfordernis einer SSL/TLS-Verbindung zu einem Aurora PostgreSQL DB-Cluster

Verwenden `rds.force_ssl` Sie Parameter, um SSL/TLS Verbindungen zu Ihrem Aurora PostgreSQL-DB-Cluster zu benötigen.

- Um SSL/TLS Verbindungen zu benötigen, setzen Sie den `rds.force_ssl` Parameterwert auf 1 (ein).
- Um die erforderlichen SSL/TLS Verbindungen auszuschalten, setzen Sie den `rds.force_ssl` Parameterwert auf 0 (aus).

Der Standardwert dieses Parameters hängt von der Aurora-PostgreSQL-Version ab:

- Für Aurora-PostgreSQL-Versionen 17 und höher: Der Standardwert ist 1 (on).
- Für Aurora-PostgreSQL-Versionen 16 und niedriger: Der Standardwert ist 0 (off).

> **Note**
>
> Wenn Sie ein Hauptversions-Upgrade von Aurora-PostgreSQL-Version 16 oder früher auf Version 17 oder höher durchführen, ändert sich der Standardwert des Parameters von 0 (off)

auf 1 (on). Diese Änderung kann zu Verbindungsfehlern bei Anwendungen führen, die nicht für SSL konfiguriert sind. Sie können zum vorherigen Standardverhalten zurückkehren, indem Sie diesen Parameter auf 0 (off) setzen.

Weitere Informationen zur Handhabung von Parametern finden Sie unter Parametergruppen für Amazon Aurora.

Durch die Aktualisierung des `rds.force_ssl` Parameters wird auch der `ssl` PostgreSQL-Parameter auf 1 (on) gesetzt und die `pg_hba.conf` Datei Ihres DB-Clusters so geändert, dass sie die neue Konfiguration unterstützt. SSL/TLS

Wenn der `rds.force_ssl` Parameter für einen DB-Cluster auf 1 gesetzt ist, erhalten Sie beim Herstellen der Verbindung eine Ausgabe, die der folgenden ähnelt, was darauf hinweist, dass dies jetzt erforderlich SSL/TLS ist:

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

Ermitteln des SSL/TLS-Verbindungsstatus

Der Verschlüsselungsstatus Ihrer Verbindung wird auf dem Anmelde-Banner angezeigt, wenn Sie sich mit dem DB-Cluster verbinden:

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

Sie können die `sslinfo` Erweiterung auch laden und dann die `ssl_is_used()` Funktion aufrufen, um festzustellen, ob sie verwendet SSL/TLS wird. Die Funktion gibt `t` zurück, wenn die Verbindung SSL/TLS verwendet, andernfalls gibt sie `f` zurück.

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
---------
t
(1 row)
```

Sie können den Befehl `select ssl_cipher()` verwenden, um die SSL/TLS-Verschlüsselung zu bestimmen:

```
postgres=> select ssl_cipher();
ssl_cipher
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

Wenn Sie `set rds.force_ssl` aktivieren und Ihren DB-Cluster neu starten, werden Nicht-SSL-Verbindungen abgelehnt, und die folgende Mitteilung wird angezeigt:

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database
 "postgres", SSL off
$
```

Informationen zur Option `sslmode` finden Sie unter Datenbankverbindung-Steuerungsfunktionen in der PostgreSQL-Dokumentation.

Konfigurieren von Chiffrier-Suiten für Verbindungen mit Aurora-PostgreSQL-DB-Clustern

Durch die Verwendung von konfigurierbaren Chiffrier-Suiten können Sie mehr Kontrolle über die Sicherheit Ihrer Datenbankverbindungen haben. Sie können eine Liste von Cipher Suites angeben, denen Sie die Sicherung von SSL/TLS Client-Verbindungen zu Ihrer Datenbank ermöglichen möchten. Mit konfigurierbaren Chiffrier-Suiten können Sie die Verbindungsverschlüsselung steuern, die Ihr Datenbankserver akzeptiert. Dies hilft, die Verwendung von unsicheren oder veralteten Chiffren zu verhindern.

Konfigurierbare Chiffrier-Suiten werden in Aurora PostgreSQL Version 11.8 und höher unterstützt.

Um die Liste der zulässigen Chiffren für die Verschlüsselung von Verbindungen anzugeben, ändern Sie die `ssl_ciphers`-Cluster-Parameter. Setzen Sie den `ssl_ciphers` Parameter mithilfe der, der oder der RDS-API auf eine Zeichenfolge mit kommagetrennten Chiffrierwerten in einer Cluster-Parametergruppe. AWS-Managementkonsole AWS CLI Um Cluster-Parameter festzulegen, siehe Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

Die folgende Tabelle zeigt die unterstützten Verschlüsselungen für die gültigen Versionen der Aurora-PostgreSQL-Engine.

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
9.6, 10.20 und niedriger, 11.15 und niedriger, 12.10 und niedriger, 13.6 und niedriger	DHE-RSA- -SHA AES128	Ja	Nein	Nein
	DHE-RSA- AES128 -SHA256	Nein	Ja	Nein
	DHE-RSA- -GCM-AES128 SHA256	Nein	Ja	Nein
	AES256DHE-RSA-SHA	Nein	Ja	Nein
	DHE-RSA- AES256 -SHA256	Nein	Ja	Nein
		Ja	Ja	Nein
		Nein	Ja	Nein
		Nein	Ja	Nein

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
	DHE-RSA- -GCM- AES256 SHA384	Ja	Ja	Nein
	AES256ECDHE-ECDSA-SHA	Nein	Ja	Nein
	ECDHE-ECDSA- -GCM- AES256 SHA384	Nein	Ja	Nein
	ECDHE-RSA- - AES256 SHA384	Ja	Ja	Nein
	ECDHE-RSA- AES128 -SHA	Nein	Ja	Nein
	ECDHE-RSA- - AES128 SHA256			
	ECDHE-RSA- -GCM AES128 - SHA256			
	AES256ECDHE-RSA-SHA			
	ECDHE-RSA- AES256 -GCM- SHA384			

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
10.21, 11.16, 12.11, 13.7, 14.3 und 14.4	ECDHE-RSA-AES128 -SCHÄTZE_	Ja	Ja	Nein
	ECDHE-RSA _WITH_AES _128_CBC_SHA	Nein	Ja	Nein
		Ja	Ja	Nein
	TLS_ECDHE _RSA_WITH _AES_128_GCM_ SHA256	Nein	Ja	Nein
		Ja	Ja	Nein
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Nein	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE _RSA_WITH _AES_256_GCM_ SHA384	Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE _RSA_WITH _AES_128_GCM_ SHA256			
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA			
	TLS_ECDHE _RSA_WITH			

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
	_AES_256_GCM_ SHA384 TLS_RSA_M IT_AES_256_GCM_ SHA384 TLS_RSA_W ITH_AES_2 56_CBC_SHA TLS_RSA_M IT_AES_128_GCM_ SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA CHACHA2TL S_ECDHE_RSA_MIT_ 0_ 05_ POLY13 SHA256			

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
10.22, 11.17, 12.12, 13.8, 14.5 und 15.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Nein	Ja	Nein
		Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Ja	Ja	Nein
		Nein	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Nein	Ja	Nein
		Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Ja	Ja	Nein
		Ja	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256			

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA TLS_ECDHE _RSA_WITH _AES_256_GCM_ SHA384 TLS_RSA_M IT_AES_256_GCM_ SHA384 TLS_RSA_W ITH_AES_2 56_CBC_SHA TLS_RSA_M IT_AES_128_GCM_ SHA256 TLS_RSA_M IT_AES_128_CBC_ SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA CHACHA2TL S_ECDHE_RSA_MIT_ 0_ 05_ POLY13 SHA256			

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
11.20, 12.15, 13.11, 14.8, 15.3, 16.1 und höher	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Nein	Ja	Nein
		Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Ja	Ja	Nein
		Nein	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Nein	Ja	Nein
		Ja	Ja	Nein
		Nein	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Ja	Ja	Nein
		Ja	Ja	Nein
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Nein	Ja	Nein
		Nein	Nein	Ja
		Nein	Nein	Ja
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256			

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA TLS_ECDHE _RSA_WITH _AES_256_GCM_ SHA384 TLS_RSA_M IT_AES_256_GCM_ SHA384 TLS_RSA_W ITH_AES_2 56_CBC_SHA TLS_RSA_M IT_AES_128_GCM_ SHA256 TLS_RSA_M IT_AES_128_CBC_ SHA256 TLS_RSA_W ITH_AES_1 28_CBC_SHA CHACHA2TL S_ECDHE_RSA_MIT_ 0_ 05_ POLY13 SHA256 TLS_AES_128_GCM_ SHA256			

Engine-Versionen für Aurora PostgreSQL	Unterstützte Verschlüsselungen	TLS 1.1	TLS 1.2	TLS 1.3
	TLS_AES_256_GCM_SHA384			

Sie können auch den CLI-Befehl describe-engine-default-cluster-parameters verwenden, um zu ermitteln, welche Cipher Suites derzeit für eine bestimmte Parametergruppenfamilie unterstützt werden. Im folgenden Beispiel wird veranschaulicht, wie Sie die zulässigen Werte für `ssl_cipher`-Cluster-Parameter für Aurora PostgreSQL 11 abrufen.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postgresql11

    ...some output truncated...
 {
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure
 connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,
  ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-
SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-
SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,

 TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,T

 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
    ...some output truncated...
```

Der Parameter `ssl_ciphers` ist standardmäßig auf alle zulässigen Verschlüsselungssuites eingestellt. Weitere Informationen zu Chiffren finden Sie in der ssl_ciphers-Variablen in der PostgreSQL-Dokumentation.

Dynamische Maskierung mit Aurora PostgreSQL verwenden

Dynamische Datenmaskierung ist eine Sicherheitsfunktion, die sensible Daten in Aurora PostgreSQL-Datenbanken schützt, indem gesteuert wird, wie Daten Benutzern bei Abfragen angezeigt werden. Aurora implementiert es über die `pg_columnmask` Erweiterung. `pg_columnmask`bietet Datenschutz auf Spaltenebene, der die systemeigenen Sicherheits- und granularen Zugriffskontrollmechanismen von PostgreSQL auf Zeilenebene ergänzt.

Mit erstellen Sie Maskierungsrichtlinien`pg_columnmask`, die die Datentransparenz auf der Grundlage von Benutzerrollen festlegen. Wenn Benutzer Tabellen mit Maskierungsrichtlinien abfragen, wendet Aurora PostgreSQL zur Abfragezeit die entsprechende Maskierungsfunktion an, die auf der Rolle und der Richtliniengewichtung des Benutzers basiert. Die zugrunde liegenden Daten bleiben unverändert im Speicher.

`pg_columnmask`unterstützt die folgenden Funktionen:

- Integrierte und benutzerdefinierte Maskierungsfunktionen — Verwenden Sie vorgefertigte Funktionen für gängige Muster wie E-Mail- und Textmaskierung oder erstellen Sie Ihre eigenen benutzerdefinierten Funktionen zum Schutz sensibler Daten (PII) mithilfe von SQL-basierten Maskierungsrichtlinien.
- Mehrere Maskierungsstrategien — Verbergen Sie Informationen vollständig, ersetzen Sie Teilwerte durch Platzhalter oder definieren Sie benutzerdefinierte Maskierungsansätze.
- Priorisierung von Richtlinien — Definieren Sie mehrere Richtlinien für eine einzelne Spalte. Verwenden Sie Gewichtungen, um zu bestimmen, welche Maskierungsrichtlinie verwendet werden soll, wenn mehrere Richtlinien für eine Spalte gelten. Aurora PostgreSQL wendet Richtlinien an, die auf Gewicht und Benutzerrollenzugehörigkeit basieren.

`pg_columnmask`ist auf Aurora PostgreSQL Version 16.10 und höher und Version 17.6 und höher verfügbar. Es ist ohne zusätzliche Kosten verfügbar.

Erste Schritte mit dynamischer Maskierung

Um Daten dynamisch zu maskieren, installieren Sie die `pg_columnmask` Erweiterung in Ihrer Datenbank und erstellen Maskierungsrichtlinien für Ihre Tabellen. Der Einrichtungsprozess umfasst

die Überprüfung der Voraussetzungen, die Installation der Erweiterung, die Rollenkonfiguration, die Erstellung von Richtlinien und die Überprüfung von Validierungstests.

Installation und Konfiguration von Erweiterungen

Stellen Sie mit dem RDS Console Query Editor oder einem PostgreSQL-Client wie psql mit rds_superuser (Masteruser) -Anmeldeinformationen eine Connect zu Ihrem Aurora PostgreSQL-Cluster her.

Führen Sie den Befehl zur Erstellung der Erweiterung aus, um die Funktionalität zu aktivieren: `pg_columnmask`

```
CREATE EXTENSION pg_columnmask;
```

Mit diesem Befehl wird die `pg_columnmask` Erweiterung installiert, die erforderlichen Katalogtabellen erstellt und die integrierten Maskierungsfunktionen registriert. Die Installation der Erweiterung ist datenbankspezifisch, was bedeutet, dass Sie sie in jeder Datenbank, in der die Funktionalität benötigt wird, separat installieren müssen.

> **Note**
>
> Bei Verbindungen, die vor der Installation dieser Erweiterung hergestellt wurden, werden weiterhin unmaskierte Daten angezeigt. Schließen Sie und stellen Sie die Verbindung erneut her, um dieses Problem zu beheben.

Überprüfen Sie die Installation der Erweiterung, indem Sie die verfügbaren Maskierungsfunktionen überprüfen:

```
SELECT proname FROM pg_proc
    WHERE pronamespace = 'pgcolumnmask'::regnamespace AND proname LIKE 'mask_%';
    proname
--------Output --------
 mask_email
 mask_text
 mask_timestamp
(3 rows)
```

Verfahren zur Verwaltung von Datenmaskierungsrichtlinien

Sie können Maskierungsrichtlinien mithilfe der von der Erweiterung bereitgestellten Verfahren verwalten. `pg_columnmask` Um Maskierungsrichtlinien zu erstellen, zu ändern oder zu löschen, benötigen Sie eines der folgenden Rechte:

- Besitzer der Tabelle, für die Sie die `pg_columnmask` Richtlinie erstellen.
- Mitglied von`rds_superuser`.
- Mitglied der durch den `pgcolumnmask.policy_admin_rolname` Parameter festgelegten Rolle des `pg_columnmask` Policy-Managers.

Mit dem folgenden Befehl wird eine Tabelle erstellt, die in nachfolgenden Abschnitten verwendet wird:

```
CREATE TABLE public.customers (
    id SERIAL PRIMARY KEY,
    name TEXT,
    phone TEXT,
    address TEXT,
    email TEXT
);
```

CREATE_MASKING_POLICY

Das folgende Verfahren erstellt eine neue Maskierungsrichtlinie für eine Benutzertabelle:

Syntax

```
create_masking_policy(
    policy_name,
    table_name,
    masking_expressions,
    roles,
    weight)
```

Argumente

Parameter	Datatype	Description
`policy_name`	NAME	Der Name der Maskierun gsrichtlinie. Muss pro Tabelle eindeutig sein.
`table_name`	REGCLASS	Der qualified/unqualified Name oder die OID der Tabelle, auf die die Maskierungsrichtlinie angewendet werden soll.
`masking_expressions`	JSONB	JSON-Objekt, das Paare aus Spaltennamen und Maskierun gsfunktionen enthält. Jeder Schlüssel ist ein Spaltenna me und sein Wert ist der Maskierungsausdruck, der auf diese Spalte angewendet werden soll.
`roles`	NAME []	Die Rollen, für die diese Maskierungsrichtlinie gilt. Die Standardeinstellung ist PUBLIC.
`weight`	INT	Gewicht der Maskierungsrichtli nie. Wenn für die Anfrage eines bestimmten Benutzers mehrere Richtlinien gelten, wird die Richtlinie mit der höchsten Gewichtung (höhere Ganzzahl) auf jede maskierte Spalte angewendet. Standard = 0. Keine zwei Maskierungsrichtlinien in

Parameter	Datatype	Description
		der Tabelle können dieselbe Gewichtung haben.

Rückgabetyp

Keine

Example beim Erstellen einer Maskierungsrichtlinie, die die E-Mail-Spalte für die Rolle maskiert: **`test_user`**

```
CALL pgcolumnmask.create_masking_policy(
    'customer_mask',
    'public.customers',
    JSON_OBJECT('{
        "email", "pgcolumnmask.mask_email(email)"
    }')::JSONB,
    ARRAY['test_user'],
    100
);
```

ALTER_MASKING_POLICY

Dieses Verfahren ändert eine bestehende Maskierungsrichtlinie. `ALTER_MASKING_POLICY`kann die Maskierungsausdrücke der Richtlinie, die Gruppe der Rollen, für die die Richtlinie gilt, und die Gewichtung der Maskierungsrichtlinie ändern. Wenn einer dieser Parameter weggelassen wird, bleibt der entsprechende Teil der Richtlinie unverändert.

Syntax

```
alter_masking_policy(
    policy_name,
    table_name,
    masking_expressions,
    roles,
    weight)
```

Argumente

Parameter	Datatype	Description
`policy_name`	NAME	Bestehender Name der Maskierungsrichtlinie.
`table_name`	REGCLASS	Die qualified/unqualified Name-OID der Tabelle, die die Maskierungsrichtlinie enthält.
`masking_expressions`	JSONB	Neues JSON-Objekt mit Spaltennamen und Maskierungsfunktionspaaren oder andernfalls NULL.
`roles`	NAME []	Die Liste der neuen Rollen, für die diese Maskierungsrichtlinie gilt, andernfalls NULL.
`weight`	INT	Neue Gewichtung für die Maskierungsrichtlinie oder andernfalls NULL.

Rückgabetyp

Keine

Example zum Hinzufügen der Analystenrolle zu einer vorhandenen Maskierungsrichtlinie, ohne andere Richtlinienattribute zu ändern.

```
CALL pgcolumnmask.alter_masking_policy(
    'customer_mask',
    'public.customers',
    NULL,
    ARRAY['test_user', 'analyst'],
    NULL
);

-- Alter the weight of the policy without altering other details
CALL pgcolumnmask.alter_masking_policy(
```

```
    'customer_mask',
    'customers',
    NULL,
    NULL,
    4
);
```

DROP_MASKING_POLICY

Durch dieses Verfahren wird eine bestehende Maskierungsrichtlinie entfernt.

Syntax

```
drop_masking_policy(
        policy_name,
        table_name)
```

Argumente

Parameter	Datatype	Description
policy_name	NAME	Bestehender Name der Maskierungsrichtlinie.
table_name	REGCLASS	Die qualified/unqualified Name-OID der Tabelle, die die Maskierungsrichtlinie enthält.

Rückgabetyp

Keine

Example beim Löschen der Maskierungsrichtlinie customer_mask

```
-- Drop a masking policy
    CALL pgcolumnmask.drop_masking_policy(
        'customer_mask',
        'public.customers',
    );
```

Umgehen von Identifikatoren in der DDL-Prozedur für Maskierungsrichtlinien

Bei der Erstellung von Datenmaskierungsrichtlinien mit Identifikatoren in Anführungszeichen ist ein korrektes Escaping erforderlich, um sicherzustellen, dass die Objektverweise und die Richtlinien korrekt angewendet werden. Um Bezeichner in Anführungszeichen innerhalb der Verfahren zur Verwaltung von `pg_columnmask` Maskierungsrichtlinien zu verwenden, gehen Sie wie folgt vor:

- Richtlinienname — Muss in doppelte Anführungszeichen gesetzt werden.
- Tabellenname — Sowohl der Schemaname als auch der Tabellenname müssen bei Bedarf einzeln in doppelte Anführungszeichen gesetzt werden.
- Maskierungsausdrücke — Spalten- und Funktionsnamen in Maskierungsausdrücken müssen in doppelte Anführungszeichen eingeschlossen werden, und die Anführungszeichen selbst müssen mit einem umgekehrten Schrägstrich maskiert werden.
- Rollen — Das Array der Rollennamen wird automatisch in Anführungszeichen gesetzt. Der Rollenname sollte genau dem Namen entsprechen, wie er unter angegeben ist, `pg_roles` einschließlich Groß- und Kleinschreibung.

Example der Syntax mit Escape-Zeichen und Anführungszeichen

Dieses Beispiel zeigt die richtige Syntax für Escapes und Anführungszeichen bei der Erstellung von Maskierungsrichtlinien für Tabellen, Spalten, Funktionen und Rollen, die Namen in Groß- und Kleinschreibung verwenden oder Bezeichner in Anführungszeichen in Aurora PostgreSQL erfordern.

```
-- Create a table and columns with mixed case name
CREATE TABLE public."Employees" (
    "Name" TEXT,
    "Email" TEXT,
    ssn VARCHAR(20)
);

-- Create a role with mixed case name
CREATE ROLE "Masked_user";

-- Create a function with mixed case name
CREATE OR REPLACE FUNCTION public."MaskEmail"(text)
    RETURNS character varying
    LANGUAGE plpgsql
    IMMUTABLE PARALLEL SAFE
    AS $$ BEGIN
```

```
        RETURN 'XXXXXXXX'::text;
    END $$;

-- Now use these objects with mixed case names in
-- masking policy management procedures
CALL pgcolumnmask.create_masking_policy(
    '"Policy1"',  -- policy name should be surrounded with double quotes for quoting
    'public."Employees"', -- table and schema name should be individually
                          -- surrounded with double quotes for quoting
    JSON_OBJECT('{
        "\"Email\"", "\"MaskEmail\"(\"Email\")"
    }')::JSONB, -- masking expression should have double quotes around function names
                -- and columns names etc when needed. Also the double quotes itself
                -- should be escaped using \ (backslash) since this is a JSON string
    ARRAY['Masked_user'], -- Rolename do not need quoting
                          -- (this behaviour may change in future release)
    100
);

SELECT * FROM pgcolumnmask.pg_columnmask_policies
    WHERE tablename = 'Employees';
-[ RECORD 1 ]-----+-------------------------------------
schemaname        | public
tablename         | Employees
policyname        | Policy1
roles             | {Masked_user}
masked_columns    | {Email}
masking_functions | {"(\"MaskEmail\"(\"Email\"))::text"}
weight            | 100
```

Administrative Ansichten

Sie können die gesamte `pg_columnmask` Richtlinie in der öffentlich zugänglichen `pgcolumnmask.pg_columnmask_policies` Verwaltungsansicht überprüfen. Die folgenden Informationen sind in dieser Ansicht verfügbar. Die Ansicht gibt nur die Maskierungsrichtlinien zurück, die dem aktuellen Benutzer gehören.

Name der Spalte	Datentyp	Beschreibung
schemaname	NAME	Schema der Beziehung, an die die Richtlinie angehängt ist

Name der Spalte	Datentyp	Beschreibung
tablename	NAME	Name der Beziehung, an die die Richtlinie angehängt ist
Name der Richtlinie	NAME	Name der Maskierungsrichtlinie. Alle Maskierungsrichtlinien haben eindeutige Namen
Rollen	TEXT []	Rolle, für die die Richtlinie gilt.
masked_columns	TEXT []	Maskierte Spalten
maskierende Funktionen	TEXT []	Maskierungsfunktionen
Gewichtung	INT	Gewicht der beigefügten Richtlinie

Vordefinierte Datenmaskierungsfunktionen

`pg_columnmask`Die Erweiterung bietet integrierte Hilfsfunktionen, die in der Sprache C geschrieben sind (für eine schnellere Ausführung) und als Maskierungsausdruck für `pg_columnmask` Richtlinien verwendet werden können.

mask_text

Eine Funktion zum Maskieren von Textdaten mit konfigurierbaren Sichtbarkeitsoptionen.

Argumente

Parameter	Datatype	Description
`input`	TEXT	Die ursprüngliche Textzeichenfolge, die maskiert werden soll
`mask_char`	CHAR (1)	Für die Maskierung verwendetes Zeichen (Standard: 'X')

Parameter	Datatype	Description
`visible_prefix`	INT	Anzahl der Zeichen am Anfang des Eingabetextes, die unmaskiert bleiben (Standard: 0)
`visible_suffix`	INT	Anzahl der Zeichen am Ende des Eingabetextes, die unmaskiert bleiben (Standard: 0)
`use_hash_mask`	BOOLEAN	Bei TRUE wird anstelle von mask_char eine Hash-basierte Maskierung verwendet (Standard: FALSE)

Example der Verwendung verschiedener Maskierungsoptionen

Maskiert die gesamte Eingabezeichenfolge mit dem Standardzeichen 'X'

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World');
  mask_text
-------------
 XXXXXXXXXXX
```

Verwenden Sie das `mask_char` Argument, um Texteingaben mit einem anderen Zeichen zu maskieren

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World', '*');
  mask_text
-------------
 ***********
```

Verwenden Sie `visible_prefix` und `visible_suffix` -Parameter, um zu steuern, wie viele Zeichen am Anfang und Ende des Textes unmaskiert bleiben

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World', '*', 5, 1);
  mask_text
```

```
-------------
 Hello*****d
```

Wenn wahr, `use_hash_mask` wird die Eingabezeichenfolge mit zufälligen Zeichen maskiert, `mask_char` das Argument wird ignoriert, aber trotzdem `visible_prefix` berücksichtigt `visible_suffix`

```
postgres=> SELECT pgcolumnmask.mask_text('Hello World', '*', 2, 2, true);
  mask_text
-------------
 Hex36dOHild
```

mask_timestamp

Parameter	Datatype	Description
`ts_to_mask`	TIMESTAMP (ZEITSTEMPEL)	Der ursprüngliche Zeitstempel, der maskiert werden soll
`mask_part`	TEXT	Gibt an, welcher Teil des Zeitstempels maskiert werden soll (Standard: 'all') Gültige Werte: 'Jahr', 'Monat', 'Tag', 'Stunde', 'Minute', 'Sekunde', 'alles'
`mask_value`	TIMESTAMP (ZEITSTEMPEL)	Der Zeitstempelwert, der für die Maskierung verwendet werden soll (Standard: '1900-01-01 00:00:00 ')

Example Verwendung von **`mask_timestamps`**

Diese Beispiele veranschaulichen die vollständige Maskierung des Zeitstempels auf einen Standardwert, die teilweise Maskierung bestimmter Zeitstempelkomponenten (nur Jahr) und die Maskierung mit einem benutzerdefinierten Ersatzwert.

Maskiert den Eingabewert vollständig mit dem Standardzeitstempel

```
postgres=> SELECT pgcolumnmask.mask_timestamp('2023-06-15 14:30:00');
   mask_timestamp
---------------------
 1900-01-01 00:00:00
```

Um nur einen Teil des Zeitstempels zu maskieren, zum Beispiel nur das Jahr

```
postgres=> SELECT pgcolumnmask.mask_timestamp('2023-06-15 14:30:00', 'year');
   mask_timestamp
---------------------
 1900-06-15 14:30:00
```

Um den maskierten Wert für den Zeitstempel zu ändern, verwenden Sie das Argument `mask_value`

```
postgres=> SELECT pgcolumnmask.mask_timestamp('2023-06-15 14:30:00', 'all', '2012-12-12
 12:12:12');
   mask_timestamp
---------------------
 2012-12-12 12:12:12
```

mask_timestamp

Eine Funktion zum Maskieren von E-Mail-Adressen unter Beibehaltung der E-Mail-Struktur.

Parameter	Datatype	Description
`input`	TEXT	Die ursprüngliche E-Mail-Adresse, die maskiert werden soll
`mask_char`	CHAR (1)	Für die Maskierung verwendetes Zeichen (Standard: 'X')
`mask_local`	BOOLEAN	Bei TRUE wird der lokale Teil der E-Mail (vor @) maskiert (Standard: TRUE)
`mask_domain`	BOOLEAN	Bei TRUE wird der Domänenteil der E-Mail (nach @) maskiert (Standard: TRUE)

Example Verwendung von **`mask_email`**

Diese Beispiele veranschaulichen die vollständige E-Mail-Maskierung, benutzerdefinierte Maskenzeichen und die selektive Maskierung entweder des lokalen Teils oder des Domainteils der E-Mail-Adresse.

Vollständige Maskierung

```
postgres=> SELECT pgcolumnmask.mask_email('user@example.com');
    mask_email
------------------
 XXXX@XXXXXXX.com
```

Dient `mask_char` zum Ändern des für die Maskierung verwendeten Zeichens

```
postgres=> SELECT pgcolumnmask.mask_email('user@example.com', '*');
    mask_email
------------------
 ****@*******.com
```

Verwenden Sie `mask_local` und`mask_domain`, um die Maskierung auf lokaler Ebene und in der Domäne zu steuern

```
postgres=> SELECT pgcolumnmask.mask_email('user@example.com', '*', true, false);
    mask_email
------------------
 ****@example.com

postgres=> SELECT pgcolumnmask.mask_email('user@example.com', '*', false, true);
    mask_email
------------------
 user@*******.com
```

Implementierung von pg_columnmask in einem Workflow end-to-end

In diesem Abschnitt wird eine vollständige Implementierung der `pg_columnmask` Verwendung einer Beispieltabelle für Mitarbeiter mit sensiblen Daten demonstriert. Sie erfahren, wie Sie benutzerdefinierte Maskierungsfunktionen erstellen, mehrere Maskierungsrichtlinien mit unterschiedlichen Gewichtungsstufen für verschiedene Rollen (Praktikant, Support, Analyst) definieren und beobachten, wie Benutzer mit einer oder mehreren Rollenmitgliedschaften

unterschiedliche Ebenen maskierter Daten sehen. Die Beispiele behandeln auch das Maskierungsverhalten in DML-Anweisungen mit RETURNING-Klauseln, Trigger für Tabellen im Vergleich zu Ansichten und Policy-Management-Operationen wie Umbenennen, Ändern von Gewichtungen und Bereinigung.

1. Erstellen Sie eine Beispieltabelle mit einigen sensiblen Daten:

```
CREATE SCHEMA hr;

CREATE TABLE hr.employees (
    id INT PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT,
    ssn TEXT,
    salary NUMERIC(10,2)
 );

INSERT INTO hr.employees VALUES
    (1, 'John Doe', 'john.doe@example.com', '123-45-6789', 50000.00),
    (2, 'Jane Smith', 'jane.smith@example.com', '987-65-4321', 60000.00);
```

2. Erstellen Sie benutzerdefinierte Maskierungsfunktionen:

```
CREATE OR REPLACE FUNCTION public.mask_ssn(ssn TEXT)
    RETURNS TEXT AS $$
    BEGIN
        RETURN 'XXX-XX-' || RIGHT(ssn, 4);
    END;
    $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION public.mask_salary(salary NUMERIC, multiplier NUMERIC
 DEFAULT 0.0)
    RETURNS NUMERIC AS $$
    BEGIN
        RETURN salary * multiplier;
    END;
    $$ LANGUAGE plpgsql;
```

3. Erstellen Sie mehrere Richtlinien mit unterschiedlichen Maskierungsstufen, die auf den Benutzerrollen basieren:

```
-- Create different roles
```

```
CREATE ROLE analyst_role;
CREATE ROLE support_role;
CREATE ROLE intern_role;

GRANT USAGE ON SCHEMA hr TO analyst_role, support_role, intern_role;
GRANT SELECT ON hr.employees TO analyst_role, support_role, intern_role;
-------------------------------------------------------------------------

-- Low-Weight Policy (Intern)
CALL pgcolumnmask.create_masking_policy(
    'employee_mask_strict',
    'hr.employees',
    JSON_BUILD_OBJECT('name', 'pgcolumnmask.mask_text(name, ''*'')',
                      'email', 'pgcolumnmask.mask_email(email)',
                      'ssn', 'pgcolumnmask.mask_text(ssn, ''*'')',
                      'salary', 'public.mask_salary(salary)')::JSONB,
    ARRAY['intern_role'],
    10  -- Lowest weight
);

-------------------------------------------------------------------------
-- Medium-Weight Policy (Support)
CALL pgcolumnmask.create_masking_policy(
    'employee_mask_moderate',
    'hr.employees',
    JSON_BUILD_OBJECT('email', 'pgcolumnmask.mask_email(email, ''#'')',
                      'ssn', 'public.mask_ssn(ssn)',
                      'salary', 'public.mask_salary(salary)')::JSONB,
    ARRAY['support_role'],
    50   -- Medium weight
);

-------------------------------------------------------------------------
-- High-Weight Policy (Analyst)
CALL pgcolumnmask.create_masking_policy(
    'employee_mask_light',
    'hr.employees',
    JSON_BUILD_OBJECT('ssn', 'public.mask_ssn(ssn)',
                      'salary', 'public.mask_salary(salary, 0.9)')::JSONB,
    ARRAY['analyst_role'],
    100   -- Highest weight
);
```

4. Die folgenden Beispiele zeigen, wie verschiedene Benutzer Daten auf der Grundlage ihrer Rollenzugehörigkeit und der Gewichtung ihrer Richtlinien sehen.

```
-- Create users
CREATE USER sarah_intern;
GRANT intern_role TO sarah_intern;

CREATE USER lisa_support;
GRANT support_role TO lisa_support;

CREATE USER mike_analyst;
GRANT analyst_role TO mike_analyst;

CREATE USER ethan_support_intern;
GRANT support_role, intern_role TO ethan_support_intern;

CREATE USER john_analyst_intern;
GRANT analyst_role, intern_role TO john_analyst_intern;
```

Als Praktikant (strengste Maskierung):

```
SET ROLE sarah_intern;

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     | salary
----+------------+------------------------+-------------+--------
  1 | ********   | XXXXXXXX@XXXXXXX.com   | ********** |   0.00
  2 | ********** | XXXXXXXXXX@XXXXXXX.com | ********** |   0.00
```

Als Support-Benutzer (moderate Maskierung):

```
SET ROLE lisa_support;

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     | salary
----+------------+------------------------+-------------+--------
  1 | John Doe   | ########@#######.com   | XXX-XX-6789 |   0.00
  2 | Jane Smith | ##########@#######.com | XXX-XX-4321 |   0.00
```

Als Analyst (leichteste Maskierung):

```
SET ROLE mike_analyst;

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
  1 | John Doe   | john.doe@example.com   | XXX-XX-6789 | 45000.00
  2 | Jane Smith | jane.smith@example.com | XXX-XX-4321 | 54000.00
```

Als ethan_support_intern-Benutzer, der sowohl Praktikant als auch Support-Benutzer ist:

```
SET ROLE ethan_support_intern;

-- masking policies appliable to this user: employee_mask_strict and
 employee_mask_moderate
-- id : unmasked because no masking policy appliable on ethan_support_intern
--            masks these columns
-- name : masked because of employee_mask_strict policy
-- email, ssn, salary : both employee_mask_strict and employee_mask_moderate mask
 these columns
--                      but employee_mask_moderate will be use because of higher
 weight

SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     | salary
----+------------+------------------------+-------------+--------
  1 | ********   | ########@#######.com   | XXX-XX-6789 |   0.00
  2 | ********** | ##########@#######.com | XXX-XX-4321 |   0.00
```

Als john_analyst_intern, der sowohl Praktikant als auch Analyst ist:

```
SET ROLE john_analyst_intern;

-- masking policies appliable to this user: employee_mask_strict and
 employee_mask_light
-- id : unmasked because no masking policy appliable on john_analyst_intern
--            masks these columns
-- name, email : masked because of employee_mask_strict
-- ssn, salary : both employee_mask_strict and employee_mask_light mask these
 columns
--               but employee_mask_light will be use because of higher weight
```

```
SELECT * FROM hr.employees;
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
  1 | ********   | XXXXXXXX@XXXXXXX.com   | XXX-XX-6789 | 45000.00
  2 | ********** | XXXXXXXXXX@XXXXXXX.com | XXX-XX-4321 | 54000.00
```

Grundlegendes zum Maskierungsverhalten bei DML-Vorgängen

`pg_columnmask`gilt konsistent für alle DML-Operationen, einschließlich INSERT-, UPDATE-, DELETE- und MERGE-Anweisungen. Wenn Sie diese Operationen ausführen, maskiert Aurora PostgreSQL Daten nach einem Kernprinzip: Alle aus dem Speicher gelesenen Daten werden gemäß den geltenden Richtlinien des aktuellen Benutzers maskiert.

Die Maskierung wirkt sich auf einige der folgenden Abfragekomponenten aus, z. B.:

- WHERE-Klauseln
- JOIN-Bedingungen
- Unterabfragen
- RETURNING-Klausen

Alle diese Komponenten arbeiten mit maskierten Werten, nicht mit den Originaldaten. Daten werden zwar unmaskiert in den Speicher geschrieben, Benutzer sehen jedoch nur ihre maskierte Ansicht, wenn sie sie zurücklesen.

Aurora PostgreSQL erzwingt alle Datenbankeinschränkungen (NOT NULL, UNIQUE, CHECK, FOREIGN KEY) für die tatsächlich gespeicherten Werte, nicht für maskierte Werte. Dies kann gelegentlich zu offensichtlichen Inkonsistenzen führen, wenn die Maskierungsfunktionen nicht sorgfältig entworfen wurden.

Die Maskierung funktioniert zusammen mit Berechtigungen auf Spaltenebene:

- Benutzer ohne SELECT-Rechte können keine Spalten lesen
- Benutzern mit SELECT-Rechten werden maskierte Werte gemäß ihren geltenden Richtlinien angezeigt

Verstehen des Maskierungsverhaltens in Triggerfunktionen

Wenn `pg_columnmask` Richtlinien auf Tabellen angewendet werden, ist es wichtig zu verstehen, wie Maskierung mit Triggerfunktionen interagiert. Trigger sind Datenbankfunktionen, die automatisch als Reaktion auf bestimmte Ereignisse in einer Tabelle ausgeführt werden, z. B. INSERT-, UPDATE- oder DELETE-Operationen.

Standardmäßig wendet DDM je nach Triggertyp unterschiedliche Maskierungsregeln an:

Tabellentrigger

Übergangstabellen sind unmaskiert — Triggerfunktionen für Tabellen haben Zugriff auf unmaskierte Daten in ihren Übergangstabellen sowohl für alte als auch für neue Zeilenversionen

Tabellenbesitzer erstellen Trigger und sind Eigentümer der Daten, sodass sie vollen Zugriff haben, um ihre Tabellen effektiv zu verwalten

Trigger anzeigen (STATT Trigger)

Übergangstabellen sind maskiert — Triggerfunktionen in Ansichten sehen maskierte Daten entsprechend den Berechtigungen des aktuellen Benutzers

Eigentümer von Ansichten können sich von Besitzern von Basistabellen unterscheiden und sollten die Maskierungsrichtlinien für die zugrunde liegenden Tabellen beachten

Zwei Konfigurationsparameter auf Serverebene steuern das Triggerverhalten bei maskierten Tabellen. Diese können nur gesetzt werden durch: `rds_superuser`

- Trigger für maskierte Tabellen einschränken — Verhindert die Ausführung von Triggern, wenn ein maskierter Benutzer DML-Operationen an Tabellen mit entsprechenden Maskierungsrichtlinien ausführt.
- Trigger für Ansichten mit maskierten Tabellen einschränken: — Verhindert die Ausführung von Triggern für Ansichten, wenn die Ansichtsdefinition Tabellen mit Maskierungsrichtlinien enthält, die für den aktuellen Benutzer gelten.

Example der Unterschiede zwischen der Funktionsanwendung auf Tabelle und Ansicht

Im folgenden Beispiel wird eine Triggerfunktion erstellt, die alte und neue Zeilenwerte ausgibt. Anschließend wird veranschaulicht, wie sich dieselbe Funktion, wenn sie an eine Tabelle angehängt wird, anders verhält als bei einer Ansicht.

```
-- Create trigger function
CREATE OR REPLACE FUNCTION print_changes()
    RETURNS TRIGGER AS
    $$
        BEGIN
        RAISE NOTICE 'Old row: name=%, email=%, ssn=%, salary=%',
            OLD.name, OLD.email, OLD.ssn, OLD.salary;

        RAISE NOTICE 'New row: name=%, email=%, ssn=%, salary=%',
            NEW.name, NEW.email, NEW.ssn, NEW.salary;

        RETURN NEW;
        END;
    $$ LANGUAGE plpgsql;

-- Create trigger
CREATE TRIGGER print_changes_trigger
    BEFORE UPDATE ON hr.employees
    FOR EACH ROW
    EXECUTE FUNCTION print_changes();

-- Grant update to analyst role
GRANT UPDATE ON hr.employees TO analyst_role;

-- Unmasked data must be seen inside trigger even for masked user for the OLD and NEW
-- row passed to trigger function
BEGIN;
SET ROLE mike_analyst;
UPDATE hr.employees SET id = id + 10 RETURNING *;
NOTICE:  Old row: name=John Doe, email=john.doe@example.com, ssn=123-45-6789,
 salary=50000.00
NOTICE:  New row: name=John Doe, email=john.doe@example.com, ssn=123-45-6789,
 salary=50000.00
NOTICE:  Old row: name=Jane Smith, email=jane.smith@example.com, ssn=987-65-4321,
 salary=60000.00
NOTICE:  New row: name=Jane Smith, email=jane.smith@example.com, ssn=987-65-4321,
 salary=60000.00
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
 11 | John Doe   | john.doe@example.com   | XXX-XX-6789 | 45000.00
 12 | Jane Smith | jane.smith@example.com | XXX-XX-4321 | 54000.00
(2 rows)
```

```
ROLLBACK;


-- Triggers on views (which are supposed to see masked data for new/old row)
CREATE VIEW hr.view_over_employees AS SELECT * FROM hr.employees;
GRANT UPDATE, SELECT ON hr.view_over_employees TO analyst_role;

-- Create trigger for this view
CREATE TRIGGER print_changes_trigger
    INSTEAD OF UPDATE ON hr.view_over_employees
    FOR EACH ROW
    EXECUTE FUNCTION print_changes();

-- Masked new and old rows should be passed to trigger if trigger is on view
BEGIN;
SET ROLE mike_analyst;
UPDATE hr.view_over_employees SET id = id + 10 RETURNING *;
NOTICE:  Old row: name=John Doe, email=john.doe@example.com, ssn=XXX-XX-6789,
 salary=45000.00
NOTICE:  New row: name=John Doe, email=john.doe@example.com, ssn=XXX-XX-6789,
 salary=45000.00
NOTICE:  Old row: name=Jane Smith, email=jane.smith@example.com, ssn=XXX-XX-4321,
 salary=54000.00
NOTICE:  New row: name=Jane Smith, email=jane.smith@example.com, ssn=XXX-XX-4321,
 salary=54000.00
 id |    name    |         email          |     ssn     |  salary
----+------------+------------------------+-------------+----------
 11 | John Doe   | john.doe@example.com   | XXX-XX-6789 | 45000.00
 12 | Jane Smith | jane.smith@example.com | XXX-XX-4321 | 54000.00
(2 rows)
ROLLBACK;
```

Wir empfehlen, das Verhalten des Triggers zu überprüfen, bevor Sie Trigger in maskierten Tabellen implementieren. Tabellen-Trigger haben Zugriff auf unmaskierte Daten in Übergangstabellen, während View-Trigger maskierte Daten sehen.

Example der Maskierungsrichtlinie beim Umbenennen

Das folgende Beispiel zeigt, wie vorhandene Richtlinien mithilfe des `rename_masking_policy` Verfahrens umbenannt werden.

```
-- Rename the strict policy
CALL pgcolumnmask.rename_masking_policy(
```

```
    'employee_mask_strict',
    'hr.employees',
    'intern_protection_policy'
);

-- Verify the rename
SELECT policyname, roles, weight
    FROM pgcolumnmask.pg_columnmask_policies
    WHERE tablename = 'employees'
    ORDER BY weight DESC;

        policyname        |     roles      | weight
--------------------------+----------------+--------
 employee_mask_light      | {analyst_role} |    100
 employee_mask_moderate   | {support_role} |     50
 intern_protection_policy | {intern_role}  |     10
```

Example der Änderung des politischen Gewichts

Das folgende Beispiel zeigt, wie das Gewicht von Policen geändert werden kann, um ihr Gewicht zu ändern.

```
-- Change weight of moderate policy
CALL pgcolumnmask.alter_masking_policy(
    'employee_mask_moderate'::NAME,
    'hr.employees'::REGCLASS,
    NULL,    -- Keep existing masking expressions
    NULL,    -- Keep existing roles
    75       -- New weight
);

-- Verify the changes
SELECT policyname, roles, weight
    FROM pgcolumnmask.pg_columnmask_policies
    WHERE tablename = 'employees'
    ORDER BY weight DESC;
        policyname        |     roles      | weight
--------------------------+----------------+--------
 employee_mask_light      | {analyst_role} |    100
 employee_mask_moderate   | {support_role} |     75
 intern_protection_policy | {intern_role}  |     10
```

Example des Aufräumens

Das folgende Beispiel zeigt, wie alle Richtlinien, Tabellen und Benutzer gelöscht werden.

```
-- Drop policies
CALL pgcolumnmask.drop_masking_policy(
    'intern_protection_policy',
    'hr.employees'
);

CALL pgcolumnmask.drop_masking_policy(
    'employee_mask_moderate',
    'hr.employees'
);

CALL pgcolumnmask.drop_masking_policy(
    'employee_mask_light',
    'hr.employees'
);

-- Drop table and functions
DROP VIEW IF EXISTS hr.view_over_employees;
DROP TABLE IF EXISTS hr.employees;
DROP SCHEMA IF EXISTS hr;
DROP FUNCTION IF EXISTS public.mask_ssn(text);
DROP FUNCTION IF EXISTS public.mask_salary(numeric, numeric);

-- Drop users
DROP USER sarah_intern, lisa_support, mike_analyst,
    ethan_support_intern, john_analyst_intern;
DROP ROLE intern_role, support_role, analyst_role;
```

Verwaltungsrolle für Maskierungsrichtlinien konfigurieren

Die PostgreSQL-Erweiterung zur Spaltenmaskierung ermöglicht es Ihnen`pg_columnmask`, die Verwaltung von Maskierungsrichtlinien an eine bestimmte Rolle zu delegieren, anstatt Tabellenbesitzerrechte zu verlangen`rds_superuser`. Dies ermöglicht eine genauere Kontrolle darüber, wer Maskierungsrichtlinien erstellen, ändern und löschen kann.

Gehen Sie wie folgt vor, um die Rolle zu konfigurieren, der Verwaltungsrechte für Maskierungsrichtlinien zugewiesen werden sollen:

1. Erstellen Sie die Rolle des Richtlinienadministrators — Erstellen Sie als Nächstes eine neue Rolle`rds_superuser`, die für die Verwaltung von Maskierungsrichtlinien zuständig ist:

   ```
   CREATE ROLE mask_admin NOLOGIN;
   ```

2. PostgreSQL-Parameter konfigurieren — Legen Sie in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe den `pgcolumnmask.policy_admin_rolname` Engine-Konfigurationsparameter auf den Namen der Rolle fest, die Sie erstellt haben:

   ```
   pgcolumnmask.policy_admin_rolname = mask_admin
   ```

   Diese Engine-Konfigurationsparameter können in einer DB-Cluster-Parametergruppe festgelegt werden und erfordern keinen Instance-Neustart. Einzelheiten zur Aktualisierung des Parameters finden Sie unterÄndern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

3. Weisen Sie Benutzern die Rolle zu. Weisen Sie die `mask_admin` Rolle den Benutzern zu`rds_superuser`, die in der Lage sein sollten, Maskierungsrichtlinien zu verwalten:

   ```
   CREATE USER alice LOGIN;
   CREATE USER bob LOGIN;
   GRANT mask_admin TO alice, bob;
   ```

   Stellen Sie außerdem sicher, dass die Benutzer über die USAGE-Rechte für die Schemas verfügen, in denen sie Maskierungsrichtlinien verwalten werden:

   ```
   GRANT USAGE ON SCHEMA hr TO alice, bob;
   ```

Wenn Benutzer `alice` nun bob eine Verbindung zur Datenbank herstellen, können sie die `pg_columnmask` Standarderweiterungsfunktionen verwenden, um Maskierungsrichtlinien für alle Tabellen in allen Schemas zu erstellen, zu ändern und zu löschen, für die sie USAGE Berechtigungen für das Schema haben.

Bewährte Methoden für die sichere Implementierung von pg_columnmask

Der folgende Abschnitt enthält bewährte Sicherheitsmethoden für die Implementierung `pg_columnmask` in Ihrer Aurora PostgreSQL-Umgebung. Folgen Sie diesen Empfehlungen, um:

- Richten Sie eine sichere, rollenbasierte Architektur für die Zugriffskontrolle ein

- Entwickeln Sie Maskierungsfunktionen, die Sicherheitslücken verhindern
- Verstehen und kontrollieren Sie das Triggerverhalten mit maskierten Daten

Rollenbasierte Sicherheitsarchitektur

Definieren Sie eine Rollenhierarchie, um Zugriffskontrollen in Ihrer Datenbank zu implementieren. Aurora PostgreSQL `pg_columnmask` erweitert diese Kontrollen, indem es eine zusätzliche Ebene für die feinkörnige Datenmaskierung innerhalb dieser Rollen bereitstellt.

Erstellen Sie spezielle Rollen, die auf die Unternehmensfunktionen abgestimmt sind, anstatt einzelnen Benutzern Berechtigungen zu gewähren. Dieser Ansatz bietet eine bessere Überprüfbarkeit und vereinfacht das Berechtigungsmanagement, wenn sich Ihre Organisationsstruktur weiterentwickelt.

Example der Schaffung einer organisatorischen Rollenhierarchie

Im folgenden Beispiel wird eine organisatorische Rollenhierarchie mit speziellen Rollen für verschiedene Funktionen erstellt und anschließend einzelnen Benutzern die entsprechenden Rollen zugewiesen. In diesem Beispiel werden zuerst Organisationsrollen (analyst_role, support_role) erstellt, dann wird einzelnen Benutzern die Mitgliedschaft in diesen Rollen gewährt. Diese Struktur ermöglicht es Ihnen, Berechtigungen auf Rollenebene und nicht für jeden einzelnen Benutzer zu verwalten.

```
-- Create organizational role hierarchy
CREATE ROLE data_admin_role;
CREATE ROLE security_admin_role;
CREATE ROLE analyst_role;
CREATE ROLE support_role;
CREATE ROLE developer_role;

-- Specify security_admin_role as masking policy manager in the DB cluster parameter
-- group pgcolumnmask.policy_admin_rolname = 'security_admin_role'

-- Create specific users and assign to appropriate roles
CREATE USER security_manager;
CREATE USER data_analyst1, data_analyst2;
CREATE USER support_agent1, support_agent2;

GRANT security_admin_role TO security_manager;
GRANT analyst_role TO data_analyst1, data_analyst2;
```

```
GRANT support_role TO support_agent1, support_agent2;
```

Implementieren Sie das Prinzip der geringsten Rechte, indem Sie nur die für jede Rolle erforderlichen Mindestberechtigungen gewähren. Vermeiden Sie die Gewährung umfassender Berechtigungen, die ausgenutzt werden könnten, wenn Anmeldeinformationen kompromittiert werden.

```
-- Grant specific table permissions rather than schema-wide access
GRANT SELECT ON sensitive_data.customers TO analyst_role;
GRANT SELECT ON sensitive_data.transactions TO analyst_role;
-- Do not grant: GRANT ALL ON SCHEMA sensitive_data TO analyst_role;
```

Richtlinienadministratoren benötigen USAGE Rechte für Schemas, in denen sie Maskierungsrichtlinien verwalten. Gewähren Sie diese Rechte selektiv und folgen Sie dabei dem Prinzip der geringsten Rechte. Führen Sie regelmäßige Überprüfungen der Schemazugriffsberechtigungen durch, um sicherzustellen, dass nur autorisiertes Personal über die Funktionen zur Richtlinienverwaltung verfügt.

Die Konfiguration der Rollenparameter des Policy-Administrators ist nur auf Datenbankadministratoren beschränkt. Dieser Parameter kann nicht auf Datenbank- oder Sitzungsebene geändert werden, wodurch verhindert wird, dass Benutzer ohne Benutzerrechte die Zuweisungen von Richtlinienadministratoren außer Kraft setzen. Diese Einschränkung stellt sicher, dass die Kontrolle der Maskierungsrichtlinien zentralisiert und sicher bleibt.

Weisen Sie die Rolle des Richtlinienadministrators bestimmten Personen und nicht Gruppen zu. Dieser gezielte Ansatz gewährleistet einen selektiven Zugriff auf die Verwaltung von Maskierungsrichtlinien, da Richtlinienadministratoren alle Tabellen in der Datenbank maskieren können.

Entwicklung sicherer Maskierungsfunktionen

Entwickeln Sie Maskierungsfunktionen unter Verwendung von Semantik für frühe Bindungen, um eine korrekte Nachverfolgung von Abhängigkeiten sicherzustellen und Sicherheitslücken bei späten Bindungen zu vermeiden, z. B. die Änderung von Suchpfaden während der Laufzeit. Es wird empfohlen, die `BEGIN ATOMIC` Syntax für SQL-Funktionen zu verwenden, um die Validierung während der Kompilierung (d. h. frühes Binden) und das Abhängigkeitsmanagement zu ermöglichen.

```
-- Example - Secure masking function with early binding
CREATE OR REPLACE FUNCTION secure_mask_ssn(input_ssn TEXT)
    RETURNS TEXT
    LANGUAGE SQL
```

```
    IMMUTABLE PARALLEL SAFE STRICT
    BEGIN ATOMIC
        SELECT CASE
            WHEN input_ssn IS NULL THEN NULL
            WHEN length(input_ssn) < 4 THEN repeat('X', length(input_ssn))
            ELSE repeat('X', length(input_ssn) - 4) || right(input_ssn, 4)
        END;
    END;
```

Alternativ können Sie Funktionen erstellen, die immun gegen Änderungen des Suchpfads sind, indem Sie alle Objektreferenzen explizit per Schema qualifizieren und so ein konsistentes Verhalten über verschiedene Benutzersitzungen hinweg sicherstellen.

```
-- Function immune to search path changes
CREATE OR REPLACE FUNCTION data_masking.secure_phone_mask(phone_number TEXT)
    RETURNS TEXT
    LANGUAGE SQL
    IMMUTABLE PARALLEL SAFE STRICT
    AS $$
    SELECT CASE
        WHEN phone_number IS NULL THEN NULL
        WHEN public.length(public.regexp_replace(phone_number, '[^0-9]', '', 'g')) < 10
 THEN 'XXX-XXX-XXXX'
        ELSE public.regexp_replace(
            phone_number,
            '([0-9]{3})[0-9]{3}([0-9]{4})',
            public.concat('\1-XXX-\2')
        )
    END;
    $$;
```

Implementieren Sie die Eingabevalidierung innerhalb von Maskierungsfunktionen, um Sonderfälle zu behandeln und unerwartetes Verhalten zu verhindern. Schließen Sie immer die NULL-Behandlung ein und validieren Sie die Eingabeformate, um ein konsistentes Maskierungsverhalten sicherzustellen.

```
-- Robust masking function with comprehensive input validation
CREATE OR REPLACE FUNCTION secure_mask_phone(phone_number TEXT)
    RETURNS TEXT
    LANGUAGE SQL
    IMMUTABLE PARALLEL SAFE STRICT
    BEGIN ATOMIC
```

```
        SELECT CASE
            WHEN phone_number IS NULL THEN NULL
            WHEN length(trim(phone_number)) = 0 THEN phone_number
            WHEN length(regexp_replace(phone_number, '[^0-9]', '', 'g')) < 10 THEN
 'XXX-XXX-XXXX'
            ELSE regexp_replace(phone_number, '([0-9]{3})[0-9]{3}([0-9]{4})', '\1-XXX-
\2')
        END;
    END;
```

DML löst Verhalten mit pg_columnmask aus

Bei Tabellentriggern werden Übergangstabellen vollständig demaskiert. Bei View-Triggern (IOT) werden Übergangstabellen entsprechend den Anzeigeberechtigungen des aktuellen Benutzers maskiert.

Tabellentrigger werden mit pg_columnmask ausgelöst

Triggern wird eine Übergangstabelle übergeben, die die alte und die neue Version der Zeilen enthält, die durch die auslösende DML-Abfrage geändert wurden. Je nachdem, wann der Trigger ausgelöst wird, füllt Aurora PostgreSQL die alten und neuen Zeilen auf. Ein `BEFORE INSERT` Trigger hat beispielsweise nur neue Versionen der Zeilen und leere alte Versionen, weil es keine alte Version gibt, auf die verwiesen werden kann.

`pg_columnmask`maskiert keine Übergangstabellen innerhalb von Triggern für Tabellen. Trigger können maskierte Spalten in ihrem Hauptteil verwenden, und es werden unmaskierte Daten angezeigt. Der Ersteller des Triggers sollte sicherstellen, wie der Trigger für einen Benutzer ausgeführt wird. Das folgende Beispiel funktioniert in diesem Fall korrekt.

```
-- Example for table trigger uses masked column in its definition
-- Create a table and insert some rows
CREATE TABLE public.credit_card_table (
    name TEXT,
    credit_card_no VARCHAR(16),
    is_fraud BOOL
);

INSERT INTO public.credit_card_table (name, credit_card_no, is_fraud)
    VALUES
    ('John Doe', '4532015112830366', false),
    ('Jane Smith', '5410000000000000', true),
    ('Brad Smith', '1234567891234567', true);
```

```
-- Create a role which will see masked data and grant it privileges
CREATE ROLE intern_user;
GRANT SELECT, DELETE ON public.credit_card_table TO intern_user;

-- Trigger which will silenty skip delete of non fraudelent credit cards
CREATE OR REPLACE FUNCTION prevent_non_fraud_delete()
    RETURNS TRIGGER AS
    $$
    BEGIN
        IF OLD.is_fraud = false THEN
            RETURN NULL;
        END IF;
        RETURN OLD;
    END;
    $$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_non_fraud_delete
    BEFORE DELETE ON credit_card_table
    FOR EACH ROW
    EXECUTE FUNCTION prevent_non_fraud_delete();

CREATE OR REPLACE FUNCTION public.return_false()
    RETURNS BOOLEAN
    LANGUAGE SQL
    IMMUTABLE PARALLEL SAFE STRICT
    BEGIN ATOMIC
      SELECT false;
    END;

-- A masking policy that masks both credit card number and is_fraud column.
-- If we apply masking inside trigger then prevent_non_fraud_delete trigger will
-- allow deleting more rows to masked user (even non fraud ones).
CALL pgcolumnmask.create_masking_policy(
    'mask_credit_card_no_&_is_fraud'::NAME,
    'public.credit_card_table'::REGCLASS,
    JSON_BUILD_OBJECT('credit_card_no', 'pgcolumnmask.mask_text(credit_card_no)',
                      'is_fraud', 'public.return_false()')::JSONB,
    ARRAY['intern_user']::NAME[],
    10::INT
);

-- Test trigger behaviour using intern_user
BEGIN;
```

```
SET ROLE intern_user;
-- credit card number & is_fraud is completely masked from intern_user
SELECT * FROM public.credit_card_table;
    name    |  credit_card_no  | is_fraud
------------+------------------+----------
 John Doe   | XXXXXXXXXXXXXXXX | f
 Jane Smith | XXXXXXXXXXXXXXXX | f
 Brad Smith | XXXXXXXXXXXXXXXX | f
(3 rows)

-- The delete trigger lets the intern user delete rows for Jane and Brad even though
-- intern_user sees their is_fraud = false, but the table trigger works with
 original
-- unmasked value
DELETE FROM public.credit_card_table RETURNING *;
    name    |  credit_card_no  | is_fraud
------------+------------------+----------
 Jane Smith | XXXXXXXXXXXXXXXX | f
 Brad Smith | XXXXXXXXXXXXXXXX | f
(2 rows)

COMMIT;
```

Der Trigger-Ersteller gibt unmaskierte Daten an den Benutzer weiter, wenn er nicht vorsichtig mit den Anweisungen umgeht, die er in seinem Triggertext verwendet. Wenn Sie beispielsweise a verwenden, wird `RAISE NOTICE '%', masked_column;` die Spalte an den aktuellen Benutzer ausgegeben.

```
-- Example showing table trigger leaking column value to current user
CREATE OR REPLACE FUNCTION leaky_trigger_func()
    RETURNS TRIGGER AS
    $$
    BEGIN
        RAISE NOTICE 'Old credit card number was: %', OLD.credit_card_no;
        RAISE NOTICE 'New credit card number is %', NEW.credit_card_no;
        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

CREATE TRIGGER leaky_trigger
    AFTER UPDATE ON public.credit_card_table
    FOR EACH ROW
```

```
    EXECUTE FUNCTION leaky_trigger_func();

-- Grant update on column is_fraud to auditor role
-- auditor will NOT HAVE PERMISSION TO READ DATA
CREATE ROLE auditor;
GRANT UPDATE (is_fraud) ON public.credit_card_table TO auditor;

-- Also add auditor role to existing masking policy on credit card table
CALL pgcolumnmask.alter_masking_policy(
    'mask_credit_card_no_&_is_fraud'::NAME,
    'public.credit_card_table'::REGCLASS,
    NULL::JSONB,
    ARRAY['intern_user', 'auditor']::NAME[],
    NULL::INT
);

-- Log in as auditor
-- [auditor]
-- Update will fail if trying to read data from the table
UPDATE public.credit_card_table
    SET is_fraud = true
    WHERE credit_card_no = '4532015112830366';
ERROR:  permission denied for table cc_table

-- [auditor]
-- But leaky update trigger will still print the entire row even though
-- current user does not have permission to select from public.credit_card_table
UPDATE public.credit_card_table SET is_fraud = true;
NOTICE:  Old credit_card_no was: 4532015112830366
NOTICE:  New credit_card_no is 4532015112830366
```

Wird bei Ansichten mit pg_columnmask ausgelöst (anstelle von Triggern)

Trigger können nur für Ansichten in PostgreSQL erstellt werden. Sie werden für die Ausführung von DML-Anweisungen in Ansichten verwendet, die nicht aktualisierbar sind. Transittabellen werden immer intern maskiert und nicht innerhalb von Triggern (IOT), da die in der View-Abfrage verwendete View und die Basistabellen unterschiedliche Besitzer haben können. In diesem Fall können für Basistabellen einige Maskierungsrichtlinien gelten, die für den Eigentümer der Ansicht gelten, und der Eigentümer der Ansicht muss in seinen Triggern immer maskierte Daten aus Basistabellen sehen. Dies unterscheidet sich von Triggern für Tabellen, da in diesem Fall der Trigger-Ersteller und die Daten in den Tabellen demselben Benutzer gehören, was hier nicht der Fall ist.

```
-- Create a view over credit card table
CREATE OR REPLACE VIEW public.credit_card_view
    AS
    SELECT * FROM public.credit_card_table;

-- Truncate credit card table and insert fresh data
TRUNCATE TABLE public.credit_card_table;
INSERT INTO public.credit_card_table (name, credit_card_no, is_fraud)
    VALUES
    ('John Doe', '4532015112830366', false),
    ('Jane Smith', '5410000000000000', true),
    ('Brad Smith', '1234567891234567', true);

CREATE OR REPLACE FUNCTION public.print_changes()
    RETURNS TRIGGER AS
    $$
    BEGIN
        RAISE NOTICE 'Old row: name=%, credit card number=%, is fraud=%',
            OLD.name, OLD.credit_card_no, OLD.is_fraud;

        RAISE NOTICE 'New row: name=%, credit card number=%, is fraud=%',
            NEW.name, NEW.credit_card_no, NEW.is_fraud;

    RETURN NEW;
   END;
   $$ LANGUAGE plpgsql;

CREATE TRIGGER print_changes_trigger
    INSTEAD OF UPDATE ON public.credit_card_view
    FOR EACH ROW
    EXECUTE FUNCTION public.print_changes();

GRANT SELECT, UPDATE ON public.credit_card_view TO auditor;

-- [auditor]
-- Login as auditor role
BEGIN;

-- Any data coming out from the table will be masked in instead of triggers
-- according to masking policies applicable to current user
UPDATE public.credit_card_view
    SET name = CONCAT(name, '_new_name')
    RETURNING *;
```

```
NOTICE:  Old row: name=John Doe, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=John Doe_new_name, credit card number=XXXXXXXXXXXXXXXX, is
 fraud=f
NOTICE:  Old row: name=Jane Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Jane Smith_new_name, credit card number=XXXXXXXXXXXXXXXX, is
 fraud=f
NOTICE:  Old row: name=Brad Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Brad Smith_new_name, credit card number=XXXXXXXXXXXXXXXX, is
 fraud=f
        name         |  credit_card_no  | is_fraud
---------------------+------------------+----------
 John Doe_new_name   | XXXXXXXXXXXXXXXX | f
 Jane Smith_new_name | XXXXXXXXXXXXXXXX | f
 Brad Smith_new_name | XXXXXXXXXXXXXXXX | f

 -- Any new data going into the table using INSERT or UPDATE command will be
 unmasked
 UPDATE public.credit_card_view
    SET credit_card_no = '9876987698769876'
    RETURNING *;
NOTICE:  Old row: name=John Doe, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=John Doe, credit card number=9876987698769876, is fraud=f
NOTICE:  Old row: name=Jane Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Jane Smith, credit card number=9876987698769876, is fraud=f
NOTICE:  Old row: name=Brad Smith, credit card number=XXXXXXXXXXXXXXXX, is fraud=f
NOTICE:  New row: name=Brad Smith, credit card number=9876987698769876, is fraud=f
    name    |  credit_card_no  | is_fraud
------------+------------------+----------
 John Doe   | 9876987698769876 | f
 Jane Smith | 9876987698769876 | f
 Brad Smith | 9876987698769876 | f

 COMMIT;
```

Datenbank-/Benutzerebene zur Steuerung des GuCs Triggerverhaltens

Zwei Konfigurationsparameter steuern das Trigger-Ausführungsverhalten für Benutzer mit entsprechenden Maskierungsrichtlinien. Verwenden Sie diese Parameter, um zu verhindern, dass Trigger in maskierten Tabellen oder Ansichten ausgeführt werden, wenn zusätzliche Sicherheitseinschränkungen erforderlich sind. Beide Parameter sind standardmäßig deaktiviert, sodass Trigger normal ausgeführt werden können.

Erster GUC: Beschränkung der Triggerauslösung für maskierte Tabellen

Spezifikationen:

- Name: `pgcolumnmask.restrict_dml_triggers_for_masked_users`
- Typ: `boolean`
- Standard: `false` (Trigger dürfen ausgeführt werden)

Wenn dieser Wert auf TRUE gesetzt ist, wird die Ausführung von Triggern in maskierten Tabellen für maskierte Benutzer verhindert. `pg_columnmask`durchläuft den Fehler.

Zweiter GUC: Einschränkung beim Auslösen von Triggern für Ansichten mit maskierten Tabellen

Spezifikationen:

- Name: `pgcolumnmask.restrict_iot_triggers_for_masked_users`
- Typ: `boolean`
- Standard: `false` (Trigger dürfen ausgeführt werden)

Wenn der Wert auf TRUE gesetzt ist, wird die Ausführung von Triggern in Ansichten verhindert, deren Definition maskierte Tabellen für maskierte Benutzer enthält.

Diese Parameter funktionieren unabhängig voneinander und sind wie standardmäßige Datenbankkonfigurationsparameter konfigurierbar.

Aurora PostgreSQL pg_columnmask Datenverschiebungsszenarien

`pg_columnmask`Das Verhalten variiert bei verschiedenen Datenverschiebungsvorgängen, je nachdem, ob der Vorgang auf der Speicher-, logischen oder Anwendungsebene stattfindet. Operationen auf Speicherebene (wie das Klonen) verhalten sich anders als logische Operationen (z. B.`pg_dump`) und Operationen auf Anwendungsebene (wie FDW-Abfragen). In diesem Abschnitt wird das Maskierungsverhalten für gängige Szenarien wie Replikation, Backups, Exporte und Migrationen beschrieben und die jeweiligen Sicherheitsauswirkungen erläutert.

Themen

- Aurora Global Database und Read Replicas
- Datenbankklon und Snapshot-Wiederherstellung
- Logische Replikation
- Blau/Grün-Bereitstellungen
- Zero-ETL- und CDC-Streams

- AWS Database Migration Service
- Datenexporte
- Ansichten und materialisierte Ansichten
- Daten speichern und wiederherstellen
- Wrapper für ausländische Daten

Aurora Global Database und Read Replicas

`pg_columnmask`Aurora-Richtlinien werden in Datenbanksystemtabellen innerhalb des Cluster-Volumes gespeichert. Alle Replikate greifen auf dieselben Richtlinien zu und geben konsistent maskierte Ergebnisse zurück. Bei Bereitstellungen von Aurora Global Database werden `pg_columnmask` Richtlinien AWS-Regionen zusammen mit anderen Datenbanksystemtabellen auf sekundäre Daten repliziert, wodurch ein konsistenter Datenschutz in allen Regionen gewährleistet wird. In Failover-Szenarien bleiben alle `pg_columnmask` Richtlinien intakt und funktionsfähig.

Datenbankklon und Snapshot-Wiederherstellung

Bei Aurora Fast Clone- und Snapshot-Wiederherstellungsvorgängen werden alle `pg_columnmask` Richtlinien, Rollen und Konfigurationen als Teil der Datenbanksystemtabellen beibehalten. Die geklonte oder wiederhergestellte Datenbank erbt alle vorhandenen Richtlinien aus dem Quellcluster. Nach dem Klonen oder Wiederherstellen behält jeder Datenbankcluster unabhängige Richtlinien bei. `pg_columnmask`

Logische Replikation

Während der ersten Synchronisation verwendet die logische Replikation standardmäßige SQL COPY-Operationen, und `pg_columnmask` Richtlinien werden auf der Grundlage der Berechtigungen des Replikationsbenutzers durchgesetzt. Während der laufenden CDC (Change Data Capture) werden keine Maskierungsrichtlinien angewendet und unmaskierte Daten werden über WAL-Datensätze repliziert. Benutzer mit `pg_create_subscription` Rechten können möglicherweise unmaskierte Daten exfiltrieren, indem sie die Replikation auf ein von ihnen kontrolliertes System einrichten.

Blau/Grün-Bereitstellungen

Bei der Wiederherstellung von Snapshots werden `pg_columnmask` Richtlinien automatisch berücksichtigt. Die grüne Umgebung beginnt mit einer identischen Kopie aller Richtlinien aus

der blauen Umgebung. Bei der Replikation von blau nach grün werden Daten nicht maskiert. Nachfolgende Änderungen der Maskierungsrichtlinien (DDL-Befehle) auf dem blauen Cluster werden nicht auf den grünen Cluster repliziert und machen RDS-Bereitstellungen ungültig. blue/green

Zero-ETL- und CDC-Streams

Die Datenreplikation wird durch Richtlinien nicht beeinträchtigt. `pg_columnmask` Zero-ETL unterstützt die DDL-Replikation, aber keine `pg_columnmask` Replikations- oder RLS-Richtlinien. In Zero-ETL werden keine Maskierungsrichtlinien auf replizierte Daten angewendet.

AWS Database Migration Service

Die anfängliche Datensynchronisierung wird je nach dem für die DMS-Aufgabe ausgewählten Benutzer maskiert oder entlarvt. CDC-Daten werden immer demaskiert. `pg_columnmask`Verwandte interne RLS-Richtlinien können zwar migriert werden, sie funktionieren jedoch nicht auf Zielen, die nicht pg_columnmask-fähig sind.

Datenexporte

`pg_columnmask`behandelt Exporte wie jede andere Abfrageoperation — die Maskierung wird auf der Grundlage der Berechtigungen des ausführenden Benutzers angewendet. Dies gilt für SQL-Befehle wie COPY, SELECT INTO, CREATE TABLE AS und die S3-Exportfunktion von Aurora PostgreSQL.

> **Note**
>
> Wenn maskierte Benutzer Daten exportieren, enthalten die resultierenden Dateien maskierte Werte, die bei der Wiederherstellung gegen Datenbankbeschränkungen verstoßen können.

Ansichten und materialisierte Ansichten

Beachten Sie bei der Verwendung von Ansichten die folgenden Überlegungen:

- Reguläre Ansichten — Verwenden Sie immer `INVOKER` Semantik. Die Maskierungsrichtlinien des aktuellen Benutzers gelten bei der Abfrage der Ansicht, unabhängig davon, wer die Ansicht erstellt hat.
- Materialisierte Ansichten — Bei der Aktualisierung gelten die Maskierungsrichtlinien des Besitzers der materialisierten Ansicht, nicht die Richtlinien des Benutzers, der die Aktualisierung durchführt.

Wenn der Besitzer über Maskierungsrichtlinien verfügt, enthält die materialisierte Ansicht immer maskierte Daten.

Daten speichern und wiederherstellen

`pg_dump`arbeitet wie ein normaler Datenbankbenutzer und wendet Maskierungsrichtlinien an, die auf den Berechtigungen des verbindenden Benutzers basieren. Wenn ein maskierter Benutzer einen Speicherauszug durchführt, enthält die Sicherungsdatei maskierte Daten. `pg_columnmask`Richtlinien sind im Dump als Teil des Datenbankschemas enthalten. Eine erfolgreiche Wiederherstellung setzt voraus, dass alle referenzierten Rollen in der Zieldatenbank vorhanden sind und dass die `pg_columnmask` Erweiterung auf dem Ziel installiert ist.

> **Note**
>
> `pg_dump`Unterstützt ab PostgreSQL 18 die `–no-policies` Option, die sowohl Row Level Security (RLS) als auch `pg_columnmask` Maskierungsrichtlinien von Datenbank-Dumps ausschließt. Weitere Informationen finden Sie unter pg_dump.

Wrapper für ausländische Daten

Bei der Verwendung von Fremddaten-Wrappern werden Maskierungsrichtlinien auf Remotetabellen auf der Grundlage der Berechtigungen des zugewiesenen Benutzers auf dem Quellserver angewendet, nicht auf den Berechtigungen des lokalen abfragenden Benutzers. Sie können zwar über FDW auf maskierte Remotedaten zugreifen, aber Sie können DDM- oder RLS-Richtlinien nicht direkt für Fremdtabellen in Ihrer lokalen Datenbank erstellen.

Aktualisieren von Anwendungen für die Verbindung zu Aurora-PostgreSQL-DB-Clustern mit neuen SSL/TLS-Zertifikaten

Am 13. Januar 2023 veröffentlichte Amazon RDS neue Zertifizierungsstellen-Zertifikate (Certificate Authority, CA) zum Herstellen von Verbindungen mit Ihren Aurora-DB-Clustern mithilfe von Secure Socket Layer oder Transport Layer Security (SSL/TLS). Im Folgenden finden Sie Informationen dazu, wie Sie Ihre Anwendungen aktualisieren, um die neuen Zertifikate verwenden zu können.

In diesem Thema finden Sie Informationen dazu, wie Sie ermitteln, ob Client-Anwendungen für die Herstellung von Verbindungen mit Ihren DB-Clustern SSL/TLS verwenden. Wenn dies der

Fall ist, können Sie weiter überprüfen, ob diese Anwendungen zur Herstellung von Verbindungen Zertifikatverifizierungen erfordern.

> **Note**
>
> Einige Anwendungen sind so konfiguriert, dass sie nur dann Verbindungen mit Aurora PostgreSQL-DB-Clustern herstellen, wenn sie das Zertifikat auf dem Server erfolgreich identifizieren können.
> Für solche Anwendungen müssen Sie die Trust Stores Ihrer Client-Anwendung aktualisieren, damit diese die neuen CA-Zertifikate enthalten.

Nach der Aktualisierung der CA-Zertifikate in den Trust Stores Ihrer Client-Anwendungen können Sie die Zertifikate auf Ihren DB-Clustern rotieren. Es wird nachdrücklich empfohlen, diese Verfahren vor der Implementierung in Produktionsumgebungen in einer Entwicklungs- oder Testumgebung zu testen.

Weitere Informationen zur Zertifikatrotation finden Sie unter Rotieren Ihrer SSL/TLS-Zertifikate. Weitere Informationen zum Herunterladen von Zertifikaten finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln. Informationen zum Verwenden von SSL/TLS mit PostgreSQL-DB-Clustern finden Sie unter Sicherung von Aurora-PostgreSQL-Daten mit SSL/TLS.

Themen

- Ermitteln, ob Anwendungen Verbindungen mit Aurora-PostgreSQL-DB-Clustern mithilfe von SSL herstellen
- Ermitteln, ob ein Client zum Herstellen von Verbindungen Zertifikatverifizierungen erfordert
- Aktualisieren des Trust Stores Ihrer Anwendung
- Verwenden von SSL/TLS-Verbindungen für verschiedene Arten von Anwendungen

Ermitteln, ob Anwendungen Verbindungen mit Aurora-PostgreSQL-DB-Clustern mithilfe von SSL herstellen

Prüfen Sie die DB-Cluster-Konfiguration auf den Wert des `rds.force_ssl`-Parameters. Standardmäßig ist der Parameter `rds.force_ssl` auf `0` festgelegt. Wenn der Parameter `rds.force_ssl` auf 1 (ein) festgelegt ist, müssen Clients SSL/TLS für Verbindungen verwenden. Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Wenn `rds.force_ssl` nicht auf 1 (an) festgelegt ist, fragen Sie `pg_stat_ssl` ab, um zu prüfen, welche Verbindungen SSL verwenden. Beispielsweise gibt die folgende Abfrage nur SSL-Verbindungen und Informationen zu den Clients zurück, die SSL verwenden.

```
select datname, usename, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity
 on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and usename<>'rdsadmin';
```

Nur Zeilen, die SSL/TLS-Verbindungen verwenden, werden mit Informationen zur Verbindung angezeigt. Dies ist eine Beispielausgabe.

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
(2 rows)
```

Die vorherige Abfrage zeigt nur die aktuellen Verbindungen zum Zeitpunkt der Abfrage an. Das Fehlen von Ergebnissen weist nicht darauf hin, dass es keine Anwendungen gibt, die SSL-Verbindungen verwenden. Möglicherweise werden zu anderen Zeitpunkten weitere SSL-Verbindungen hergestellt.

Ermitteln, ob ein Client zum Herstellen von Verbindungen Zertifikatverifizierungen erfordert

Wenn ein Client wie psql oder JDBC mit SSL-Unterstützung konfiguriert ist, versucht dieser zunächst standardmäßig, die Verbindung zur Datenbank über SSL herzustellen. Wenn der Client keine Verbindung über SSL herstellen kann, stellt er die Verbindung ohne SSL her. Der für libpq-basierte Clients (wie psql) und JDBC verwendete `sslmode`-Standardmodus ist auf `prefer` festgelegt. Das Zertifikat auf dem Server wird nur verifiziert, wenn für `sslrootcert` die Einstellung `sslmode` auf `verify-ca` oder `verify-full` festgelegt ist. Wenn das Zertifikat ungültig ist, wird ein Fehler ausgelöst.

Verwenden Sie `PGSSLROOTCERT` zur Verifizierung des Zertifikats mit der Umgebungsvariablen `PGSSLMODE`, wobei `PGSSLMODE` auf `verify-ca` oder `verify-full` festgelegt wird.

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

Verwenden Sie das Argument `sslrootcert` zur Verifizierung des Zertifikats mit `sslmode` im Verbindungszeichenfolgenformat, wobei `sslmode` auf `verify-ca` oder `verify-full` festgelegt ist.

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=primaryuser dbname=postgres"
```

Wenn Sie beispielsweise in vorherigen Fall ein ungültiges Stammzertifikat verwenden, sehen Sie auf Ihrem Client einen Fehler ähnlich dem folgenden.

```
psql: SSL error: certificate verify failed
```

Aktualisieren des Trust Stores Ihrer Anwendung

Informationen zum Aktualisieren des Trust Stores für PostgreSQL-Anwendungen finden Sie unter Secure TCP/IP Connections with SSL in der PostgreSQL-Dokumentation.

> **Note**
>
> Wenn Sie den Trust Store aktualisieren, können Sie ältere Zertifikate beibehalten und die neuen Zertifikate einfach hinzufügen.

Aktualisieren des Trust Stores Ihrer Anwendung für JDBC

Sie können den Trust Store für Anwendungen aktualisieren, die JDBC für SSL/TLS-Verbindungen verwenden.

Informationen zum Herunterladen des Stammverzeichnisses finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

Beispiele für Skripte, die Zertifikate importieren, finden Sie unter Beispielskript für den Import von Zertifikaten in Ihren Trust Store.

Verwenden von SSL/TLS-Verbindungen für verschiedene Arten von Anwendungen

Im Folgenden finden Sie Informationen zum Verwenden von SSL/TLS-Verbindungen für verschiedene Arten von Anwendungen:

- psql

  Der Client wird über die Befehlszeile durch die Angabe von Optionen als Verbindungszeichenfolge oder Umgebungsvariablen aufgerufen. Im Fall von SSL/TLS-Verbindungen sind die relevanten Optionen `sslmode` (Umgebungsvariable `PGSSLMODE`), `sslrootcert` (Umgebungsvariable `PGSSLROOTCERT`).

  Die vollständige Liste der Optionen finden Sie unter Parameter Key Words in der PostgreSQL-Dokumentation. Die vollständige Liste der Umgebungsvariablen finden Sie unter Environment Variables in der PostgreSQL-Dokumentation.

- pgAdmin

  Dieser browserbasierte Client bietet eine benutzerfreundlichere Oberfläche zum Herstellen von Verbindungen mit PostgreSQL-Datenbanken.

  Informationen zum Konfigurieren von Verbindungen finden Sie in der pgAdmin-Dokumentation.

- JDBC

  JDBC ermöglicht Datenbankverbindungen mit Java-Anwendungen.

  Allgemeine Informationen zum Herstellen von Verbindungen mit PostgreSQL-Datenbanken über JDBC finden Sie unter Connecting to the Database in der PostgreSQL-Dokumentation. Informationen zum Herstellen von Verbindungen über SSL/TLS finden Sie unter Configuring the Client in der PostgreSQL-Dokumentation.

- Python

  Eine verbreitet für die Herstellung von Verbindungen mit PostgreSQL-Datenbanken verwendete Python-Bibliothek ist `psycopg2`.

  Informationen zum Verwenden von `psycopg2` finden Sie in der psycopg2-Dokumentation. Ein kurzes Tutorial zum Herstellen von Verbindungen mit PostgreSQL-Datenbanken finden Sie unter Psycopg2-Tutorial. Informationen zu den vom Verbindungsbefehl akzeptierten Optionen finden Sie unter psycopg2-Modulinhalte.

> **Important**
>
> Wenn Sie festgestellt haben, dass Ihre Datenbankverbindungen SSL/TLS verwenden, und Sie den Trust Store Ihrer Anwendung aktualisiert haben, können Sie Ihre Datenbank so

aktualisieren, dass sie die rds-ca-rsa2048-g1-Zertifikate verwendet. Anleitungen hierzu finden Sie in Schritt 3 unter Aktualisieren des CA-Zertifikats durch Ändern der DB-Instance.

Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL

Sie können Kerberos verwenden, um Benutzer zu authentifizieren, wenn sie sich mit Ihrem DB-Cluster mit PostgreSQL verbinden. Konfigurieren Sie dazu Ihre so, dass sie AWS Directory Service for Microsoft Active Directory für die Kerberos-Authentifizierung verwendet wird. AWS Directory Service for Microsoft Active Directory wird auch genannt. AWS Managed Microsoft AD Es ist eine Funktion, die mit verfügbar ist Directory Service. Weitere Informationen finden Sie unter Was ist Directory Service? im AWS Directory Service Administratorhandbuch.

Erstellen Sie zunächst ein AWS Managed Microsoft AD Verzeichnis zum Speichern von Benutzeranmeldeinformationen. Anschließend stellen Sie Ihrem PostgreSQL-DB-Cluster die Active Directory Domain und weitere Informationen zur Verfügung. Wenn Benutzer sich mit PostgreSQL-DB-Clustern authentifizieren, werden Authentifizierungsanforderungen an das AWS Managed Microsoft AD -Verzeichnis weitergeleitet.

Wenn Sie alle Ihre Anmeldeinformationen im selben Verzeichnis aufbewahren, können Sie Zeit und Mühe sparen. Sie haben einen zentralen Ort für die Speicherung und Verwaltung von Anmeldeinformationen für mehrere DB-Cluster. Die Verwendung eines Verzeichnisses kann auch Ihr allgemeines Sicherheitsprofil verbessern.

Außerdem können Sie von Ihrem eigenen On-Premises Microsoft Active Directory auf Anmeldeinformationen zugreifen. Dazu erstellen Sie eine vertrauensvolle Domain-Beziehung, damit das AWS Managed Microsoft AD -Verzeichnis Ihrem On-Premises Microsoft Active Directory vertraut. Auf diese Weise können Ihre Benutzer auf Ihre PostgreSQL-Clusters- mit derselben Windows Single Sign-On-Oberfläche (SSO) zugreifen, die sie auch für den Zugriff auf Workloads in Ihrem lokalen Netzwerk verwenden.

Eine Datenbank kann Kerberos AWS Identity and Access Management (IAM) oder sowohl Kerberos- als auch IAM-Authentifizierung verwenden. Da die Kerberos- und IAM-Authentifizierung jedoch unterschiedliche Authentifizierungsmethoden bereitstellen, kann sich ein bestimmter Datenbankbenutzer nur mit der einen oder anderen Authentifizierungsmethode bei einer Datenbank anmelden, jedoch nicht mit beiden. Weitere Informationen zur IAM-Authentifizierung finden Sie unter IAM-Datenbankauthentifizierung.

> **Note**
>
> RDS für PostgreSQL unterstützt keine Kerberos-Authentifizierung für Active-Directory-Gruppen.

Themen

- Verfügbarkeit von Regionen und Versionen
- Übersicht über die Kerberos-Authentifizierung für PostgreSQL-DB-Cluster
- Einrichten der Kerberos-Authentifizierung für PostgreSQL-DB-Cluster
- Verwalten eines DB-Clusters von Aurora PostgreSQL in einer Active Directory-Domain
- Herstellen einer Verbindung zu PostgreSQL mit Kerberos-Authentifizierung
- Verwenden von AD-Sicherheitsgruppen für die Aurora-PostgreSQL-Zugriffskontrolle

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen über die Verfügbarkeit von Versionen und Regionen von Aurora PostgreSQL mit Kerberos-Authentifizierung finden Sie unter Kerberos-Authentifizierung mit Aurora PostgreSQL.

Übersicht über die Kerberos-Authentifizierung für PostgreSQL-DB-Cluster

Um die Kerberos-Authentifizierung für PostgreSQL-DB-Cluster einzurichten, führen Sie die folgenden Schritte aus, die später näher erläutert werden:

1. Wird verwendet AWS Managed Microsoft AD , um ein Verzeichnis zu erstellen. AWS Managed Microsoft AD Sie können die AWS-Managementkonsole, oder die Directory Service API verwenden AWS CLI, um das Verzeichnis zu erstellen. Stellen Sie sicher, dass Sie die relevanten ausgehenden Ports in der Verzeichnissicherheitsgruppe öffnen, damit das Verzeichnis mit der kommunizieren kann.
2. Erstellen Sie eine Rolle, die Amazon Aurora Amazon gewährt, um Anrufe in Ihr AWS Managed Microsoft AD Verzeichnis zu tätigen. Erstellen Sie dazu eine AWS Identity and Access Management (IAM-) Rolle, die die verwaltete IAM-Richtlinie verwendet. `AmazonRDSDirectoryServiceAccess`

Damit die IAM-Rolle Zugriff gewährt, muss der Endpunkt AWS Security Token Service (AWS STS) in der richtigen AWS Region für Ihr Konto aktiviert sein. AWS AWS STS Endpunkte sind standardmäßig in allen aktiv AWS-Regionen, und Sie können sie ohne weitere Aktionen verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Aktivierung und Deaktivierung AWS STS in einer AWS Region.

3. Erstellen und konfigurieren Sie Benutzer im AWS Managed Microsoft AD Verzeichnis mithilfe der Microsoft Active Directory-Tools. Weitere Informationen zum Erstellen von Benutzern in Ihrem Active Directory finden Sie unter Verwalten von Benutzern und Gruppen in AWS Managed Microsoft AD im Directory Service Administratorhandbuch.
4. Wenn Sie planen, das Verzeichnis und die DB-Instance in verschiedenen AWS Konten oder virtuellen privaten Clouds (VPCs) zu lokalisieren, konfigurieren Sie VPC-Peering. Weitere Informationen finden Sie unter Was ist VPC Peering? im Amazon VPC Peering Guide.
5. Erstellen oder ändern Sie PostgreSQL-DB-Cluster entweder über die Konsole, CLI oder RDS-API mit einer der folgenden Methoden:
   - Erstellen eines DB-Clusters von Aurora PostgreSQL und Herstellen einer Verbindung
   - Ändern eines Amazon Aurora-DB-Clusters
   - Wiederherstellen aus einem DB-Cluster-Snapshot
   - Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit

   Sie können die in derselben Amazon Virtual Private Cloud (VPC) wie das Verzeichnis oder in einem anderen AWS Konto oder einer anderen VPC lokalisieren. Wenn Sie die PostgreSQL-DB- erstellen oder ändern, gehen Sie wie folgt vor:

   - Geben Sie den Domänenbezeichner (`d-*`-Bezeichner) an, der beim Erstellen Ihres Verzeichnisses generiert wurde.
   - Geben Sie außerdem den Namen der IAM-Rolle an, die Sie erstellt haben.
   - Stellen Sie sicher, dass die Sicherheitsgruppe der DB-Instance eingehenden Datenverkehr von der Sicherheitsgruppe des Verzeichnisses empfangen kann.
6. Verwenden Sie die RDS-Master-Benutzer-Anmeldeinformationen, um sich mit PostgreSQL-DB-Clustern zu verbinden. Erstellen Sie den Benutzer in PostgreSQL, der extern identifiziert werden soll. Extern identifizierte Benutzer können sich über die Kerberos-Authentifizierung bei der PostgreSQL-DB- anmelden.

Einrichten der Kerberos-Authentifizierung für PostgreSQL-DB-Cluster

Um die Kerberos-Authentifizierung einzurichten, führen Sie die folgenden Schritte aus.

Themen

- Schritt 1: Erstellen Sie ein Verzeichnis mit AWS Managed Microsoft AD
- Schritt 2: (Optional) Erstellen einer Vertrauensbeziehung zwischen Ihrem On-Premises Active Directory und Directory Service
- Schritt 3: Erstellen Sie eine IAM-Rolle für Amazon Aurora Amazon für den Zugriff auf Directory Service
- Schritt 4: Anlegen und Konfigurieren von Benutzern
- Schritt 5: Aktivieren des VPC-übergreifenden Datenverkehrs zwischen dem Verzeichnis und der DB-Instance
- Schritt 6: Erstellen oder Ändern von PostgreSQL-DB-
- Schritt 7: Erstellen von PostgreSQL-Benutzern für Ihre Kerberos-Prinzipale
- Schritt 8: Konfigurieren eines PostgreSQL-Clients

Schritt 1: Erstellen Sie ein Verzeichnis mit AWS Managed Microsoft AD

Directory Service erstellt ein vollständig verwaltetes Active Directory in der AWS Cloud. Wenn Sie ein AWS Managed Microsoft AD Verzeichnis erstellen, Directory Service erstellt zwei Domänencontroller und DNS-Server für Sie. Die Verzeichnisserver werden in verschiedenen Subnetzen in einer VPC erstellt. Diese Redundanz trägt dazu bei, dass Ihr Verzeichnis auch im Fehlerfall erreichbar bleibt.

Wenn Sie ein AWS Managed Microsoft AD Verzeichnis erstellen, führt der AWS Directory Service die folgenden Aufgaben in Ihrem Namen aus:

- Richtet ein Active Directory in Ihrer VPC ein.
- Erstellt ein Konto für den Verzeichnisadministrator mit dem Benutzernamen `Admin` und dem angegebenen Passwort. Mit diesem Konto verwalten Sie das Verzeichnis.

> **Important**
>
> Stellen Sie sicher, dass Sie dieses Passwort speichern. Directory Service speichert dieses Passwort nicht und es kann nicht abgerufen oder zurückgesetzt werden.

- Erstellt eine Sicherheitsgruppe für die Verzeichniscontroller. Die Sicherheitsgruppe muss die Kommunikation mit der PostgreSQL-DB- zulassen.

AWS Erstellt beim Start AWS Directory Service for Microsoft Active Directory eine Organisationseinheit (OU), die alle Objekte Ihres Verzeichnisses enthält. Diese OU erhält den NetBIOS-Namen, den Sie beim Erstellen des Verzeichnisses eingegeben haben, und befindet sich im Domänenstamm. Der Domänenstamm gehört und wird von diesem verwaltet AWS.

Das `Admin` Konto, das mit Ihrem AWS Managed Microsoft AD Verzeichnis erstellt wurde, verfügt über Berechtigungen für die gängigsten Verwaltungsaktivitäten Ihrer Organisationseinheit:

- Erstellen, Aktualisieren oder Löschen von Benutzern
- Hinzufügen von Ressourcen zu Ihrer Domäne, etwa Datei- oder Druckserver, und anschließendes Gewähren der zugehörigen Ressourcenberechtigungen für Benutzer in der OU
- Zusätzliche OUs Container erstellen
- Delegieren von Befugnissen
- Wiederherstellen von gelöschten Objekten aus dem Active Directory-Papierkorb
- Führen Sie Active Directory- und DNS-Module (Domain Name Service) für Windows PowerShell im Active Directory-Webdienst aus

Das `Admin`-Konto hat auch die Berechtigung, die folgenden domänenweiten Aktivitäten durchzuführen:

- Verwalten von DNS-Konfigurationen (Hinzufügen, Entfernen oder Aktualisieren von Datensätzen, Zonen und Weiterleitungen)
- Aufrufen von DNS-Ereignisprotokollen
- Anzeigen von Sicherheitsereignisprotokollen

Um ein Verzeichnis zu erstellen mit AWS Managed Microsoft AD

1. Wählen Sie im Navigationsbereich Directory Service -Konsole den Eintrag Directories (Verzeichnisse) und wählen Sie Set up directory (Verzeichnis einrichten) aus.
2. Wählen Sie AWS Managed Microsoft AD. AWS Managed Microsoft AD ist die einzige Option, die derzeit für die Verwendung mit Amazon Aurora Amazon unterstützt wird.
3. Wählen Sie Weiter aus.

4. Geben Sie auf der Seite Enter directory information (Verzeichnisinformationen eingeben) die folgenden Informationen ein:

   Edition

   Wählen Sie die Edition aus, die Ihre Anforderungen erfüllt.

   DNS-Name des Verzeichnisses

   Den vollständig qualifizierten Namen für das Verzeichnis, z. B. **`corp.example.com`**.

   NetBIOS-Name des Verzeichnisses

   Ein optionaler Kurzname für das Verzeichnis, z. B. `CORP`.

   Verzeichnisbeschreibung

   Eine optionale Beschreibung des Verzeichnisses.

   Administratorpasswort

   Das Passwort für den Verzeichnisadministrator. Mit der Verzeichniserstellung wird ein Administratorkonto mit dem Benutzernamen `Admin` und diesem Passwort angelegt.

   Das Passwort für den Verzeichnisadministrator darf nicht das Wort "admin" enthalten. Beachten Sie beim Passwort die Groß- und Kleinschreibung und es muss 8 bis 64 Zeichen lang sein. Zudem muss es mindestens ein Zeichen aus dreien der vier folgenden Kategorien enthalten:

   - Kleinbuchstaben (a–z)
   - Großbuchstaben (A–Z)
   - Zahlen (0–9)
   - Nicht-alphanumerische Zeichen (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

   Passwort bestätigen

   Geben Sie das Administratorpasswort erneut ein.

   > **Important**
   >
   > Stellen Sie sicher, dass Sie dieses Passwort speichern. Directory Service speichert dieses Passwort nicht und es kann nicht abgerufen oder zurückgesetzt werden.

5. Wählen Sie Weiter aus.

6. Geben Sie auf der Seite Choose VPC and subnets (VPC und Subnetze wählen) die folgenden Informationen an.

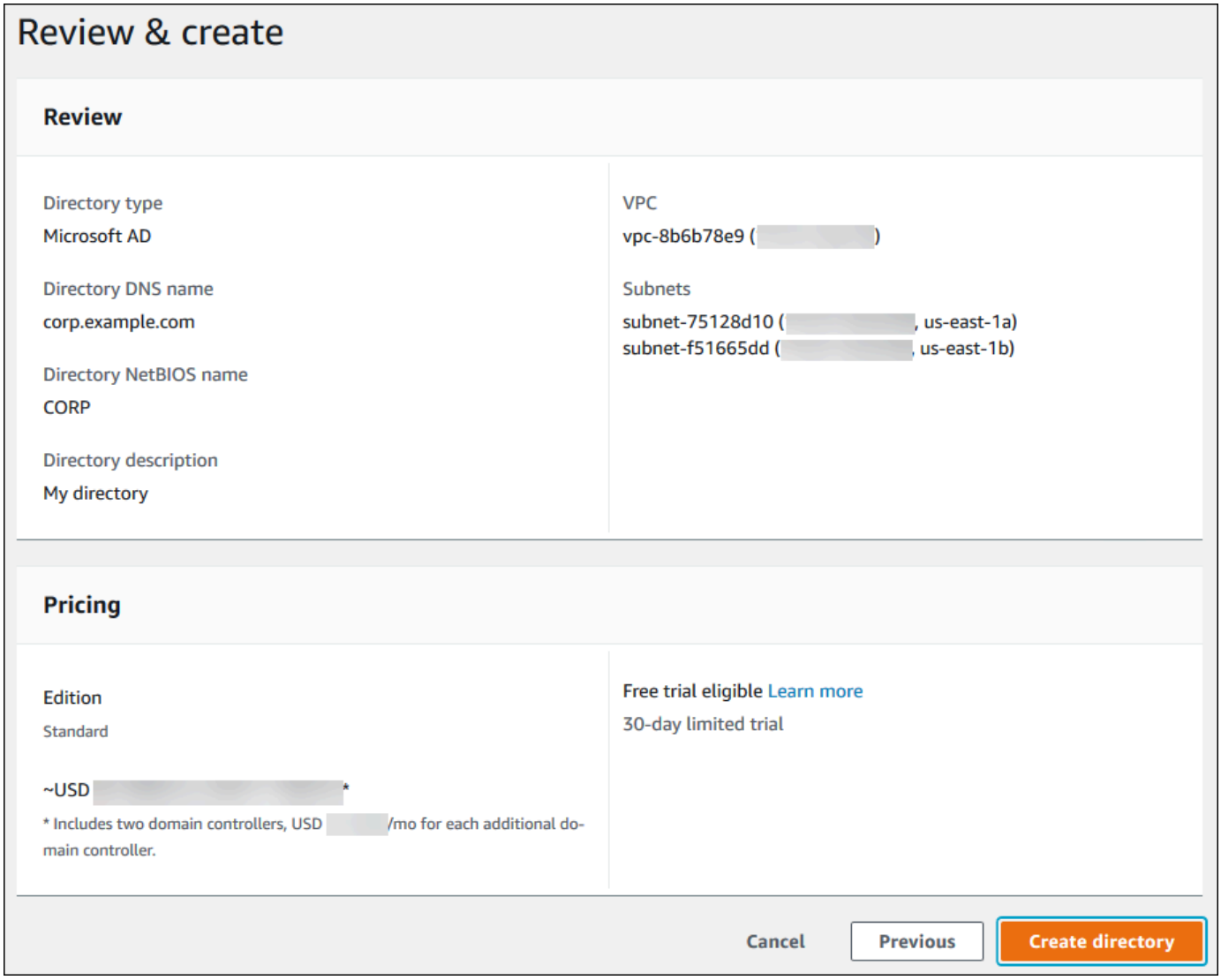
   VPC

   Wählen Sie die VPC für das Verzeichnis aus. Sie können PostgreSQL-DB-Cluster in derselben VPC oder in einer anderen VPC erstellen.

   Subnetze

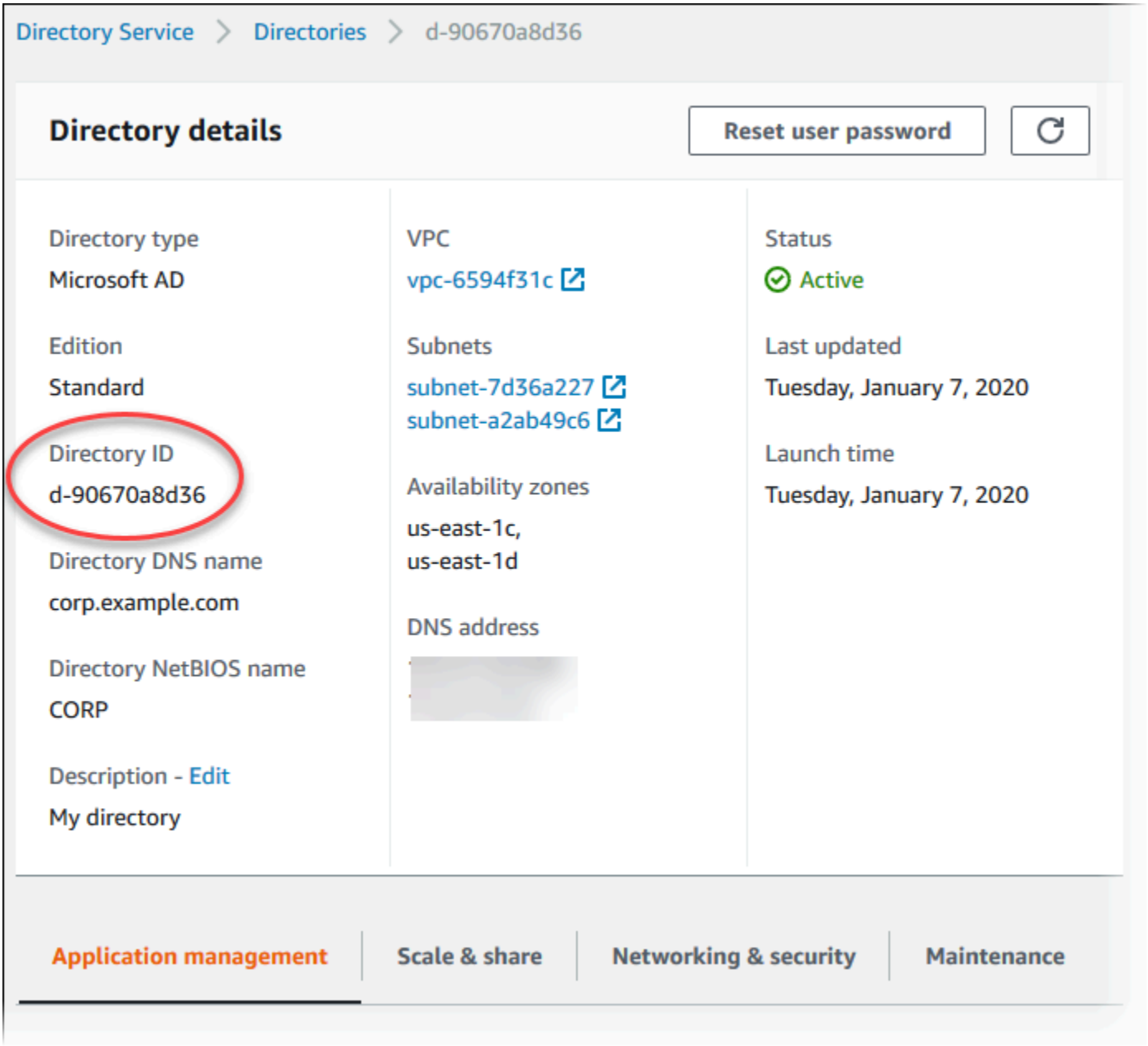
   Wählen Sie Subnetze für die Verzeichnis-Server aus. Die beiden Subnetze müssen zu verschiedenen Availability-Zonen gehören.

7. Wählen Sie Weiter aus.
8. Überprüfen Sie die Verzeichnisinformationen. Wenn Änderungen erforderlich sind, klicken Sie auf Previous (Zurück) und nehmen Sie die Änderungen vor. Wenn die Informationen richtig sind, wählen Sie Create directory (Verzeichnis erstellen).



Es dauert einige Minuten, bis das Verzeichnis erstellt wurde. Wenn es erfolgreich erstellt wurde, ändert sich der Wert Status in Active (Aktiv).

Um Informationen über das Verzeichnis anzuzeigen, wählen Sie die Verzeichnis-ID in der Verzeichnisauflistung aus. Notieren Sie sich den Wert Directory ID. Sie benötigen diesen Wert, wenn Sie Ihre PostgreSQL DB-Instance erstellen oder ändern.



Schritt 2: (Optional) Erstellen einer Vertrauensbeziehung zwischen Ihrem On-Premises Active Directory und Directory Service

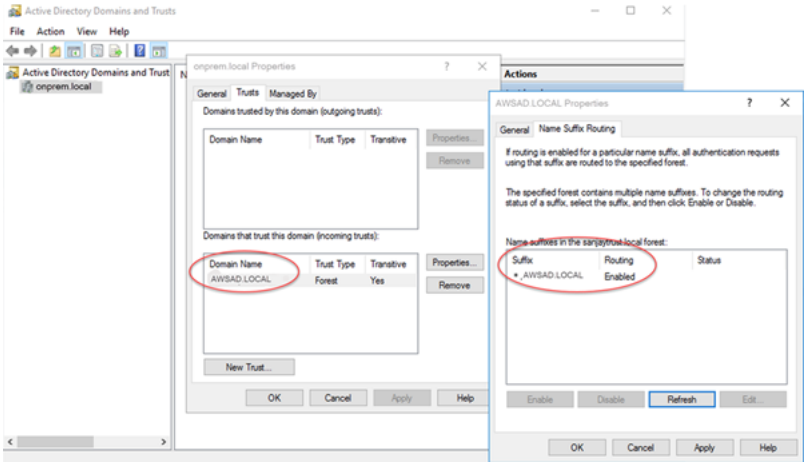
Wenn Sie Ihr eigenes lokales Microsoft Active Directory nicht verwenden möchten, fahren Sie mit for Schritt 3: Erstellen Sie eine IAM-Rolle für Amazon Aurora Amazon für den Zugriff auf Directory Service.

Um die Kerberos-Authentifizierung mit Ihrem lokalen Active Directory zu erhalten, müssen Sie eine vertrauensvolle Domänenbeziehung mithilfe einer Gesamtvertrauensstellung zwischen Ihrem lokalen Microsoft Active Directory und dem AWS Managed Microsoft AD Verzeichnis (erstellt in) einrichten.

Schritt 1: Erstellen Sie ein Verzeichnis mit AWS Managed Microsoft AD Die Vertrauensstellung kann unidirektional sein, wobei das AWS Managed Microsoft AD Verzeichnis dem lokalen Microsoft Active Directory vertraut. Die Vertrauensstellung kann auch bidirektional erfolgen, wobei beide Active Directories einander vertrauen. Weitere Informationen zum Einrichten von Vertrauensstellungen mithilfe von finden Sie unter Wann Directory Service sollte eine Vertrauensstellung eingerichtet werden? im Administratorhandbuch.AWS Directory Service

> **Note**
>
> Verwendung eines lokalen Microsoft Active Directory:
>
> - Windows-Clients müssen die Verbindung über den Domänennamen auf dem Endpunkt herstellen und nicht über Directory Service rds.amazonaws.com. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu PostgreSQL mit Kerberos-Authentifizierung.
> - Windows-Clients können keine Verbindung über benutzerdefinierten Aurora-Endpunkte herstellen. Weitere Informationen hierzu finden Sie unter Amazon-Aurora-Endpunktverbindungen.
> - Globale Datenbanken:
>   - Windows-Clients können eine Verbindung nur über Instanzendpunkte oder Cluster-Endpunkte in der primären AWS-Region Datenbank herstellen.
>   - Windows-Clients können keine Verbindung über Clusterendpunkte auf der Sekundärseite herstellen. AWS-Regionen

Stellen Sie sicher, dass der lokale Microsoft Active Directory-Domänenname ein DNS-Suffix-Routing enthält, das der neu erstellten Vertrauensstellung entspricht. Im folgenden Screenshot wird ein Beispiel gezeigt.



Schritt 3: Erstellen Sie eine IAM-Rolle für Amazon Aurora Amazon für den Zugriff auf Directory Service

Damit Amazon Aurora Sie anrufen Directory Service kann, benötigt Ihr AWS Konto eine IAM-Rolle, die die verwaltete IAM-Richtlinie verwendet. `AmazonRDSDirectoryServiceAccess` Diese Rolle ermöglicht es Amazon Aurora, Aufrufe von Directory Service durchzuführen. (Beachten Sie, dass sich diese IAM-Rolle für den Zugriff von der IAM-Rolle Directory Service unterscheidet, für die Sie verwendet werden.) IAM-Datenbankauthentifizierung

Wenn Sie mit dem eine DB-Instance erstellen AWS-Managementkonsole und Ihr Konsolen-Benutzerkonto über die `iam:CreateRole` entsprechende Berechtigung verfügt, erstellt die Konsole automatisch die benötigte IAM-Rolle. In diesem Fall lautet der Rollenname `rds-directoryservice-kerberos-access-role`. Andernfalls müssen Sie die IAM-Rolle manuell erstellen. Wenn Sie diese IAM-Rolle erstellen`Directory Service`, wählen Sie die AWS verwaltete Richtlinie aus und fügen Sie `AmazonRDSDirectoryServiceAccess` sie ihr hinzu.

Weitere Informationen zum Erstellen von IAM-Rollen für einen Dienst finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS Dienst im IAM-Benutzerhandbuch.

> **Note**
>
> Die für die Windows-Authentifizierung für RDS für Microsoft SQL Server verwendete IAM-Rolle kann nicht für Amazon Aurora verwendet werden.

Alternativ können Sie Richtlinien mit den erforderlichen Berechtigungen erstellen, anstatt die verwaltete Richtlinie `AmazonRDSDirectoryServiceAccess` zu verwenden. In diesem Fall muss die IAM-Rolle die folgende IAM-Vertrauensrichtlinie haben.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Die Rolle muss auch über die folgende IAM-Rollenrichtlinie verfügen.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Wenn Sie sich anmelden möchten AWS-Regionen, verwenden Sie regionsspezifische Dienstprinzipale in den Vertrauensrichtlinien für IAM-Rollen. Wenn Sie eine Vertrauensrichtlinie für Services in diesen Regionen erstellen, geben Sie den Regionalcode im Service-Prinzipal an.

Das folgende Beispiel zeigt eine Vertrauensrichtlinie, die regionsspezifischen Service-Prinzipale enthält.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.REGION-CODE.amazonaws.com",
          "rds.REGION-CODE.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Ersetzen Sie REGION-CODE durch den Code für Ihre spezifische Region. Verwenden Sie beispielsweise die folgenden Service-Prinzipale für die Region Asien-Pazifik (Melbourne):

```
"Service": [
  "directoryservice.rds.ap-southeast-4.amazonaws.com",
  "rds.ap-southeast-4.amazonaws.com"
]
```

Schritt 4: Anlegen und Konfigurieren von Benutzern

Sie können Benutzer mit dem Tool „Active Directory Users and Computers“ erstellen. Dieses Tool ist ein Active Directory Domain Service und ein Active Directory Lightweight Directory Service. Weitere

Informationen finden Sie unter Hinzufügen von Benutzern und Computern zur Active-Directory-Domain. In diesem Fall handelt es sich bei Benutzern um Einzelpersonen oder andere Entitäten, z. B. um ihre Computer, die Teil der Domain sind und deren Identitäten im Verzeichnis verwaltet werden.

Um Benutzer in einem Directory Service Verzeichnis zu erstellen, müssen Sie mit einer Windows-basierten EC2 Amazon-Instance verbunden sein, die Mitglied des Directory Service Verzeichnisses ist. Gleichzeitig müssen Sie als Benutzer angemeldet sein, der über Berechtigungen zum Erstellen von Benutzern verfügt. Weitere Informationen finden Sie unter Erstellen eines Benutzers im AWS Directory Service Administration Guide.

Schritt 5: Aktivieren des VPC-übergreifenden Datenverkehrs zwischen dem Verzeichnis und der DB-Instance

Wenn Sie beabsichtigen, das Verzeichnis und den DB-Cluster in derselben VPC zu platzieren, überspringen Sie diesen Schritt und fahren Sie mit Schritt 6: Erstellen oder Ändern von PostgreSQL-DB- fort.

Wenn Sie planen, das Verzeichnis und die DB-Instance unterschiedlich zu lokalisieren VPCs, konfigurieren Sie den VPC-übergreifenden Verkehr mithilfe von VPC-Peering oder Transit Gateway.AWS

Das folgende Verfahren ermöglicht den Datenverkehr zwischen VPCs mithilfe von VPC-Peering. Folgen Sie den Anleitungen unter Was ist VPC Peering? im Handbuch zu Amazon Virtual Private Cloud-Peering.

Aktivieren des VPC-übergreifenden Datenverkehrs mit VPC Peering

1. Richten Sie geeignete VPC-Routing-Regeln ein, um sicherzustellen, dass Netzwerk-Datenverkehr in beide Richtungen fließen kann.
2. Stellen Sie sicher, dass die Sicherheitsgruppe der DB-Instance eingehenden Datenverkehr von der Sicherheitsgruppe des Verzeichnisses empfangen kann.
3. Stellen Sie sicher, dass keine ACL-Regel (Network Access Control List) zum Blockieren des Datenverkehrs vorhanden ist.

Wenn ein anderes AWS Konto Eigentümer des Verzeichnisses ist, müssen Sie das Verzeichnis gemeinsam nutzen.

Um das Verzeichnis von mehreren AWS Konten gemeinsam zu nutzen

1. Beginnen Sie mit der gemeinsamen Nutzung des Verzeichnisses mit dem AWS Konto, unter dem die DB-Instance erstellt werden soll. Folgen Sie dazu den Anweisungen im Directory Service Administratorhandbuch Tutorial: Teilen Ihres AWS verwalteten Microsoft AD-Verzeichnisses für einen nahtlosen EC2 Domänenbeitritt.
2. Melden Sie sich mit dem Konto für die DB-Instance bei der Directory Service Konsole an und stellen Sie sicher, dass die Domain den `SHARED` Status hat, bevor Sie fortfahren.
3. Notieren Sie sich den Wert der Verzeichnis-ID, während Sie mit dem Konto für die DB-Instance bei der Directory Service Konsole angemeldet sind. Sie verwenden diese Verzeichnis-ID, um die DB-Instance mit der Domäne zu verbinden.

Schritt 6: Erstellen oder Ändern von PostgreSQL-DB-

Erstellen oder ändern Sie PostgreSQL-DB-Cluster für die Verwendung mit Ihrem Verzeichnis. Sie können die Konsole, CLI oder RDS-API verwenden, um DB-Cluster einem Verzeichnis zuzuordnen. Sie können dafür eine der folgenden Möglichkeiten auswählen:

- Erstellen Sie mit der Konsole, dem create-db-clusterCLI-Befehl oder dem Vorgang Create DBCluster RDS API einen neuen PostgreSQL-DB-Cluster. Detaillierte Anweisungen finden Sie unter Erstellen eines DB-Clusters von Aurora PostgreSQL und Herstellen einer Verbindung.
- Ändern Sie einen vorhandenen PostgreSQL-DB-Cluster mithilfe der Konsole, des modify-db-clusterCLI-Befehls oder der Operation Modify DBCluster RDS API. Detaillierte Anweisungen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.
- Stellen Sie einen PostgreSQL-DB-Cluster aus einem DB-Snapshot mithilfe der Konsole, des CLI-Befehls restore-db-cluster-from-db-snapshot oder der Operation Restore From RDS API wieder her DBCluster. DBSnapshot Detaillierte Anweisungen finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.
- Stellen Sie einen PostgreSQL-DB-Cluster point-in-time mithilfe der Konsole, des Befehls restore-db-instance-to- point-in-time CLI oder der Operation Restore DBCluster ToPointInTime RDS API auf einem wieder her. Detaillierte Anweisungen finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit.

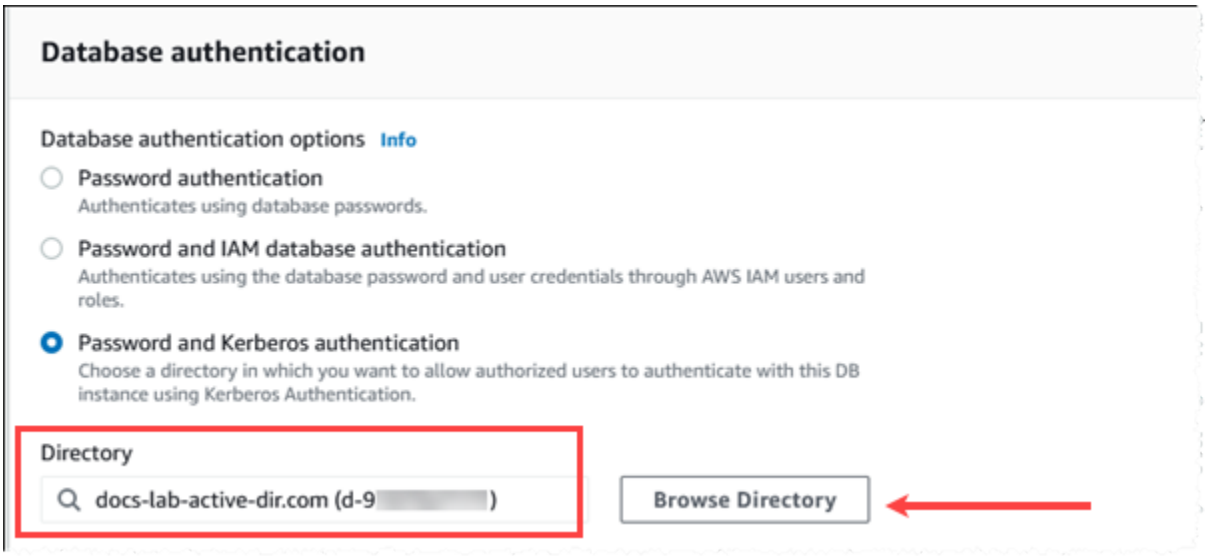
Die Kerberos-Authentifizierung wird nur für PostgreSQL-DB-Cluster- in einer VPC unterstützt. Die DB-Instance kann sich in derselben VPC wie das Verzeichnis oder in einer anderen VPC befinden.

Der DB-Cluster muss eine Sicherheitsgruppe verwenden, die ausgehenden Datenverkehr innerhalb der VPC des Verzeichnisses zulässt, damit der DB-Cluster mit dem Verzeichnis kommunizieren kann.

> **Note**
>
> Die Aktivierung der Kerberos-Authentifizierung wird derzeit auf dem DB-Cluster von Aurora PostgreSQL während der Migration von RDS für PostgreSQL nicht unterstützt. Sie können die Kerberos-Authentifizierung nur auf einem eigenständigen DB-Cluster von Aurora PostgreSQL aktivieren.

Konsole

Wenn Sie die Konsole zum Erstellen, Ändern oder Wiederherstellen eines DB-Clusters verwenden, wählen Sie Kerberos-Authentifizierung im Datenbank-Authentifizierung-Abschnitt. Dann wählen Sie Verzeichnis durchsuchen. Wählen Sie das Verzeichnis aus oder wählen Sie Erstellen eines neuen Verzeichnisses, um den Directory Service zu verwenden.



AWS CLI

Wenn Sie den verwenden AWS CLI, sind die folgenden Parameter erforderlich, damit die das von Ihnen erstellte Verzeichnis verwenden kann:

- Für den `--domain`-Parameter verwenden Sie den Domänenbezeichner („d-*“-Bezeichner), der beim Erstellen des Verzeichnisses generiert wurde.
- Verwenden Sie für den `--domain-iam-role-name`-Parameter die von Ihnen erstellte Rolle, die die verwaltete IAM-Richtlinie `AmazonRDSDirectoryServiceAccess` verwendet.

Beispielsweise ändert der folgende CLI-Befehl einen DB-Cluster zur Verwendung eines Verzeichnisses.

```
aws rds  modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID
 --domain-iam-role-name role-name
```

> **Important**
>
> Wenn Sie einen DB-Cluster ändern, um die Kerberos-Authentifizierung zu aktivieren, starten Sie den DB-Cluster neu, nachdem Sie die Änderung vorgenommen haben.

Schritt 7: Erstellen von PostgreSQL-Benutzern für Ihre Kerberos-Prinzipale

Zu diesem Zeitpunkt ist Ihr DB-Cluster von Aurora PostgreSQL mit der AWS Managed Microsoft AD -Domain verbunden. Die Benutzer, die Sie in dem Verzeichnis in Schritt 4: Anlegen und Konfigurieren von Benutzern erstellt haben, müssen als PostgreSQL-Datenbankbenutzer eingerichtet sein und über Berechtigungen verfügen, um sich bei der Datenbank anzumelden. Dazu melden Sie sich als Datenbankbenutzer mit `rds_superuser`-Rechten an. Wenn Sie beispielsweise beim Erstellen Ihres DB-Clusters von Aurora PostgreSQL die Standardeinstellungen akzeptiert haben, verwenden Sie `postgres`, wie in den folgenden Schritten gezeigt.

So erstellen Sie PostgreSQL-Datenbankbenutzer für Ihre Kerberos-Prinzipale

1. Verwenden Sie `psql`, um eine Verbindung mit dem DB-Instance-Endpunkt Ihres DB-Clusters von Aurora PostgreSQL mit `psql` herzustellen. Im folgenden Beispiel wird das `postgres`-Standardkonto für die `rds_superuser`-Rolle verwendet.

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. Erstellen Sie einen Datenbankbenutzernamen für jeden Kerberos-Prinzipal (Active-Directory-Benutzername), der Zugriff auf die Datenbank haben soll. Verwenden Sie den kanonischen Benutzernamen (Identität), wie er in der Active-Directory-Instance definiert ist, d. h. einen `alias` in Kleinbuchstaben (Benutzername in Active Directory) und den Namen der Active-Directory-Domain für diesen Benutzernamen in Großbuchstaben. Der Active-Directory-Benutzername ist ein extern authentifizierter Benutzer. Setzen Sie den Namen daher in Anführungszeichen, wie im Folgenden gezeigt.

```
postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
CREATE ROLE
```

3. Weisen Sie dem Datenbankbenutzer die `rds_ad`-Rolle zu.

```
postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
GRANT ROLE
```

Nachdem Sie alle PostgreSQL-Benutzer für Ihre Active-Directory-Benutzeridentitäten erstellt haben, können Benutzer mit ihren Kerberos-Anmeldeinformationen auf den DB-Cluster von Aurora PostgreSQL zugreifen.

Es ist erforderlich, dass die Datenbankbenutzer, die sich mit Kerberos authentifizieren, dies von Client-Computern aus tun, die Mitglieder der Active-Directory-Domain sind.

Datenbankbenutzer, denen die `rds_ad`-Rolle zugewiesen wurde, können nicht auch über die `rds_iam`-Rolle verfügen. Dies gilt auch für verschachtelte Mitgliedschaften. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung.

Konfigurieren Ihres DB-Clusters von Aurora PostgreSQL für Benutzernamen, bei denen die Groß-/Kleinschreibung unterschieden wird

Die Aurora-PostgreSQL-Versionen 14.5, 13.8, 12.12 und 11.17 unterstützen den PostgreSQL-Parameter `krb_caseins_users`. Dieser Parameter unterstützt Active-Directory-Benutzernamen ohne Berücksichtigung der Groß- und Kleinschreibung. Standardmäßig ist dieser Parameter auf „false“ eingestellt, sodass bei Benutzernamen von Aurora PostgreSQL zwischen Groß- und Kleinschreibung interpretiert wird. Das ist das Standardverhalten in allen älteren Versionen von Aurora PostgreSQL. Sie können diesen Parameter in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe jedoch auf `true` festlegen und Ihrem DB-Cluster von Aurora PostgreSQL erlauben, Benutzernamen ohne Berücksichtigung von Groß- und Kleinschreibung zu interpretieren. Erwägen Sie, diese Einstellung als Annehmlichkeit für Ihre Datenbankbenutzer vorzunehmen, die bei der Authentifizierung mit Active Directory manchmal die Groß- und Kleinschreibung ihres Benutzernamens falsch eingeben.

Um den `krb_caseins_users`-Parameter zu ändern, muss Ihr DB-Cluster von Aurora PostgreSQL eine benutzerdefinierte DB-Cluster-Parametergruppe verwenden. Informationen über das Arbeiten mit einer benutzerdefinierten DB-Cluster-Parametergruppe finden Sie unter Parametergruppen für Amazon Aurora.

Sie können das AWS CLI oder das verwenden AWS-Managementkonsole , um die Einstellung zu ändern. Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

Schritt 8: Konfigurieren eines PostgreSQL-Clients

Gehen Sie folgendermaßen vor, um einen PostgreSQL-Client zu konfigurieren:

- Erstellen Sie eine krb5.conf-Datei (oder eine vergleichbare Datei), um auf die Domäne zu verweisen.
- Stellen Sie sicher, dass der Datenverkehr zwischen dem Client-Host und fließen kann Directory Service. Verwenden Sie ein Netzwerk-Dienstprogramm wie Netcat für die folgenden Aufgaben:
  - Überprüfen Sie den Datenverkehr über DNS für Port 53.
  - Überprüfen Sie den Datenverkehr TCP/UDP für Port 53 und für Kerberos, einschließlich der Ports 88 und 464 für. Directory Service
- Stellen Sie sicher, dass der Datenverkehr zwischen dem Client-Host und der DB-Instance über den Datenbank-Port fließen kann. Verwenden Sie beispielsweise psql, um eine Verbindung herzustellen und auf die Datenbank zuzugreifen.

Im Folgenden finden Sie einen krb5.conf-Beispielinhalt für. AWS Managed Microsoft AD

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

Nachfolgend ein Beispiel für den Inhalt von krb5.conf für ein lokales Microsoft Active Directory.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
```

```
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

Verwalten eines DB-Clusters von Aurora PostgreSQL in einer Active Directory-Domain

Sie können die Konsole, die CLI oder die RDS-API verwenden, um Ihren DB-Cluster und ihre Beziehung zu Ihrem Microsoft Active Directory zu verwalten. Sie können z. B. ein Active Directory zuordnen, um die Kerberos-Authentifizierung zu aktivieren. Sie können auch die Zuordnung für ein Active Directory entfernen, um die Kerberos-Authentifizierung zu deaktivieren. Sie können auch DB-Cluster verschieben, die von einem Microsoft Active Directory zu einem anderen extern authentifiziert werden.

Sie können z. B. mithilfe der CLI Folgendes tun:

- Um die Aktivierung der Kerberos-Authentifizierung für eine fehlgeschlagene Mitgliedschaft erneut zu versuchen, verwenden Sie den CLI-Befehl modify-db-cluster. Geben Sie die Verzeichnis-ID der aktuellen Mitgliedschaft für die Option `--domain` an.
- Um die Kerberos-Authentifizierung für eine DB-Instance zu deaktivieren, verwenden Sie den CLI-Befehl modify-db-cluster. Geben Sie `none` für die Option `--domain` an.
- Um eine DB-Instance von einer Domäne in eine andere zu verschieben, verwenden Sie den CLI-Befehl modify-db-cluster. Geben Sie die Domänen-ID der neuen Domäne für die Option `--domain` an.

Grundlegendes zur Domänenmitgliedschaft

Nachdem Sie Ihren DB-Cluster erstellt oder geändert haben, werden die DB-Instances zu Mitgliedern der Domäne. Sie können den Status der Domänenmitgliedschaft in der Konsole anzeigen oder den CLI-Befehl describe-db-instances ausführen. Der Status der DB-Instance kann einer der folgenden sein:

- `kerberos-enabled` – Für die DB-Instance ist die Kerberos-Authentifizierung aktiviert.
- `enabling-kerberos` – AWS ist dabei, die Kerberos-Authentifizierung auf dieser DB-Instance zu aktivieren.
- `pending-enable-kerberos` – Das Aktivieren der Kerberos-Authentifizierung ist für diese DB-Instance ausstehend.
- `pending-maintenance-enable-kerberos` – AWS versucht, die Kerberos-Authentifizierung auf der DB-Instance während des nächsten geplanten Wartungsfensters zu aktivieren.
- `pending-disable-kerberos` – Das Deaktivieren der Kerberos-Authentifizierung ist für diese DB-Instance ausstehend.
- `pending-maintenance-disable-kerberos` – AWS versucht, die Kerberos-Authentifizierung auf der DB-Instance während des nächsten geplanten Wartungsfensters zu deaktivieren.
- `enable-kerberos-failed` – Ein Konfigurationsproblem verhinderte, dass AWS die Kerberos-Authentifizierung auf der DB-Instance aktivierte. Beheben Sie das Konfigurationsproblem, bevor Sie den Befehl zum Ändern der DB-Instance erneut ausgeben.
- `disabling-kerberos` – AWS ist dabei, die Kerberos-Authentifizierung auf dieser DB-Instance zu deaktivieren.

Eine Anfrage zur Aktivierung der Kerberos-Authentifizierung kann wegen eines Netzwerkverbindungsproblems oder einer falschen IAM-Rolle fehlschlagen. In einigen Fällen kann der Versuch, die Kerberos-Authentifizierung zu aktivieren, fehlschlagen, wenn Sie einen DB-Cluster erstellen oder ändern. Wenn dies passiert, stellen Sie sicher, dass Sie die richtige IAM-Rolle verwenden, und ändern Sie dann den DB-Cluster, um der Domäne beizutreten.

Herstellen einer Verbindung zu PostgreSQL mit Kerberos-Authentifizierung

Sie können sich über die pgAdmin-Schnittstelle oder über eine Befehlszeilenschnittstelle wie psql per Kerberos-Authentifizierung mit PostgreSQL verbinden. Weitere Informationen zum Herstellen von Verbindungen finden Sie unter Herstellen einer Verbindung mit einem Amazon-Aurora-PostgreSQL-

DB-Cluster. Informationen zum Abrufen des Endpunkts, der Portnummer und anderer Details, die für die Verbindung benötigt werden, finden Sie unter Anzeigen der Endpunkte für einen Aurora-Cluster.

> **Note**
>
> GSSAPI-Authentifizierung und Verschlüsselung in PostgreSQL werden von der Kerberos-Bibliothek `libkrb5.so` implementiert. Features wie `postgres_fdw` und `dblink` beruhen ebenfalls auf dieser Bibliothek für ausgehende Verbindungen mit Kerberos-Authentifizierung oder -Verschlüsselung.

pgAdmin

Um pgAdmin für die Verbindung zu PostgreSQL mit Kerberos-Authentifizierung zu verwenden, führen Sie die folgenden Schritte aus:

1. Starten Sie die Anwendung pgAdmin auf Ihrem Client-Computer.
2. Klicken Sie auf der Registerkarte Dashboard auf Add New Server (Neuen Server hinzufügen).
3. Geben Sie im Dialogfeld (Erstellen – Server) auf der Registerkarte Allgemein einen Namen für den Server in pgAdmin ein.
4. Geben Sie auf der Registerkarte Connection (Verbindung) die folgenden Informationen aus der Datenbank von Aurora PostgreSQL ein:
   - Geben Sie für Host den Endpunkt für die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL an. Ein Endpunkt sieht in etwa wie folgt aus:

     ```
     AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
     ```

     Wenn Sie eine Verbindung mit einem lokalen Microsoft Active Directory von einem Windows-Client aus herstellen möchten, verwenden Sie den Domänennamen des von AWS verwalteten Active Directorys statt `rds.amazonaws.com` im Host-Endpunkt. Angenommen, der Domänenname für das AWS Managed Active Directory lautet `corp.example.com`. Für Host würde der Endpunkt wie folgt angegeben werden:

     ```
     AUR-cluster-instance.111122223333.aws-region.corp.example.com
     ```

   - Geben Sie unter Port den zugewiesenen Port ein.

- Geben Sie unter Wartungsdatenbank den Namen der initialen Datenbank ein, mit der sich der Client verbinden soll.
- Geben Sie unter Benutzername den Benutzernamen ein, den Sie für die Kerberos-Authentifizierung in Schritt 7: Erstellen von PostgreSQL-Benutzern für Ihre Kerberos-Prinzipale eingegeben haben.

5. Wählen Sie Speichern.

Psql

Um psql für die Verbindung mit PostgreSQL mit Kerberos-Authentifizierung zu verwenden, führen Sie die folgenden Schritte aus:

1. Führen Sie an einer Eingabeaufforderung den folgenden Befehl aus.

   ```
   kinit username
   ```

   Ersetzen Sie *username* durch den Benutzernamen. Geben Sie in der Eingabeaufforderung das im Microsoft Active Directory für den Benutzer gespeicherte Passwort ein.

2. Wenn der PostgreSQL-DB-Cluster eine öffentlich zugängliche VPC verwendet, geben Sie eine IP-Adresse für Ihren DB-Cluster-Endpunkt in Ihrer `/etc/hosts`-Datei auf dem EC2-Client ein. Die folgenden Befehle rufen beispielsweise die IP-Adresse ab und fügen sie dann in die `/etc/hosts`-Datei ein.

   ```
   % dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
   ;; Truncated, retrying in TCP mode.
   ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
   34.210.197.118

   % echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
   hosts
   ```

   Wenn Sie eine lokale Microsoft Active Directory von einem Windows-Client verwenden, müssen Sie eine Verbindung über einen speziellen Endpunkt herstellen. Anstatt die Amazon-Domäne `rds.amazonaws.com` im Host-Endpunkt zu verwenden, verwenden Sie den Domänennamen des AWS-Managed Active Directory.

Angenommen, der Domänenname für IhrAWS Managed Active Directory lautet `corp.example.com`. Verwenden Sie dann das Format *`PostgreSQL-endpoint.AWS-Region.corp.example.com`* für den Endpunkt und legen Sie es in der `/etc/hosts`-Datei ab.

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
hosts
```

3. Verwenden Sie den folgenden psql-Befehl, um sich bei einem PostgreSQL-DB-Cluster anzumelden, der/die in Active Directory integriert ist. Verwenden Sie einen Cluster oder Instance-Endpunkt.

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.rds.amazonaws.com postgres
```

Um sich beim PostgreSQL DB-Cluster von einem Windows-Client aus unter Verwendung eines lokalen Active Directory anzumelden, verwenden Sie den folgenden psql-Befehl mit dem Domänennamen aus dem vorhergehenden Schritt (`corp.example.com`):

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

Verwenden von AD-Sicherheitsgruppen für die Aurora-PostgreSQL-Zugriffskontrolle

Ab den Aurora-PostgreSQL-Versionen 14.10 und 15.5 kann die Aurora-PostgreSQL-Zugriffskontrolle über Sicherheitsgruppen von AWS Directory Service für Microsoft Active Directory (AD) erfolgen. Frühere Versionen von Aurora PostgreSQL unterstützen die Kerberos-basierte Authentifizierung mittels AD nur für einzelne Benutzer. Jeder AD-Benutzer musste explizit für den DB-Cluster bereitgestellt werden, um Zugriff zu erhalten.

Anstatt jeden AD-Benutzer je nach Geschäftsanforderungen explizit für den DB-Cluster bereitzustellen, können Sie AD-Sicherheitsgruppen wie unten beschrieben nutzen:

- AD-Benutzer sind Mitglieder verschiedener AD-Sicherheitsgruppen in einem Active Directory. Diese werden nicht vom DB-Clusteradministrator vorgegeben, sondern basieren auf Geschäftsanforderungen und werden von einem AD-Administrator verwaltet.
- DB-Cluster-Administratoren erstellen DB-Rollen in DB-Instances auf der Grundlage von Geschäftsanforderungen. Diese DB-Rollen können unterschiedliche Berechtigungen haben.
- DB-Cluster-Administratoren konfigurieren eine Zuordnung von AD-Sicherheitsgruppen zu DB-Rollen auf DB-Cluster-Basis.
- DB-Benutzer können mit ihren AD-Anmeldeinformationen auf DB-Cluster zugreifen. Der Zugriff richtet sich nach der Mitgliedschaft in einer AD-Sicherheitsgruppe. AD-Benutzer erhalten oder verlieren automatisch Zugriff auf der Grundlage ihrer AD-Gruppenmitgliedschaft.

Voraussetzungen

Vor dem Einrichten der Erweiterung für AD-Sicherheitsgruppen muss Folgendes durchgeführt worden sein:

- Einrichten einer Kerberos-Authentifizierung für PostgreSQL-DB-Cluster. Weitere Informationen finden Sie unter Einrichtung der Kerberos-Authentifizierung für PostgreSQL-DB-Cluster.

> **Note**
>
> Überspringen Sie für AD-Sicherheitsgruppen Schritt 7: Erstellen von PostgreSQL-Benutzern für Kerberos-Prinzipale in diesem Einrichtungsverfahren.

> **Important**
>
> Wenn Sie AD-Sicherheitsgruppen in einem Aurora-PostgreSQL-Cluster aktivieren, für den die Kerberos-Authentifizierung bereits aktiviert ist, können Authentifizierungsprobleme auftreten. Dies passiert, wenn Sie `pg_ad_mapping` zum Parameter `shared_preload_libraries` hinzufügen und Ihre Datenbank neu starten. Bei der Verwendung von Cluster-Endpunkten können Anmeldeversuche mit einem AD-Benutzer fehlschlagen, der kein Datenbankbenutzer ist, der die Rolle `rds_ad` innehat. Dies kann möglicherweise auch zu Abstürzen der Engine führen. Zur Behebung des Problems deaktivieren Sie die Kerberos-Authentifizierung in Ihrem Cluster und aktivieren Sie sie anschließend erneut. Diese Problemumgehung ist für vorhandene Instances erforderlich, wirkt sich jedoch nicht auf solche aus, die nach April 2025 erstellt wurden.

- Verwalten eines DB-Clusters in einer Domain. Weitere Informationen finden Sie unter Verwalten eines DB-Clusters in einer Domain.

Einrichten der pg_ad_mapping-Erweiterung

Aurora PostgreSQL unterstützt jetzt eine `pg_ad_mapping`-Erweiterung zur Verwaltung der Zuordnung zwischen AD-Sicherheitsgruppen und DB-Rollen im Aurora PostgreSQL-Cluster. Weitere Informationen zu den Funktionen, die von `pg_ad_mapping` bereitgestellt werden, finden Sie unter Verwenden von Funktionen der `pg_ad_mapping`-Erweiterung.

Wenn Sie die `pg_ad_mapping`-Erweiterung in Ihrem DB-Cluster von Aurora PostgreSQL bereitstellen möchten, fügen Sie zunächst `pg_ad_mapping` zu den freigegebenen Bibliotheken in der benutzerdefinierten DB-Parametergruppe für Ihren DB-Cluster von Aurora PostgreSQL hinzu. Weitere Informationen über das Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe finden Sie unter Parametergruppen für Amazon Aurora. Als Nächstes installieren Sie die `pg_ad_mapping`-Erweiterung. Die Schritte in diesem Abschnitt veranschaulichen die Vorgehensweise. Sie können die AWS-Managementkonsole oder die AWS CLI verwenden.

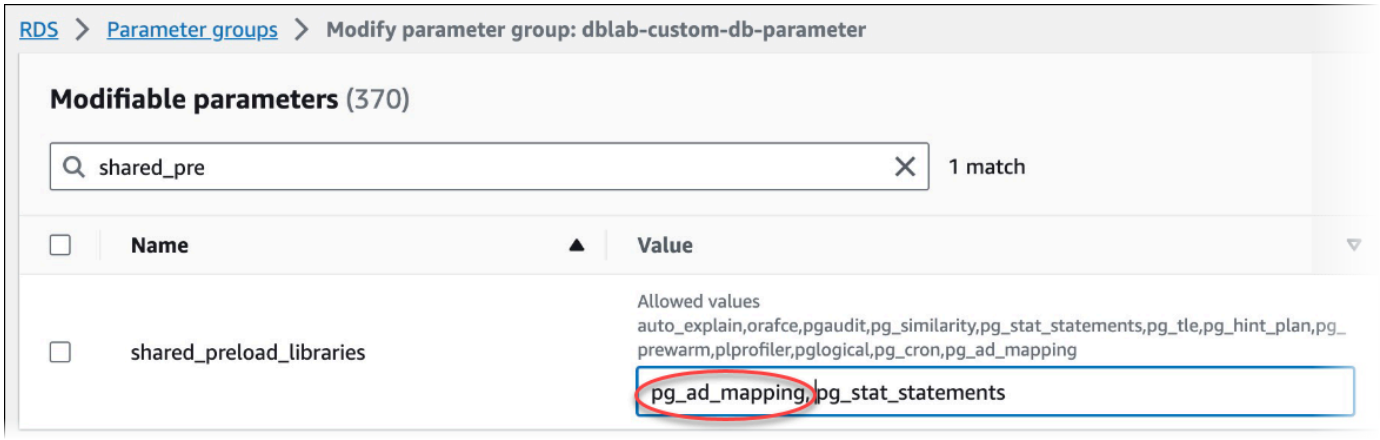
Sie müssen über Berechtigungen als `rds_superuser`-Rolle verfügen, um alle diese Aufgaben ausführen zu können.

Bei den folgenden Schritten wird davon ausgegangen, dass Ihr DB-Cluster von Aurora PostgreSQL einer benutzerdefinierten DB-Cluster-Parametergruppe zugeordnet ist.

Konsole

So richten Sie die **`pg_ad_mapping`**-Erweiterung ein

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Writer-Instance des DB-Clusters von Aurora PostgreSQL aus.
3. Öffnen Sie die Registerkarte Konfiguration für Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL. Suchen Sie in den Instance-Details den Link Parameter group (Parametergruppe).
4. Wählen Sie den Link aus, um die benutzerdefinierten Parameter zu öffnen, die Ihrem DB-Cluster von Aurora PostgreSQL zugeordnet sind.
5. Geben Sie in das Suchfeld Parameters (Parameter) `shared_pre` ein, um den `shared_preload_libraries`-Parameter zu finden.
6. Wählen Sie Edit parameters (Parameter bearbeiten) aus, um auf die Eigenschaftswerte zuzugreifen.
7. Fügen Sie `pg_ad_mapping` der Liste im Feld Values (Werte) hinzu. Verwenden Sie ein Komma, um Elemente in der Werteliste zu trennen.



8. Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu, damit Ihre Änderung des `shared_preload_libraries`-Parameters wirksam wird.
9. Wenn die Instance verfügbar ist, überprüfen Sie, ob `pg_ad_mapping` initialisiert wurde. Stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her und führen Sie den folgenden Befehl aus.

```
SHOW shared_preload_libraries;
shared_preload_libraries
```

```
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

10. Sobald pg_ad_mapping initialisiert ist, können Sie die Erweiterung erstellen. Sie müssen die Erweiterung nach dem Initialisieren der Bibliothek erstellen, um die von der Erweiterung bereitgestellten Funktionen verwenden zu können.

```
CREATE EXTENSION pg_ad_mapping;
```

11. Schließen Sie die psql-Sitzung.

```
labdb=> \q
```

AWS CLI

So richten Sie pg_ad_mapping ein

Wenn Sie pg_ad_mapping mit der AWS CLI einrichten möchten, rufen Sie die Operation modify-db-parameter-group auf, um diesen Parameter zu Ihrer benutzerdefinierten Parametergruppe hinzuzufügen, so wie im folgenden Verfahren gezeigt.

1. Verwenden Sie den folgenden AWS CLI-Befehl, um dem shared_preload_libraries-Parameter pg_ad_mapping hinzuzufügen.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_ad_mapping,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Verwenden Sie den folgenden AWS CLI-Befehl, um die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu zu starten, sodass pg_ad_mapping initialisiert wird.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Wenn die Instance verfügbar ist, können Sie überprüfen, ob `pg_ad_mapping` initialisiert wurde. Stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her und führen Sie den folgenden Befehl aus.

```
SHOW shared_preload_libraries;
shared_preload_libraries
-------------------------
rdsutils,pg_ad_mapping
(1 row)
```

   Wenn pg_ad_mapping initialisiert ist, können Sie die Erweiterung erstellen.

```
CREATE EXTENSION pg_ad_mapping;
```

4. Schließen Sie die `psql`-Sitzung, damit Sie die AWS CLI verwenden können.

```
labdb=> \q
```

Abrufen der SID der Active-Directory-Gruppe in PowerShell

Eine Sicherheits-ID (SID) wird verwendet, um einen Sicherheitsprinzipal oder eine Sicherheitsgruppe eindeutig zu identifizieren. Immer wenn eine Sicherheitsgruppe oder ein Konto in Active Directory erstellt wird, wird dieser bzw. diesem eine SID zugewiesen. Zum Abrufen der SID der AD-Sicherheitsgruppe aus Active Directory können Sie das Get-ADGroup-Cmdlet vom Windows-Client-Computer verwenden, der mit dieser Active-Directory-Domain verbunden ist. Der Identity-Parameter gibt den Active-Directory-Gruppennamen zum Abrufen der entsprechenden SID an.

Im folgenden Beispiel wird die SID der AD-Gruppe *adgroup1* zurückgegeben.

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID

             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

Zuordnen der DB-Rolle zur AD-Sicherheitsgruppe

Sie müssen die AD-Sicherheitsgruppen in der Datenbank explizit als PostgreSQL-DB-Rolle bereitstellen. Ein AD-Benutzer, der Teil mindestens einer bereitgestellten AD-Sicherheitsgruppe ist, erhält Zugriff auf die Datenbank. Sie sollten der auf AD-Gruppensicherheit basierenden DB-Rolle nicht die Rolle `rds_ad role` zuweisen. Die Kerberos-Authentifizierung für die Sicherheitsgruppe wird ausgelöst, indem das Suffix eines Domainnamens wie *`user1@example.com`* verwendet wird. Diese DB-Rolle unterstützt keine Passwort- oder IAM-Authentifizierung für den Zugriff auf die Datenbank.

> **Note**
>
> AD-Benutzer, die über eine entsprechende DB-Rolle in der Datenbank mit der ihnen zugewiesenen `rds_ad`-Rolle verfügen, können sich nicht als Teil der AD-Sicherheitsgruppe anmelden. Sie erhalten Zugriff über die DB-Rolle als Einzelbenutzer.

So ist accounts-group beispielsweise eine Sicherheitsgruppe in AD, für die Sie diese Sicherheitsgruppe in Aurora PostgreSQL als accounts-role bereitstellen können.

AD-Sicherheitsgruppe	PostgreSQL-DB-Rolle
accounts-group	accounts-role

Wenn Sie die DB-Rolle der AD-Sicherheitsgruppe zuordnen, müssen Sie sicherstellen, dass für die DB-Rolle das LOGIN-Attribut festgelegt ist und dass sie über die CONNECT-Berechtigungen für die erforderliche Anmeldedatenbank verfügt.

```
postgres => alter role accounts-role login;

ALTER ROLE
postgres => grant connect on database accounts-db to accounts-role;
```

Der Administrator kann nun mit der Erstellung der Zuordnung zwischen der AD-Sicherheitsgruppe und der PostgreSQL-DB-Rolle fortfahren.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', <SID>, <Weight>);
```

Informationen zum Abrufen der SID der AD-Sicherheitsgruppe finden Sie unter Abrufen der SID der Active-Directory-Gruppe in PowerShell.

Es ist möglich, dass ein AD-Benutzer mehreren Gruppen angehört. In diesem Fall erbt der AD-Benutzer die Berechtigungen der DB-Rolle, da diese mit der höchsten Gewichtung bereitgestellt wurde. Ist die Gewichtung der beiden Rollen identisch, erbt der AD-Benutzer die Berechtigungen der DB-Rolle, die der kürzlich hinzugefügten Zuordnung entspricht. Es wird empfohlen, Gewichtungen anzugeben, die den relativen Berechtigungen der einzelnen DB-Rollen entsprechen. Je höher die Berechtigung einer DB-Rolle, desto höher die Gewichtung, die dem Zuordnungseintrag zugewiesen werden sollte. Dadurch wird eine Mehrdeutigkeit vermieden, da es keine zwei Zuordnungen mit identischer Gewichtung gibt.

Die folgende Tabelle zeigt ein Beispiel einer Zuordnung von AD-Sicherheitsgruppen zu DB-Rollen von Aurora PostgreSQL.

AD-Sicherheitsgruppe	PostgreSQL-DB-Rolle	Gewicht
accounts-group	accounts-role	7
sales-group	sales-role	10
dev-group	dev-role	7

Im folgenden Beispiel erbt `user1` die Rechte von sales-role aufgrund der höheren Gewichtung. `user2` erbt die Berechtigungen von `dev-role`, da die Zuordnung dieser Rolle nach `accounts-role` erstellt wurde, welche dieselbe Gewichtung haben wie `accounts-role`.

Username	Sicherheitsgruppenmitgliedschaft
Benutzer 1	accounts-group sales-group
user2	accounts-group dev-group

Die psql-Befehle zum Erstellen, Auflisten und Löschen der Zuordnungen sind unten aufgeführt. Derzeit ist es nicht möglich, einen einzelnen Zuordnungseintrag zu ändern. Der bestehende Eintrag muss gelöscht und die Zuordnung neu erstellt werden.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', 'S-1-5-67-890',
 7);
admin=>select pgadmap_set_mapping('sales-group', 'sales-role', 'S-1-2-34-560', 10);
admin=>select pgadmap_set_mapping('dev-group', 'dev-role', 'S-1-8-43-612', 7);

admin=>select * from pgadmap_read_mapping();

ad_sid       | pg_role        | weight | ad_grp
-------------+----------------+--------+---------------
S-1-5-67-890 | accounts-role | 7      | accounts-group
S-1-2-34-560 | sales-role    | 10     | sales-group
S-1-8-43-612 | dev-role      | 7      | dev-group
(3 rows)
```

Protokollierung/Prüfung der AD-Benutzeridentität

Verwenden Sie den folgenden Befehl, um die Datenbankrolle zu ermitteln, die vom aktuellen oder vom Sitzungsbenutzer geerbt wurde:

```
postgres=>select session_user, current_user;

session_user | current_user
-------------+--------------
dev-role     | dev-role

(1 row)
```

Verwenden Sie den folgenden Befehl, um die Identität des AD-Sicherheitsprinzipals zu ermitteln:

```
postgres=>select principal from pg_stat_gssapi where pid = pg_backend_pid();

 principal
-------------------------
 user1@example.com

(1 row)
```

Derzeit ist die AD-Benutzeridentität in den Prüfungsprotokollen nicht sichtbar. Der `log_connections`-Parameter kann aktiviert werden, um die Einrichtung einer DB-Sitzung zu protokollieren. Weitere Informationen finden Sie unter log_connections. Die Ausgabe hierfür beinhaltet die AD-Benutzeridentität, wie unten dargestellt. Die dieser Ausgabe zugeordnete Backend-PID kann dann dabei helfen, Aktionen dem tatsächlichen AD-Benutzer zuzuordnen.

```
pgrole1@postgres:[615]:LOG: connection authorized: user=pgrole1
 database=postgres application_name=psql GSS (authenticated=yes, encrypted=yes,
 principal=Admin@EXAMPLE.COM)
```

Einschränkungen

- Microsoft Entra ID, auch bekannt als Azure Active Directory, wird nicht unterstützt.

Verwenden von Funktionen der **`pg_ad_mapping`**-Erweiterung

Die `pg_ad_mapping`-Erweiterung unterstützte die folgenden Funktionen:

pgadmap_set_mapping

Diese Funktion stellt die Zuordnung zwischen der AD-Sicherheitsgruppe und der Datenbankrolle mit einer zugehörigen Gewichtung her.

Syntax

```
pgadmap_set_mapping(
ad_group,
db_role,
ad_group_sid,
weight)
```

Argumente

Parameter	Beschreibung
ad_group	Der Name der AD-Gruppe Der Wert kann weder Null noch eine leere Zeichenfolge sein.

Parameter	Beschreibung
db_role	Datenbankrolle, die der angegebenen AD-Gruppe zugeordnet werden soll Der Wert kann weder Null noch eine leere Zeichenfolge sein.
ad_group_sid	Die Sicherheits-ID, die zur eindeutigen Identifizierung der AD-Gruppe verwendet wird Der Wert beginnt mit „S-1-“ und darf weder Null noch eine leere Zeichenfolge sein. Weitere Informationen finden Sie unter Abrufen der SID der Active-Directory-Gruppe in PowerShell.
Gewichtung	Gewichtung, die der Datenbankrolle zugewiesen ist Die Rolle mit der höchsten Gewichtung hat Vorrang, wenn der Benutzer Mitglied mehrerer Gruppen ist. Der Standardwert für die Gewichtung ist 1.

Rückgabetyp

None

Nutzungshinweise

Diese Funktion fügt eine neue Zuordnung von der AD-Sicherheitsgruppe zur Datenbankrolle hinzu. Sie kann nur von einem Benutzer mit rds_superuser privilege-Berechtigung auf der primären DB-Instance des DB-Clusters ausgeführt werden.

Beispiele

```
postgres=> select pgadmap_set_mapping('accounts-group','accounts-
role','S-1-2-33-12345-67890-12345-678',10);

pgadmap_set_mapping

(1 row)
```

pgadmap_read_mapping

Diese Funktion listet die Zuordnungen zwischen der AD-Sicherheitsgruppe und der DB-Rolle auf, die mithilfe der Funktion `pgadmap_set_mapping` festgelegt wurde.

Syntax

```
pgadmap_read_mapping()
```

Argumente

None

Rückgabetyp

Parameter	Beschreibung
ad_group_sid	Die Sicherheits-ID, die zur eindeutigen Identifizierung der AD-Gruppe verwendet wird. Der Wert beginnt mit „S-1-“ und darf weder Null noch eine leere Zeichenfolge sein. Weitere Informationen dazu finden Sie unter Abrufen der SID der Active-Directory-Gruppe in PowerShell.accounts-role@example.com.
db_role	Datenbankrolle, die der angegebenen AD-Gruppe zugeordnet werden soll Der Wert kann weder Null noch eine leere Zeichenfolge sein.
Gewichtung	Gewichtung, die der Datenbankrolle zugewiesen ist. Die Rolle mit der höchsten Gewichtung hat Vorrang, wenn der Benutzer Mitglied mehrerer Gruppen ist. Der Standardwert für die Gewichtung ist 1.
ad_group	Der Name der AD-Gruppe Der Wert kann weder Null noch eine leere Zeichenfolge sein.

Nutzungshinweise

Rufen Sie diese Funktion auf, um alle verfügbaren Zuordnungen zwischen der AD-Sicherheitsgruppe und der DB-Rolle aufzulisten.

Beispiele

```
postgres=> select * from pgadmap_read_mapping();
```

```
ad_sid                              | pg_role       | weight | ad_grp
------------------------------------+---------------+--------+------------------
S-1-2-33-12345-67890-12345-678      | accounts-role | 10     | accounts-group
(1 row)

(1 row)
```

pgadmap_reset_mapping

Diese Funktion setzt eine oder alle Zuordnungen zurück, die mit der Funktion `pgadmap_set_mapping` festgelegt wurden.

Syntax

```
pgadmap_reset_mapping(
ad_group_sid,
db_role,
weight)
```

Argumente

Parameter	Beschreibung
ad_group_sid	Die Sicherheits-ID, die zur eindeutigen Identifizierung der AD-Gruppe verwendet wird
db_role	Datenbankrolle, die der angegebenen AD-Gruppe zugeordnet werden soll
Gewichtung	Gewichtung, die der Datenbankrolle zugewiesen ist

Wenn keine Argumente angegeben werden, werden alle Zuordnungen von AD-Gruppen zu DB-Rollen zurückgesetzt. Es müssen entweder alle Argumente angegeben werden oder keine.

Rückgabetyp

None

Nutzungshinweise

Rufen Sie diese Funktion auf, um eine bestimmte Zuordnung einer AD-Gruppe zu einer DB-Rolle zu löschen oder um alle Zuordnungen zurückzusetzen. Diese Funktion kann nur von einem Benutzer mit `rds_superuser`-Berechtigung auf der primären DB-Instance des DB-Clusters ausgeführt werden.

Beispiele

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+-------------------
 S-1-2-33-12345-67890-12345-678 | accounts-role| 10          | accounts-group
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(2 rows)
postgres=> select pgadmap_reset_mapping('S-1-2-33-12345-67890-12345-678', 'accounts-
role', 10);

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+---------------
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(1 row)
postgres=> select pgadmap_reset_mapping();

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+--------------
 (0 rows)
```

Migrieren von Daten nach Amazon Aurora mit PostgreSQL-Kompatibilität

Sie haben mehrere Möglichkeiten, Daten aus einer vorhandenen Datenbank in einen Amazon Aurora PostgreSQL-kompatible Edition-DB-Cluster zu migrieren. Die verfügbaren Migrationsoptionen sind auch von der Quelldatenbank und der Menge der zu migrierenden Daten abhängig. Folgende Optionen sind verfügbar:

Migrieren einer RDS-for-PostgreSQL-DB-Instance unter Verwendung eines Snapshots

Sie können Daten direkt von einem RDS-for-PostgreSQL-DB-Snapshot in einen Aurora-PostgreSQL-DB-Cluster migrieren.

Migrieren einer RDS-for-PostgreSQL-DB-Instance unter Verwendung eines Aurora-Lesereplikats

Sie können auch Daten aus einer RDS-for-PostgreSQL-DB-Instance migrieren, indem Sie ein Aurora-PostgreSQL-Lesereplikat einer RDS-for-PostgreSQL-DB-Instance erstellen. Wenn die Replikat-Verzögerung zwischen der RDS-for-PostgreSQL-DB-Instance und dem Aurora-PostgreSQL-Lesereplikat Null ist, können Sie die Replikation stoppen. An diesem Punkt können Sie die Aurora-Read Replica zu einem eigenständigen Aurora PostgreSQL-DB-Cluster für das Lesen und Schreiben machen.

Importieren von Daten aus Amazon S3 in Aurora PostgreSQL

Sie können Daten migrieren, indem Sie sie Amazon S3 in eine Tabelle importieren, die zu einem Aurora PostgreSQL-DB-Cluster gehört.

Migrieren aus einer Datenbank, die nicht PostgreSQL-kompatibel ist

Sie können AWS Database Migration Service (AWS DMS) auch verwenden, um Daten aus einer Datenbank zu migrieren, die nicht PostgreSQL-kompatibel ist. Weitere Informationen zu AWS DMS finden Sie unter Was ist AWS-Database Migration Service? im AWS Database Migration Service-Benutzerhandbuch.

> **Note**
>
> Die Aktivierung der Kerberos-Authentifizierung wird derzeit auf dem DB-Cluster von Aurora PostgreSQL während der Migration von RDS für PostgreSQL nicht unterstützt. Sie können die Kerberos-Authentifizierung nur auf einem eigenständigen DB-Cluster von Aurora PostgreSQL aktivieren.

Eine Liste der AWS-Regionen, in denen Aurora erhältlich ist, finden Sie unter Amazon Aurora in der Allgemeine AWS-Referenz.

> **Important**
>
> Wenn Sie planen, eine RDS für PostgreSQL-DB-Instance in naher Zukunft auf einen Aurora PostgreSQL-DB-Cluster zu migrieren, empfehlen wir Ihnen dringend, Upgrades von automatischen Unterversionen für die DB-Instance zu Beginn der Migrationsplanungsphase zu deaktivieren. Die Migration nach Aurora PostgreSQL kann sich verzögern, wenn die RDS für PostgreSQL-Version von Aurora PostgreSQL noch nicht unterstützt wird. Informationen zu Aurora PostgreSQL-Versionen finden Sie unter Engine-Versionen für Amazon Aurora-PostgreSQL.

Migrieren eines Snapshots einer RDS-for-PostgreSQL-DB-Instance in einen Aurora-PostgreSQL-DB-Cluster

Sie können einen DB-Snapshot einer RDS-PostgreSQL-DB-Instance migrieren und so einen Aurora-for-PostgreSQL-DB-Cluster erstellen. Der neue Aurora-PostgreSQL-DB-Cluster wird mit den Daten der ursprünglichen RDS-for-PostgreSQL-DB-Instance gefüllt. Informationen zum Erstellen von DB-Snapshots finden Sie unter Erstellen eines DB-Snapshots.

In einigen Fällen befindet sich der DB-Snapshot möglicherweise nicht in der AWS-Region, in der Sie Ihre Daten speichern möchten. Wenn ja, verwenden Sie die Amazon-RDS-Konsole zum Kopieren des DB-Snapshots in diese AWS-Region. Informationen zum Kopieren von DB-Snapshots finden Sie unter Kopieren eines DB-Snapshots.

Sie können RDS-for-PostgreSQL-Snapshots migrieren, die mit den in der angegebenen AWS-Region verfügbaren Aurora-PostgreSQL-Versionen kompatibel sind. Beispielsweise können Sie ein Snapshot von einer DB-Instance mit RDS für PostgreSQL 11.1 auf Aurora PostgreSQL Version 11.4, 11.7, 11.8 oder 11.9 in Region USA West (Nordkalifornien) migrieren. Sie können RDS für PostgreSQL 10.11 Snapshot zu Aurora PostgreSQL 10.11, 10.12, 10.13 und 10.14 migrieren. Mit anderen Worten, der RDS-for-PostgreSQL-Snapshot muss im Vergleich zu Aurora PostgreSQL die gleiche oder eine niedrigere Unterversion verwenden.

Sie können auch wählen, dass Ihr neuer Aurora PostgreSQL DB-Cluster im Ruhezustand verschlüsselt wird, indem Sie eine AWS KMS key. Diese Option steht nur für unverschlüsselte DB-Snapshots zu Verfügung.

Um einen RDS-for-PostgreSQL-DB-Snapshot in einen Aurora-PostgreSQL-DB-Cluster zu migrieren, können Sie die AWS-Managementkonsole, AWS CLI oder RDS API verwenden. Wenn Sie die AWS-Managementkonsole verwenden, nimmt die Konsole die erforderlichen Aktionen vor, um den DB-Cluster und die primäre Instance zu erstellen.

Konsole

So migrieren Sie einen PostgreSQL-DB-Snapshot mithilfe der RDS-Konsole

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Klicken Sie auf Snapshots (Snapshots).
3. Wählen Sie auf der Seite Snapshots den RDS-for-PostgreSQL-Snapshot aus, den Sie zu einem Aurora-PostgreSQL-DB-Cluster migrieren möchten.
4. Wählen Sie Actions (Aktionen) und wählen Sie Migrate snapshot (Snapshot migrieren).
5. Legen Sie auf der Seite Migrate Database (Datenbank migrieren) Folgendes fest:

   - DB-Engine-Version: Wählen Sie eine DB-Engine-Version, die Sie für die neue migrierte Instance verwenden möchten.
   - DB instance identifier (DB-Instance-Kennung): Geben Sie einen Namen für den DB-Cluster ein, der für Ihr Konto in der ausgewählten AWS-Region eindeutig ist. Dieser Bezeichner wird in den Endpunktadressen für die Instances im DB-Cluster verwendet. Sie können den Namen aussagekräftig gestalten, indem Sie darin die gewählte AWS-Region und DB-Engine angeben (z. B. **`aurora-cluster1`**).

     Für den DB-Instance-Bezeichner gelten folgende Einschränkungen:

     - Er muss zwischen 1 und 63 alphanumerische Zeichen oder Bindestriche enthalten.
     - Das erste Zeichen muss ein Buchstabe sein.
     - Darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.
     - Es muss für alle DB-Instances pro AWS-Konto und AWS-Region eindeutig sein.
   - DB-Instance-Klasse: Wählen Sie eine DB-Instance-Klasse, die über ausreichend Speicher und Kapazität für Ihre Datenbank verfügt, z. B. `db.r6g.large`. Aurora-Cluster-Volumes nehmen automatisch an Größe zu, wenn die Datenmenge in Ihrer Datenbank zunimmt. Sie müssen daher nur eine DB-Instance-Klasse auswählen, die den aktuellen Speicheranforderungen entspricht. Weitere Informationen finden Sie unter Übersicht über Amazon-Aurora-Speicher.

- Virtual Private Cloud (VPC): Wenn bereits eine VPC vorhanden ist, können Sie diese mit dem Aurora PostgreSQL-DB-Cluster verwenden, indem Sie ihre VPC-ID auswählen, z. B. `vpc-a464d1c1`. Weitere Informationen zum Erstellen einer VPC finden Sie unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).

  Anderenfalls können Sie von Amazon RDS eine neue VPC erstellen lassen, indem Sie auf Create new VPC (Neue VPC erstellen) klicken.
- DB-Subnetzgruppe: Wenn bereits eine Subnetzgruppe vorhanden ist, können Sie diese mit dem Aurora-PostgreSQL-DB-Cluster verwenden, indem Sie Ihre Subnetzgruppen-ID auswählen, z. B. `gs-subnet-group1`.
- Public Access (Öffentlicher Zugang): Wählen Sie Nein aus, um festzulegen, dass nur Ressourcen innerhalb der VPC auf Instances im DB-Cluster zugreifen dürfen. Wählen Sie Ja aus, um anzugeben, dass Ressourcen im öffentlichen Netzwerk auf Instances im DB-Cluster zugreifen dürfen.

  > **Note**
  >
  > Der Produktions-DB-Cluster muss sich nicht unbedingt in einem öffentlichen Subnetz befinden, da nur die Anwendungsserver Zugriff auf ihren DB-Cluster benötigen. Wenn sich das DB-Cluster nicht in einem öffentlichen Subnetz befinden soll, stellen Sie Public Access (Öffentlicher Zugang) auf No (Nein).
- VPC-Sicherheitsgruppe: Wählen Sie eine VPC-Sicherheitsgruppe aus, um den Zugriff auf Ihre Datenbank zu ermöglichen.
- Availability Zone: Wählen Sie die Availability Zone zum Hosten der primären Instance des Aurora PostgreSQL-DB-Clusters aus. Damit Amazon RDS eine Availability Zone für Sie auswählt, klicken Sie auf Keine Präferenz.
- Database Port (Datenbankport): Geben Sie den Standardport ein, der beim Verbinden der Instances im Aurora PostgreSQL-DB-Cluster verwendet werden soll. Der Standardwert ist `5432`.

  > **Note**
  >
  > Möglicherweise lässt die Firewall eines Unternehmens den Zugriff auf Standard-Ports, wie z. B. den PostgreSQL-Standard-Port 5432, nicht zu. Geben Sie in diesem Fall einen Port ein, der von der Firewall Ihres Unternehmens zugelassen wird.

Sie benötigen diesen Portwert später, wenn Sie eine Verbindung mit dem Aurora PostgreSQL-DB-Cluster herstellen.

- Enable Encryption (Verschlüsselung aktivieren): Wählen Sie Enable Encryption (Verschlüsselung aktivieren) aus, um Ihr neues Aurora PostgreSQL-DB-Cluster im Ruhezustand zu verschlüsseln. Wählen Sie außerdem einen KMS-Schlüssel als Wert AWS KMS key.
- Auto minor version upgrade (Upgrade einer Unterversion automatisch durchführen): Wählen Sie Enable auto minor version upgrade (Upgrade einer Unterversion automatisch durchführen) aus, wenn Aktualisierungen der PostgreSQL-DB-Engine-Version automatisch im Aurora PostgreSQL-DB-Cluster installiert werden sollen, sobald sie verfügbar sind.

  Die Option Auto Minor Version Upgrade (Upgrade einer Unterversion automatisch durchführen) gilt nur bei Upgrades für Engine-Unterversionen von PostgreSQL für Ihren Aurora PostgreSQL-DB-Cluster. Regelmäßige Patches zur Aufrechterhaltung der Systemstabilität sind davon nicht betroffen.

6. Wählen Sie Migrate (Migrieren) aus, um den DB-Snapshot zu migrieren.
7. Wählen Sie Datenbanken, um den neuen DB-Cluster zu sehen. Wählen Sie den neuen DB-Cluster an, um den Fortschritt der Migration zu überwachen. Wenn die Migration abgeschlossen ist, lautet der Status für den Cluster Available (Verfügbar). Auf der Registerkarte Konnektivität und Sicherheit finden Sie den Cluster-Endpunkt, der für die Verbindung mit der primären Writer-Instance des DB-Clusters verwendet werden soll. Weitere Informationen zum Herstellen einer Verbindung mit einem Aurora PostgreSQL-DB-Cluster finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

AWS CLI

Wenn Sie die AWS CLI verwenden, um einen RDS-for-PostgreSQL-DB-Snapshot zu Aurora PostgreSQL zu migrieren, sind zwei AWS CLI-Befehle nötig. Zuerst erstellen Sie mit dem AWS CLI-Befehl `restore-db-cluster-from-snapshot` einen neuen Aurora-PostgreSQL-DB-Cluster. Dann erstellen Sie mit dem Befehl `create-db-instance` die primäre DB-Instance im neuen Cluster, um die Migration abzuschließen. Das folgende Verfahren erstellt einen Aurora-PostgreSQL-DB-Cluster mit einer primären DB-Instance, die dieselbe Konfiguration wie die DB-Instance hat, die zum Erstellen des Snapshots verwendet wurde.

Einen RDS-for-PostgreSQL-DB-Snapshot auf einen Aurora-PostgreSQL-DB-Cluster migrieren

1. Verwenden Sie den Befehl describe-db-snapshots, um Informationen über den zu migrierenden DB-Snapshot zu erhalten. Sie können den Parameter `--db-instance-identifier` oder `--db-snapshot-identifier` im Befehl angeben. Wenn Sie keinen dieser Parameter angeben, erhalten Sie alle Snapshots.

   ```
   aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
   ```

2. Der Befehl gibt alle Konfigurationsdetails für alle Snapshots zurück, die aus der angegebenen DB-Instance erstellt wurden. Suchen Sie in der Ausgabe nach dem Snapshot, den Sie migrieren möchten, und suchen Sie nach seinem Amazon-Ressourcennamen (ARN). Weitere Informationen zu Amazon-RDS-ARNs finden Sie unter Amazon Relational Database Service (Amazon RDS). Der ARN sieht folgendermaßen oder ähnlich aus.

   ```
   “DBSnapshotArn": "arn:aws:rds:aws-region:111122223333:snapshot:<snapshot_name>"
   ```

   In der Ausgabe finden Sie auch Konfigurationsdetails zur RDS-for-PostgreSQL-DB-Instance, z. B. die Engine-Version, den zugewiesenen Speicher, ob die DB-Instance verschlüsselt ist oder nicht und so weiter.

3. Starten Sie die Migration mit dem Befehl restore-db-cluster-from-snapshot. Geben Sie die folgenden Parameter an:

   - `--db-cluster-identifier` – Der Name, den Sie dem Aurora-PostgreSQL-DB-Cluster geben möchten. Dieser Aurora-DB-Cluster ist das Ziel für Ihre DB-Snapshot-Migration.
   - `--snapshot-identifier` – Der Amazon-Ressourcenname (ARN) des zu migrierenden DB-Snapshot.
   - `--engine` – Geben Sie `aurora-postgresql` für die Aurora-DB-Cluster-Engine an.
   - `--kms-key-id` – Mit diesem optionalen Parameter können Sie einen verschlüsselten Aurora-PostgreSQL-DB-Cluster aus einem unverschlüsselten DB-Snapshot erstellen. Sie können auch einen anderen Verschlüsselungsschlüssel für den DB-Cluster auswählen als den Schlüssel, der für den DB-Snapshot verwendet wird.

> **Note**
>
> Aus einem verschlüsselten DB-Snapshot kann kein unverschlüsselter Aurora-PostgreSQL-DB-Cluster erstellt werden.

Wenn der Parameter `--kms-key-id` nicht wie gezeigt angegeben ist, erstellt restore-db-cluster-from-snapshot AWS CLI einen leeren Aurora-PostgreSQL-DB-Cluster, der entweder mit demselben Schlüssel wie der DB-Snapshot verschlüsselt wird oder unverschlüsselt ist, wenn der Quell-DB-Snapshot nicht verschlüsselt ist.

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier cluster-name \
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-snapshot-name \
    --engine aurora-postgresql
```

Für Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier new_cluster ^
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-snapshot-name ^
    --engine aurora-postgresql
```

4. Der Befehl gibt Details zum Aurora-PostgreSQL-DB-Cluster zurück, der für die Migration erstellt wird. Sie können den Status des Aurora-PostgreSQL-DB-Clusters mit dem AWS CLI-Befehl describe-db-clusters.

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. Wenn der DB-Cluster „verfügbar“ wird, verwenden Sie den Befehl create-db-instance, um den Aurora-PostgreSQL-DB-Cluster mit der DB-Instance zu füllen, die auf Ihrem Amazon-RDS-DB-Snapshot basiert. Geben Sie die folgenden Parameter an:

   - `--db-cluster-identifier` – Der Name des neuen Aurora-PostgreSQL-DB-Clusters, den Sie im vorherigen Schritt erstellt haben.

- `--db-instance-identifier` – Der Name, den Sie der DB-Instance zuweisen möchten. Diese Instance wird zum Primärknoten in Ihrem Aurora-PostgreSQL-DB-Cluster.
- `----db-instance-class` – Geben Sie die zu verwendende DB-Instance-Klasse an. Wählen Sie aus den DB-Instance-Klassen, die von der Aurora-PostgreSQL-Version unterstützt werden, zu der Sie migrieren. Weitere Informationen erhalten Sie unter DB-Instance-Klassenarten und Unterstützte DB-Engines für DB-Instance-Klassen.
- `--engine` – Geben Sie `aurora-postgresql` für die DB-Instance an.

Sie können die DB-Instance auch mit einer anderen Konfiguration als dem Quell-DB-Snapshot erstellen, indem Sie die entsprechenden Optionen im AWS CLI-Befehl `create-db-instance` angeben. Für weitere Informationen siehe den Befehl create-db-instance.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier cluster-name \
    --db-instance-identifier --db-instance-class db.instance.class \
    --engine aurora-postgresql
```

Für Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier cluster-name ^
    --db-instance-identifier --db-instance-class db.instance.class ^
    --engine aurora-postgresql
```

Wenn der Migrationsprozess abgeschlossen ist, verfügt der Aurora-PostgreSQL-Cluster über eine ausgefüllte primäre DB-Instance.

Datenmigration von einer RDS-for-PostgreSQL-DB-Instance zu einem Aurora-PostgreSQL-DB-Cluster unter Verwendung eines Aurora–Lesereplikats

Sie können eine RDS-for-PostgreSQL-DB-Instance als Grundlage für einen neuen Aurora-PostgreSQL-DB-Cluster verwenden, indem Sie eine Aurora-Lesereplikat für den Migrationsprozess verwenden. Die Aurora-Read-Replica-Option ist nur für die Migration innerhalb desselben AND-Kontos verfügbar AWS-Region und nur verfügbar, wenn die Region eine kompatible Version von Aurora PostgreSQL für Ihre RDS for PostgreSQL-DB-Instance anbietet. Kompatibel bedeutet, dass die Aurora-PostgreSQL-Version mit der RDS-for-PostgreSQL-Version identisch ist oder dass es sich um eine höhere Nebenversion in derselben Hauptversionsfamilie handelt.

Um diese Technik beispielsweise zum Migrieren einer RDS-for-PostgreSQL-11.14-DB-Instance verwenden zu können, muss die Region Aurora PostgreSQL Version 11.14 oder eine höhere Nebenversion in der PostgreSQL-Version-11-Familie anbieten.

Themen

- Übersicht über das Migrieren von Daten mittels einer Aurora-Read Replica
- Vorbereiten der Migration von Daten mithilfe einer Aurora Read Replica
- Erstellen einer Aurora Read Replica
- Hochstufen einer Aurora Read Replica

Übersicht über das Migrieren von Daten mittels einer Aurora-Read Replica

Migrieren von einer RDS-for-PostgreSQL-DB-Instance zu einem Aurora-PostgreSQL-DB-Cluster ist ein mehrstufiges Verfahren. Zuerst erstellen Sie ein Aurora-Lesereplikat Ihrer Quell-RDS-Instance-for-PostgreSQL-DB-Instance. Dies startet einen Replikationsprozess von Ihrer RDS-for-PostgreSQL-DB-Instance zu einem speziellen DB-Cluster, der als Replikat-Cluster bekannt ist. Der Replikat-Cluster besteht ausschließlich aus einem Aurora-Lesereplikat (einer Reader-Instance).

> **Note**
>
> Es kann mehrere Stunden pro Terabyte Daten dauern, bis die Migration abgeschlossen ist.

Hochstufen eines Aurora-PostgreSQL-Replikats

Nachdem Sie einen DB-Cluster von Aurora PostgreSQL erstellt haben, gehen Sie wie folgt vor, um das Aurora-Replikat hochzustufen:

1. Halten Sie den gesamten Datenbank-Schreib-Workload auf der Quell-DB-Instance von RDS für PostgreSQL an.
2. Rufen Sie die aktuelle `WAL  LSN` aus der Quell-DB-Instance von RDS für PostgreSQL ab.

   ```
   SELECT pg_current_wal_lsn();
   pg_current_wal_lsn
   --------------------
   0/F0000318
   (1 row)
   ```

3. Überprüfen Sie auf dem Replikat-Cluster von Aurora PostgreSQL, ob die wiedergegebene LSN größer ist als die LSN aus Schritt 2:

   ```
   SELECT pg_last_wal_replay_lsn();
   pg_last_wal_replay_lsn
   ------------------------
   0/F0000400
   (1 row)
   ```

   Alternativ können Sie die folgende Abfrage auf der Quell-DB-Instance von RDS für PostgreSQL verwenden:

   ```
   SELECT restart_lsn FROM pg_replication_slots;
   ```

4. Stufen Sie den Replikat-Cluster von Aurora PostgreSQL hoch.

   Wenn die Replikation anhält, wird der Replikat-Cluster zu einem eigenständigen DB-Cluster von Aurora PostgreSQL und der Leser zur Schreiber-Instance für den Cluster hochgestuft. Nun können Sie Instances zum DB-Cluster von Aurora PostgreSQL hinzufügen, um dessen Größe Ihrem Anwendungsfall entsprechend anpassen. Wenn Sie die ursprüngliche DB-Instance von RDS für PostgreSQL nicht mehr benötigen, können Sie sie löschen.

Sie können kein Aurora-Lesereplikat erstellen, falls die RDS-for-PostgreSQL-DB-Instance bereits über ein Aurora-Lesereplikat verfügt oder ein regionsübergreifendes Lesereplikat enthält.

Vorbereiten der Migration von Daten mithilfe einer Aurora Read Replica

> **Note**
>
> Bei der Vorbereitung der Migration von Daten zu Aurora PostgreSQL ist es wichtig, Tabellen ohne Protokollierung zu identifizieren und angemessen zu handhaben. Weitere Informationen finden Sie unter Handhaben von Tabellen ohne Protokollierung während der Migration.

Während des Migrationsprozesses mit Aurora-Lesereplikat werden Aktualisierungen, die mit den Daten der Quell-RDS-for-PostgreSQL-DB-Instance vorgenommen werden, asynchron in diesem Aurora-Lesereplikat des Replica-Clusters repliziert. Der Prozess verwendet die native Streaming-Replikationsfunktion von PostgreSQL, die Write-Ahead-Log-Segmente (WAL) auf der Quell-Instance speichert. Bevor Sie mit diesem Migrationsprozess beginnen, stellen Sie sicher, dass die Instance über genügend Speicherkapazität verfügt, indem Sie die Werte für die in der Tabelle aufgeführten Metriken überprüfen.

Metrik	Beschreibung
`FreeStorageSpace`	Den verfügbaren Speicherplatz. Einheiten: Byte
`OldestReplicationSlotLag`	Der Verzögerungsgröße für WAL-Daten in der Replika, die die höchste Verzögerung aufweist. Einheiten: Megabyte
`RDSToAuroraPostgreSQLReplicaLag`	Die Zeit in Sekunden, wie lange ein Aurora PostgreSQL-DB-Cluster hinter der RDS-DB-Quell-Instance liegt.
`TransactionLogsDiskUsage`	Der von den Transaktionsprotokollen verwendete Festplattenspeicher. Einheiten: Megabyte

Weitere Informationen zur Überwachung Ihrer RDS-Instance finden Sie unter Überwachung im Amazon-RDS-Benutzerhandbuch.

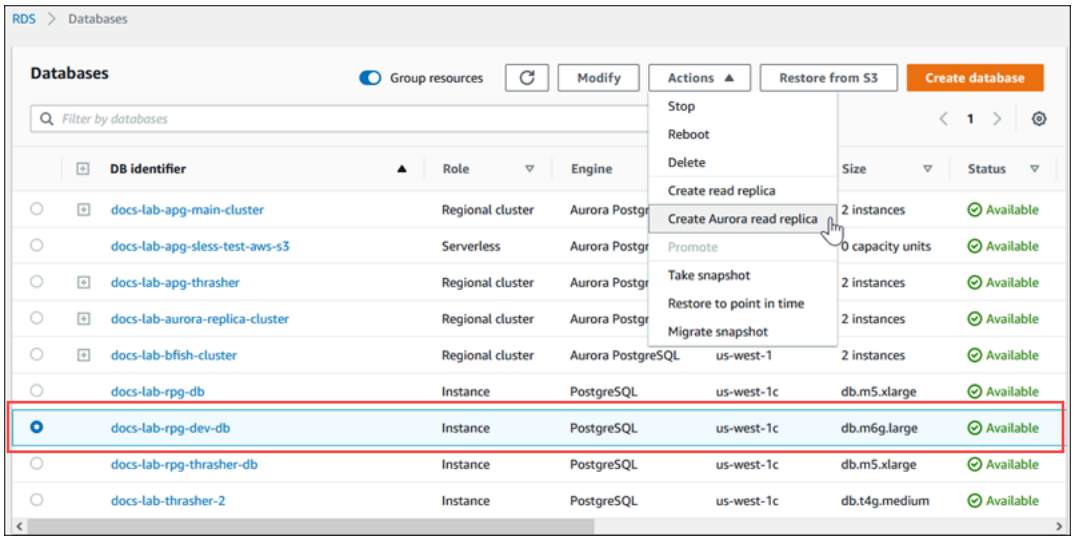
Erstellen einer Aurora Read Replica

Sie können eine Aurora-Read Replica für eine RDS for PostgreSQL-DB-Instance erstellen, indem Sie den oder den AWS-Managementkonsole verwenden. AWS CLI Die Option zum Erstellen einer Aurora-Read Replica mit dem AWS-Managementkonsole ist nur verfügbar, wenn das eine kompatible Aurora PostgreSQL-Version AWS-Region anbietet. Das heißt, es ist nur verfügbar, wenn es eine Aurora-PostgreSQL-Version gibt, die mit der RDS-for-PostgreSQL-Version oder einer höheren Nebenversion in derselben Hauptversionsfamilie identisch ist.

Konsole

Sie erstellen Sie eine Aurora Read Replica aus einer PostgreSQL-DB-Instance

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie die RDS-for-PostgreSQL-DB-Instance aus, die Sie als Quelle für Ihr Aurora-Lesereplikat verwenden möchten. Wählen Sie für Actions (Aktionen) Create Aurora read replica (Aurora Read Replica erstellen) aus. Wenn diese Auswahl nicht angezeigt wird, bedeutet dies, dass eine kompatible Aurora-PostgreSQL-Version in der Region nicht verfügbar ist.



4. Auf der Seite Einstellungen für Aurora-Lesereplikat erstellen konfigurieren Sie die Eigenschaften für den Aurora-PostgreSQL-DB-Cluster wie in der folgenden Tabelle dargestellt. Das Replica-DB-Cluster wird aus einem Snapshot der Quell-DB-Instance unter Verwendung des gleichen

Master-Benutzernamens und Kennworts wie die Quelle erstellt, sodass Sie diese derzeit nicht ändern können.

Option	Beschreibung
DB-Instance-Klasse	Wählen Sie eine DB-Instance-Klasse aus, die die Verarbeitungs- und Speicheranforderungen der primären Instance im DB-Cluster erfüllt. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.
Multi-AZ-Bereitstellung	Während der Migration nicht verfügbar
DB-Instance-Kennung	Geben Sie den Namen ein, den Sie der DB-Instan ce zuweisen möchten. Dieser Bezeichner wird in der Endpunktadresse für die primäre Instance des neuen DB-Clusters verwendet. Für den DB-Instance-Bezeichner gelten folgende Einschränkungen: • Er muss zwischen 1 und 63 alphanumerische Zeichen oder Bindestriche enthalten. • Das erste Zeichen muss ein Buchstabe sein. • Darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten. • Sie muss für alle DB-Instances, für jedes AWS Konto und für jedes Konto eindeutig sein AWS-Region.
Virtual Private Cloud (VPC)	Wählen Sie die VPC zum Hosten des DB-Clusters aus. Wählen Sie Create new VPC (Neue VPC erstellen) aus, damit Amazon RDS eine VPC für Sie erstellt. Weitere Informationen finden Sie unter Vorausset zungen für DB-Cluster.

Option	Beschreibung
DB-Subnetzgruppe	Wählen Sie die DB-Subnetzgruppe aus, die für den DB-Cluster verwendet werden soll. Wählen Sie Create new DB subnet group (Neue DB-Subnetzgruppe erstellen) aus, wenn Amazon RDS eine DB-Subnet zgruppe für Sie erstellen soll. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.
Öffentliche Zugänglichkeit	Klicken Sie auf Ja, um dem DB-Cluster eine öffentlic he IP-Adresse zuzuweisen. Wählen Sie andernfalls Nein. Die Instances in Ihrem DB-Cluster können eine Mischung aus öffentlichen und privaten DB-Instan ces sein. Weitere Informationen darüber, wie Sie den öffentlichen Zugriff für Instances deaktivieren, finden Sie unter Ausblenden einer DB-Clusters in einer VPC vor dem Internet.
Availability Zone	Legen Sie fest, ob Sie eine bestimmte Availability Zone angeben möchten. Weitere Informationen über Availability Zones finden Sie unter Regionen und Availability Zones.
VPC-Sicherheitsgruppen	Wählen Sie eine oder mehrere VPC-Sicherheitsgru ppen aus, um den Netzwerkzugriff auf den DB-Cluste r zu sichern. Wählen Sie Create new VPC security group (Neue VPC-Sicherheitsgruppe erstellen) aus, damit Amazon RDS eine VPC-Sicherheitsgruppe für Sie erstellt. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.

Option	Beschreibung
Datenbankport	Geben Sie den Port an, über den Anwendungen und Dienstprogramme auf die Datenbank zugreifen können. Aurora PostgreSQL-DB-Cluster haben standardmäßig den Standard-PostgreSQL-Port 5432 eingestellt. Die Firewalls einiger Unternehmen blockieren Verbindungen mit diesem Port. Wenn die Firewall Ihres Unternehmens den Standardport blockiert, wählen Sie einen anderen Port für den neuen DB-Cluster aus.
DB-Parametergruppe	Wählen Sie eine DB-Parametergruppe für den Aurora-PostgreSQL-DB-Cluster aus. Aurora verfügt über eine DB-Standardparametergruppe, die Sie verwenden können, oder Sie können Ihre eigene DB-Parametergruppe erstellen. Weitere Informationen zu DB-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.
DB-Cluster-Parametergruppe	Wählen Sie eine DB-Cluster-Parametergruppe für den Aurora-PostgreSQL-DB-Cluster aus. Aurora verfügt über eine DB-Cluster-Standardparametergruppe, die Sie verwenden können, oder Sie können Ihre eigene DB-Cluster-Parametergruppe erstellen. Weitere Informationen zu DB-Cluster-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.
Verschlüsselung	Wählen Sie Enable encryption (Verschlüsselung aktivieren) aus, wenn das neue Aurora-DB-Cluster im Ruhezustand verschlüsselt werden soll. Wenn Sie Verschlüsselung aktivieren wählen, wählen Sie auch einen KMS-Schlüssel als Wert AWS KMS key .

Option	Beschreibung
Priorität	Wählen Sie eine Failover-Priorität für den DB-Cluste r aus. Wenn Sie keinen Wert auswählen, wird als Standard tier-1 (Tier-1) eingestellt. Diese Priorität bestimmt die Reihenfolge, in der Aurora-Replikate bei der Wiederherstellung nach einem Ausfall der primären Instance hochgestuft werden. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.
Aufbewahrungszeitraum für Backups	Wählen Sie den Zeitraum (zwischen 1 und 35 Tage) für die Aufbewahrung von Sicherungskopien der Datenbank in Aurora aus. Sicherungskopien können sekundengenau für point-in-time Wiederherstellungen (PITR) Ihrer Datenbank verwendet werden.
Verbesserte Überwachung	Wählen Sie Erweiterte Überwachung aktivieren aus, um die Erfassung von Metriken in Echtzeit für das Betriebssystem zu aktivieren, in dem Ihr DB-Cluster ausgeführt wird. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).
Überwachungsrolle	Nur verfügbar, wenn Enable Enhanced Monitoring (Erweiterte Überwachung aktivieren) festgelegt ist. Legen Sie die zu „Enhanced Monitoring“ (Erweiterten Überwachung) verwendete AWS Identity and Access Management -(IAM)-Rolle fest. Weitere Informati onen finden Sie unter Einrichten und Aktivieren von „Enhanced Monitoring“ (Erweiterte Überwachung).
Granularität	Nur verfügbar, wenn Enable Enhanced Monitoring (Erweiterte Überwachung aktivieren) festgelegt ist. Mit ihr können Sie die Zeitspanne zwischen den Erfassung en der Kennzahlen des DB-Clusters in Sekunden festlegen.

Option	Beschreibung
Kleinere Versions-Updates automatisch aktivieren	Klicken Sie auf Ja, wenn Sie möchten, dass Ihr Aurora PostgreSQL-DB-Cluster automatisch Upgrades für PostgreSQL-DB-Engine-Unterversionen erhält, sobald sie verfügbar werden. Die Option Auto Minor Version Upgrade (Upgrade einer Unterversion automatisch durchführen) gilt nur bei Upgrades für Engine-Unterversionen von PostgreSQL für Ihren Aurora PostgreSQL-DB-Cluster. Regelmäßige Patches zur Aufrechterhaltung der Systemstabilität sind davon nicht betroffen.
Wartungsfenster	Wählen Sie den wöchentlichen Zeitraum aus, in dem Systemwartungen durchgeführt werden können.

5. Wählen Sie Read Replica erstellen aus.

AWS CLI

Um mithilfe von eine Aurora-Read Replica aus einer RDS-Quell-DB-Instance für PostgreSQL zu erstellen, verwenden Sie zunächst den create-db-clusterCLI-Befehl AWS CLI, um einen leeren Aurora-DB-Cluster zu erstellen. Sobald der DB-Cluster existiert, verwenden Sie den Befehl create-db-instance, um die primäre Instance für Ihren DB-Cluster zu erstellen. Die primäre Instance ist die erste in einem Aurora-DB-Cluster erstellte Instance. In diesem Fall wird sie zunächst als Aurora-Lesereplikat Ihrer RDS-for-PostgreSQL-DB-Instance erstellt. Wenn der Prozess abgeschlossen wird, wurde Ihre RDS-for-PostgreSQL-DB-Instance effektiv zu einem Aurora-PostgreSQL-DB-Cluster migriert.

Sie müssen das Hauptbenutzerkonto (normalerweise `postgres`), sein Passwort oder den Datenbanknamen nicht angeben. Die Aurora-Read Replica bezieht diese automatisch von der Quell-RDS für PostgreSQL-DB-Instance, die Sie beim Aufrufen der Befehle identifizieren. AWS CLI

Sie müssen allerdings die Engine-Version angeben, die für den Aurora-PostgreSQL-DB-Cluster und die DB-Instance verwendet werden soll. Die von Ihnen angegebene Version sollte mit der Quell-DB-Instance von RDS für PostgreSQL übereinstimmen. Wenn die Quell-DB-Instance von RDS für PostgreSQL verschlüsselt ist, müssen Sie auch die Verschlüsselung für die primäre Instance des

Aurora-PostgreSQL-DB-Clusters angeben. Die Migration einer verschlüsselten Instance zu einem unverschlüsselten Aurora-DB-Cluster wird nicht unterstützt.

In den folgenden Beispielen wird ein Aurora-PostgreSQL-DB-Cluster mit dem Namen `my-new-aurora-cluster` erstellt, der eine unverschlüsselte RDS-DB-Quell-Instance verwenden soll. Sie erstellen zuerst den Aurora-PostgreSQL-DB-Cluster, indem Sie den CLI-Befehl `create-db-cluster` aufrufen. Das Beispiel zeigt, wie Sie den optionalen Parameter `--storage-encrypted` verwenden, um anzugeben, dass der DB-Cluster verschlüsselt werden soll. Da die Quell-DB nicht verschlüsselt ist, wird `--kms-key-id` verwendet, um den zu verwendenden Schlüssel anzugeben. Weitere Informationen zu den erforderlichen und optionalen Parametern finden Sie in der Liste, die auf das Beispiel folgt.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster  \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-subnet-group-name my-db-subnet
    --vpc-security-group-ids sg-11111111
    --engine aurora-postgresql \
    --engine-version same-as-your-rds-instance-version \
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

Für Windows:

```
aws rds create-db-cluster  ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-subnet-group-name my-db-subnet ^
    --vpc-security-group-ids sg-11111111 ^
    --engine aurora-postgresql ^
    --engine-version same-as-your-rds-instance-version ^
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db ^
    --storage-encrypted ^
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

In der folgenden Liste finden Sie weitere Informationen zu einigen der im Beispiel gezeigten Optionen. Sofern nicht anders angegeben, sind diese Parameter erforderlich.

- `--db-cluster-identifier` – Sie müssen Ihrem neuen Aurora-PostgreSQL-DB-Cluster einen Namen geben.
- `--db-subnet-group-name` – Erstellen Sie Ihren Aurora-PostgreSQL-DB-Cluster im selben DB-Subnetz wie die DB-Quell-Instance.
- `--vpc-security-group-ids` – Legen Sie die Sicherheitsgruppe für Ihren Aurora-PostgreSQL-DB-Cluster fest.
- `--engine-version` – Legen Sie die Version fest, die Sie für den Aurora-PostgreSQL-DB-Cluster verwenden möchten. Dies sollte die gleiche Version oder eine höhere Nebenversion sein wie die Version, die von Ihrer Quell-DB-Instance von RDS für PostgreSQL verwendet wird.
- `--replication-source-identifier` – Identifizieren Sie Ihre RDS-for-PostgreSQL-DB-Instance über den Amazon-Ressourcennamen (ARN). Weitere Informationen zu Amazon RDS ARNs finden Sie unter Amazon Relational Database Service (Amazon RDS) in der Allgemeine AWS-Referenz. Ihres DB-Clusters.
- `--storage-encrypted` – Optional. Verwenden Sie diese Option, wenn die Verschlüsselung wie folgt angegeben werden muss:
  - Verwenden Sie diesen Parameter, wenn die Quell-DB-Instance über verschlüsselten Speicher verfügt. Der Aufruf von `create-db-cluster` schlägt fehl, wenn Sie diesen Parameter nicht mit einer Quell-DB-Instance verwenden, die über verschlüsselten Speicher verfügt. Wenn Sie einen anderen Schlüssel für den Aurora-PostgreSQL-DB-Cluster verwenden möchten als den von der Quell-DB-Instance verwendeten Schlüssel, müssen Sie auch die `--kms-key-id` angeben.
  - Verwenden Sie diese Option, wenn der Speicher der Quell-DB-Instance unverschlüsselt ist, der Aurora-PostgreSQL-DB-Cluster jedoch die Verschlüsselung verwenden soll. In diesem Fall müssen Sie auch den Verschlüsselungsschlüssel identifizieren, der mit dem `--kms-key-id`-Parameter verwendet werden soll.
- `--kms-key-id` – Optional. Wenn Sie diese Option verwenden, können Sie den Schlüssel angeben, der für die Speicherverschlüsselung (`--storage-encrypted`) verwendet werden soll, indem Sie den ARN, die ID, den Alias-ARN oder den Aliasnamen des Schlüssels angeben. Dieser Parameter wird nur für die folgenden Situationen benötigt:
  - Wenn Sie einen anderen Schlüssel für den Aurora-PostgreSQL-DB-Cluster auswählen möchten als den, der von der Quell-DB-Instance verwendet wird.

- Wenn Sie einen verschlüsselten Cluster aus einer unverschlüsselten Quelle erstellen möchten. In diesem Fall müssen Sie den Schlüssel angeben, den Aurora PostgreSQL für die Verschlüsselung verwenden soll.

Nachdem Sie den Aurora-PostgreSQL-DB-Cluster erstellt haben, erstellen Sie die primäre Instance mithilfe des CLI-Befehls `create-db-instance`, wie im Folgenden gezeigt:

Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-instance-class db.x2g.16xlarge \
    --db-instance-identifier rpg-for-migration \
    --engine aurora-postgresql
```

Für Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-instance-class db.x2g.16xlarge ^
    --db-instance-identifier rpg-for-migration ^
    --engine aurora-postgresql
```

In der folgenden Liste finden Sie weitere Informationen zu einigen der im Beispiel gezeigten Optionen.

- `--db-cluster-identifier` – Der Name des neuen Aurora-PostgreSQL-DB-Clusters, den Sie im vorherigen Schritt mit dem Befehl `create-db-instance` erstellt haben.
- `--db-instance-class` – Der Name der DB-Instance-Klasse, die für Ihre primäre Instance verwendet werden soll, z. B. `db.r4.xlarge`, `db.t4g.medium`, `db.x2g.16xlarge` usw. Eine Liste der verfügbaren DB-Instance-Klassen finden Sie unter DB-Instance-Klassenarten.
- `--db-instance-identifier` – Geben Sie den Namen an, den Ihre primäre Instance erhalten soll.
- `--engine aurora-postgresql` – Geben Sie `aurora-postgresql` für die Engine an.

RDS-API

Wenn Sie ein Aurora-Lesereplikat aus einer Quell-DB-Instance von RDS für PostgreSQL erstellen möchten, verwenden Sie zuerst die RDS-API-Operation `CreateDBCluster`, um einen neuen Aurora-DB-Cluster für das Aurora-Lesereplikat zu erstellen, das aus Ihrer RDS-for-PostgreSQL-Quell-DB-Instance erstellt wird. Wenn der Aurora-PostgreSQL-DB-Cluster verfügbar ist, verwenden Sie `CreateDBInstance`, um die primäre Instance für den Aurora-DB-Cluster zu erstellen.

Sie müssen das Hauptbenutzerkonto (normalerweise `postgres`), sein Passwort oder den Datenbanknamen nicht angeben. Das Aurora-Lesereplikat bezieht diese Angaben automatisch aus der RDS-for-PostgreSQL-Quell-DB-Instance, die in `ReplicationSourceIdentifier` angegeben ist.

Sie müssen allerdings die Engine-Version angeben, die für den Aurora-PostgreSQL-DB-Cluster und die DB-Instance verwendet werden soll. Die von Ihnen angegebene Version sollte mit der Quell-DB-Instance von RDS für PostgreSQL übereinstimmen. Wenn die Quell-DB-Instance von RDS für PostgreSQL verschlüsselt ist, müssen Sie auch die Verschlüsselung für die primäre Instance des Aurora-PostgreSQL-DB-Clusters angeben. Die Migration einer verschlüsselten Instance zu einem unverschlüsselten Aurora-DB-Cluster wird nicht unterstützt.

Wenn Sie den Aurora-DB-Cluster für das Aurora-Lesereplikat erstellen möchten, verwenden Sie die RDS-API-Operation `CreateDBCluster` mit folgenden Parametern:

- `DBClusterIdentifier` – Der Name des zu erstellenden DB-Clusters.
- `DBSubnetGroupName` – Der Name der DB-Subnetzgruppe, die mit diesem DB-Cluster verknüpft werden soll.
- `Engine=aurora-postgresql` – Der Name der zu verwendenden Engine.
- `ReplicationSourceIdentifier` – Der Amazon Resource Name (ARN) für die PostgreSQL-Quell-DB-Instance. Weitere Informationen zu Amazon RDS ARNs finden Sie unter Amazon Relational Database Service (Amazon RDS) in der Allgemeine Amazon Web Services-Referenz. Wenn `ReplicationSourceIdentifier` eine verschlüsselte Quelle identifiziert, verwendet Amazon RDS Ihren Standard-KMS-Schlüssel, es sei denn, Sie geben mit der Option `KmsKeyId` einen anderen Schlüssel an.
- `VpcSecurityGroupIds`— Die Liste der Amazon EC2 VPC-Sicherheitsgruppen, die diesem DB-Cluster zugeordnet werden sollen.
- `StorageEncrypted` – Gibt an, ob der DB-Cluster verschlüsselt ist. Wenn Sie diesen Parameter verwenden, ohne auch die `ReplicationSourceIdentifier` anzugeben, verwendet Amazon RDS Ihren KMS-Standardschlüssel.

- `KmsKeyId` – Der Schlüssel für einen verschlüsselten Cluster. Wenn Sie diese Option verwenden, können Sie den Schlüssel angeben, der für die Speicherverschlüsselung verwendet werden soll, indem Sie den ARN, die ID, den Alias-ARN oder den Aliasnamen des Schlüssels angeben.

Weitere Informationen finden Sie unter `CreateDBCluster` in der Amazon-RDS-API-Referenz.

Sobald der Aurora-DB-Cluster verfügbar ist, können Sie eine primäre Instance dafür erstellen, indem Sie die RDS-API-Operation `CreateDBInstance` mit folgenden Parametern verwenden:

- `DBClusterIdentifier` – Der Name Ihres DB-Clusters.
- `DBInstanceClass` – Der Name der DB-Instance-Klasse, die für Ihre primäre Instance verwendet werden soll.
- `DBInstanceIdentifier` – Der Name Ihrer primären Instance.
- `Engine=aurora-postgresql` – Der Name der zu verwendenden Engine.

Weitere Informationen finden Sie unter `CreateDBInstance` in der Amazon-RDS-API-Referenz.

Hochstufen einer Aurora Read Replica

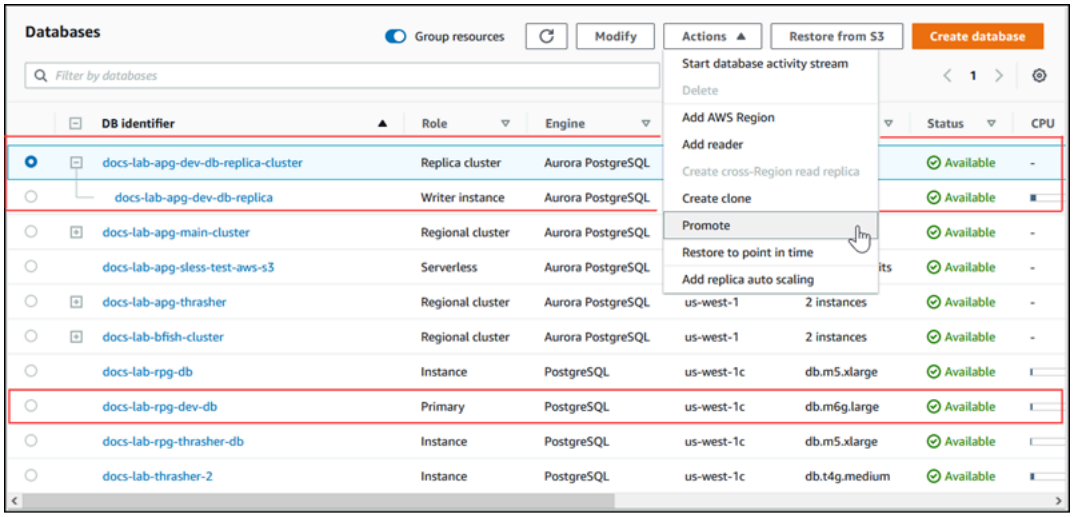
Die Migration zu Aurora PostgreSQL ist erst abgeschlossen, wenn Sie den Replikat-Cluster hochgestuft haben. Löschen Sie also die RDS-for-PostgreSQL-Quell-DB-Instance noch nicht.

Stellen Sie vor dem Hochladen des Replikat-Clusters sicher, dass die RDS-for-PostgreSQL-DB-Instance keine prozessinternen Transaktionen oder andere Aktivitäten enthält, die in die Datenbank schreiben. Wenn die Replikatverzögerung auf dem Aurora-Lesereplikat Null (0) erreicht, können Sie den Replikatcluster heraufstufen. Weitere Informationen zum Überwachen der Replikatverzögerung finden Sie unter Überwachung einer Aurora PostgreSQL-Replikation und Metriken auf Instance-Ebene für Amazon Aurora.

Konsole

So stufen Sie eine Aurora-Read Replica zu einem Aurora-DB-Cluster hoch:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie den Replikatcluster aus.



4. Wählen Sie für Actions (Aktionen)Promote (Hochstufen) aus. Dies kann einige Minuten dauern und zu Ausfallzeiten führen.

Wenn der Prozess abgeschlossen ist, ist der Aurora-Replikatcluster ein regionaler Aurora-PostgreSQL-DB-Cluster mit einer Writer-Instance, die die Daten aus der RDS-for-PostgreSQL-DB-Instance enthält.

AWS CLI

Verwenden Sie den Befehl, um eine Aurora-Read Replica zu einem eigenständigen DB-Cluster hochzustufen. promote-read-replica-db-cluster AWS CLI

Example

Für Linux, macOS oder Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Für Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

RDS-API

Verwenden Sie den RDS-API-Vorgang, um eine Aurora-Read Replica zu einem eigenständigen DB-Cluster hochzustufen. PromoteReadReplicaDBCluster

Nachdem Sie den Replikatcluster hochgestuft haben, können Sie bestätigen, dass das Hochstufen abgeschlossen wurde, indem Sie das Ereignisprotokoll wie folgt überprüfen.

Um zu bestätigen, dass das Aurora-Replikatcluster hochgestuft wurde

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Events.
3. Auf der Seite Events (Ereignisse) finden Sie den Namen Ihres Clusters in der Liste Source (Quelle). Jedes Ereignis hat eine Quelle, einen Typ, eine Uhrzeit und eine Nachricht. Sie können alle Ereignisse sehen, die in Ihrem AWS-Region für Ihr Konto aufgetreten sind. Ein erfolgreiches Hochstufen generiert die folgende Nachricht.

```
Promoted Read Replica cluster to a stand-alone database cluster.
```

Nach Abschluss der Hochstufung wird die Verknüpfung zwischen der Quell-RDS-for-PostgreSQL-DB-Instance und dem Aurora-PostgreSQL-DB-Cluster aufgehoben. Sie können Ihre Clientanwendungen für den Zugriff auf den Endpunkt des Aurora-Lesereplikats konfigurieren. Weitere Informationen zu den Aurora-Endpunkten finden Sie unter Amazon-Aurora-Endpunktverbindungen. Nun können Sie die DB-Instance bei Bedarf sicher löschen.

Optimieren der Abfrageleistung in Aurora PostgreSQL

Die Optimierung der Abfrageleistung ist von entscheidender Bedeutung, da sie dazu beiträgt, dass Datenbanken schneller und effizienter laufen und gleichzeitig weniger Ressourcen beanspruchen, was zu einer besseren Benutzererfahrung und niedrigeren Betriebskosten führt. Amazon Aurora PostgreSQL bietet mehrere Funktionen zur Optimierung der Abfrageleistung für PostgreSQL-Workloads.

Themen

- Verbesserung der Abfrageleistung für Aurora PostgreSQL mit Aurora-optimierten Lesevorgängen
- Optimieren von korrelierten Unterabfragen in Aurora PostgreSQL
- Verbessern der Abfrageleistung mit Adaptive Join

Verbesserung der Abfrageleistung für Aurora PostgreSQL mit Aurora-optimierten Lesevorgängen

Mit Aurora-optimierten Lesevorgängen können Sie eine schnellere Abfrageverarbeitung mit Aurora PostgreSQL erreichen. Eine Aurora-PostgreSQL-DB-Instance, die Aurora-optimierte Lesevorgänge verwendet, bietet eine bis zu achtmal verbesserte Abfragelatenz und bis zu 30 % Kosteneinsparungen für Anwendungen mit großen Datensätzen, die die Speicherkapazität einer DB-Instance überschreiten.

Themen

- Übersicht über Aurora-optimierte Lesevorgänge in PostgreSQL
- Verwenden von Aurora-optimierten Lesevorgängen
- Anwendungsfälle für Aurora-optimierte Lesevorgänge
- Überwachen von DB-Instances, die Aurora-optimierte Lesevorgänge verwenden
- Bewährte Methoden für Aurora-optimierte Lesevorgänge

Übersicht über Aurora-optimierte Lesevorgänge in PostgreSQL

Aurora Optimized Reads ist standardmäßig verfügbar, wenn Sie einen DB-Cluster erstellen, der Graviton-basierte R6gd-, R8gd- und Intel-basierte R6ID-Instances mit nichtflüchtigem Memory-Express () -Speicher verwendet. NVMe Dies ist in den folgenden Versionen von PostgreSQL verfügbar:

- 14.12 und höhere Versionen, 15.7 und höhere Versionen, 16.3 und höhere Versionen, 17.4 und höhere Versionen für R8gd-Instances
- 14.9 und höhere Versionen, 15.4 und höhere Versionen, 16.1 und alle höheren Versionen für R6gd- und R6id-Instances

Aurora-optimierte Lesevorgänge unterstützen zwei Funktionen: mehrstufigen Cache und temporäre Objekte.

Mehrstufiger Cache mit optimierten Lesevorgängen – Mithilfe eines mehrstufigen Caches können Sie die Caching-Kapazität Ihrer DB-Instance um das bis zu Fünffache des Instance-Speichers erweitern. Dadurch wird der Cache automatisch so verwaltet, dass er die aktuellsten, transaktionskonsistenten Daten enthält. Dadurch entfällt für Anwendungen der Aufwand, die Datenaktualität externer, auf

Ergebnismengen basierender Caching-Lösungen zu verwalten. Dies bietet eine bis zu achtmal bessere Latenz für Abfragen, bei denen zuvor Daten aus dem Aurora-Speicher abgerufen wurden.

In Aurora ist der Wert für `shared_buffers` in der Standard-Parametergruppe normalerweise auf etwa 75 % des verfügbaren Speichers festgelegt. Für die Instance-Typen r8gd, r6gd und r6id reduziert Aurora den `shared_buffers` Speicherplatz jedoch um 4,5%, um die Metadaten für den Optimized Reads-Cache zu hosten.

Optimierte temporäre Objekte mit Lesezugriff — Mithilfe temporärer Objekte können Sie eine schnellere Abfrageverarbeitung erreichen, indem Sie die temporären Dateien, die von PostgreSQL generiert werden, im lokalen Speicher ablegen. NVMe Dadurch wird der Datenverkehr zu Elastic Block Storage (EBS) über das Netzwerk reduziert. Dies bietet eine bis zu zweimal bessere Latenz und einen besseren Durchsatz für erweiterte Abfragen, bei denen große Datenmengen sortiert, verbunden oder zusammengeführt werden, die nicht in die auf einer DB-Instance verfügbare Speicherkapazität passen.

In einem I/O-optimierten Aurora-Cluster verwendet Optimized Reads sowohl den mehrstufigen Cache als auch temporäre Objekte im Speicher. NVMe Mit der für optimierte Lesevorgänge aktivierten mehrstufigen Cache-Funktion weist Aurora den doppelten Instance-Speicher für temporäre Objekte zu, etwa 10 % des Speichers für interne Operationen und den verbleibenden Speicher als mehrstufigen Cache. In einem Aurora-Standard-Cluster verwenden optimierte Lesevorgänge nur temporäre Objekte.

E/A-optimierte Aurora-Cluster ermöglichen es Ihnen, den zugewiesenen Speicherplatz für temporäre Objekte mit aktivierten optimierten Lesevorgängen mithilfe des dynamischen Parameters `aurora_temp_space_size` auf Instance-Ebene anzupassen. Diese Größenanpassungsfunktion ist in den folgenden PostgreSQL-Versionen verfügbar:

- 16.8 und alle höheren Versionen
- 15.12 und höhere 15-Versionen
- 14.17 und höhere 14-Versionen

Mit diesem Parameter können Sie die Kapazität von 2x bis 6x des Instance-Speichers skalieren, ohne dass die Datenbank-Engine neu gestartet werden muss. Wenn Sie den Speicherplatz für temporäre Objekte erweitern, wird die Änderung sofort wirksam, unabhängig von gleichzeitigen Workloads. Wenn Sie den Speicherplatz reduzieren, wird die Anpassung jedoch erst abgeschlossen, wenn in den temporären Objekten genügend ungenutzter Speicherplatz für die neue Größenanforderung vorhanden ist. Nachdem Sie die Größe temporärer Objekte mit aktivierten optimierten Lesevorgängen

geändert haben, passt sich der mehrstufige Cache automatisch an, um den verfügbaren Speicherplatz zu nutzen.

Engine	Cluster-Speicherkonfiguration	Temporäre Objekte mit optimierten Lesevorgängen	Optimierter mehrstufiger Cache mit optimierten Lesevorgängen	Unterstützte Versionen
Aurora PostgreSQL-Compatible Edition	Standard	Ja	Nein	• 14.9 und höhere Versionen, 15.4 und höhere Versionen, 16.1 und alle höheren Versionen für r6gd- und r6id-Instances • 14.12 und höhere Versionen, 15.7 und höhere Versionen, 16.3 und höhere Versionen, 17.4 und höhere Versionen für r8gd-Instanzen
	E/A-optimiert	Ja	Ja	

> **Note**
>
> Ein Wechsel zwischen IO-optimierten Clustern und Standard-Clustern auf einer NVMe basierten DB-Instance-Klasse führt zu einem sofortigen Neustart der Datenbank-Engine.

Verwenden Sie in Aurora PostgreSQL den `temp_tablespaces`-Parameter zur Konfiguration des Tabellenbereichs, in dem die temporären Objekte gespeichert werden.

Verwenden Sie den folgenden Befehl, um zu überprüfen, ob die temporären Objekte konfiguriert sind:

```
postgres=> show temp_tablespaces;
temp_tablespaces
--------------------
aurora_temp_tablespace
(1 row)
```

Das `aurora_temp_tablespace` ist ein von Aurora konfigurierter Tablespace, der auf den NVMe lokalen Speicher verweist. Sie können diesen Parameter nicht ändern oder zum Amazon-EBS-Speicher zurückkehren.

Verwenden Sie den folgenden Befehl, um zu überprüfen, ob der Cache der optimierten Lesevorgänge aktiviert ist:

```
postgres=> show shared_preload_libraries;
                shared_preload_libraries
-----------------------------------------------------------
rdsutils,pg_stat_statements,aurora_optimized_reads_cache
```

Verwenden von Aurora-optimierten Lesevorgängen

Wenn Sie eine Aurora PostgreSQL-DB-Instance mit einer der NVMe basierten DB-Instances bereitstellen, verwendet die DB-Instance automatisch Aurora Optimized Reads.

Führen Sie einen der folgenden Schritte aus, um Aurora-optimierte Lesevorgänge zu aktivieren:

- Erstellen Sie einen Aurora PostgreSQL-DB-Cluster mithilfe einer der NVMe basierten DB-Instance-Klassen. Weitere Informationen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

- Ändern Sie einen vorhandenen Aurora PostgreSQL-DB-Cluster so, dass er eine der NVMe basierten DB-Instance-Klassen verwendet. Weitere Informationen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Aurora Optimized Reads ist überall verfügbar AWS-Regionen , wo eine oder mehrere DB-Instance-Klassen mit lokalem NVMe SSD-Speicher unterstützt werden. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Um zurück zu einer Aurora-Instance mit nicht optimierten Lesevorgängen zu wechseln, ändern Sie die DB-Instance-Klasse Ihrer Aurora-Instance auf eine ähnliche Instance-Klasse ohne NVMe kurzlebigen Speicher für Ihre Datenbank-Workloads. Wenn die aktuelle DB-Instance-Klasse beispielsweise db.r6gd.4xlarge ist, wählen Sie db.r6g.4xlarge aus, um zurückzuwechseln. Weitere Informationen finden Sie unter Ändern einer Amazon-DB-Instance.

Anwendungsfälle für Aurora-optimierte Lesevorgänge

Optimierter mehrstufiger Cache mit optimierten Lesevorgängen

Im Folgenden sind einige Anwendungsfälle aufgeführt, für die optimierte Lesevorgänge mit gestuftem Cache von Vorteil sein können:

- Anwendungen im Internet-Maßstab wie Zahlungsabwicklung, Rechnungsstellung und E-Commerce mit strikter Leistung. SLAs
- Dashboards für Berichte in Echtzeit, auf denen Hunderte von Einzelabfragen zur metrics/data Erfassung ausgeführt werden.
- Generative KI-Anwendungen mit der Erweiterung pgvector zur Suche nach exakten oder nächstgelegenen Nachbarn in Millionen von Vektoreinbettungen.

Temporäre Objekte mit optimierten Lesevorgängen

Im Folgenden sind einige Anwendungsfälle aufgeführt, für die optimierte Lesevorgänge von Vorteil sein können:

- Analytische Abfragen, die allgemeine Tabellenausdrücke (CTEs), abgeleitete Tabellen und Gruppierungsoperationen enthalten.
- Lesereplikate, die die nicht optimierten Abfragen für eine Anwendung verarbeiten.
- Bedarfsgesteuerte oder dynamische Berichtsabfragen mit komplexen Operationen wie GROUP BY und ORDER BY, für die nicht immer die entsprechenden Indizes verwendet werden können.

- `CREATE INDEX`- oder `REINDEX`-Operationen zum Sortieren
- Andere Workloads, die interne temporäre Tabellen verwenden.

Überwachen von DB-Instances, die Aurora-optimierte Lesevorgänge verwenden

Sie können Ihre Abfragen, die den mehrstufigen Cache mit optimierten Lesevorgängen zum Aktivieren von Lesevorgängen verwenden, wie im folgenden Beispiel dargestellt mit dem Befehl EXPLAIN überwachen:

```
Postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT c FROM sbtest15 WHERE id=100000000


QUERY PLAN
------------------------------------------------------------------------------------------
 Index Scan using sbtest15_pkey on sbtest15  (cost=0.57..8.59 rows=1 width=121) (actual
 time=0.287..0.288 rows=1 loops=1)
   Index Cond: (id = 100000000)
   Buffers: shared hit=3 read=2 aurora_orcache_hit=2
   I/O Timings: shared/local read=0.264
 Planning:
   Buffers: shared hit=33 read=6 aurora_orcache_hit=6
   I/O Timings: shared/local read=0.607
 Planning Time: 0.929 ms
 Execution Time: 0.303 ms
(9 rows)
Time: 2.028 ms
```

> **Note**
>
> Die Felder `aurora_orcache_hit` und `aurora_storage_read` im `Buffers`-Abschnitt des Erklärungsplans werden nur angezeigt, wenn optimierte Lesevorgänge aktiviert und ihre Werte größer als Null sind. Das read-Feld ist die Summe der Felder `aurora_orcache_hit` und `aurora_storage_read`.

Sie können DB-Instances, die Aurora Optimized Reads verwenden, anhand der folgenden CloudWatch Metriken überwachen:

- `AuroraOptimizedReadsCacheHitRatio`
- `FreeEphemeralStorage`
- `ReadIOPSEphemeralStorage`
- `ReadLatencyEphemeralStorage`
- `ReadThroughputEphemeralStorage`
- `WriteIOPSEphemeralStorage`
- `WriteLatencyEphemeralStorage`
- `WriteThroughputEphemeralStorage`

Diese Metriken liefern Daten über den verfügbaren Instance-Speicher, die IOPS und den Durchsatz. Weitere Informationen zu diesen Metriken finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

Sie können die `pg_proctab` Erweiterung auch zur NVMe Speicherüberwachung verwenden.

```
postgres=>select * from pg_diskusage();

major | minor |       devname       | reads_completed | reads_merged | sectors_read |
 readtime | writes_completed | writes_merged | sectors_written | writetime | current_io
 | iotime  | totaliotime
------+-------+---------------------+-----------------+--------------+--------------
+----------+------------------+---------------+-----------------+-----------
+------------+---------+-------------
      |       | rdstemp             |           23264 |            0 |       191450 |
  11670 |          1750892 |             0 |        24540576 |    819350 |          0 |
 3847580 |      831020
      |       | rdsephemeralstorage |           23271 |            0 |       193098 |
   2620 |           114961 |             0 |        13845120 |    130770 |          0 |
  215010 |      133410
(2 rows)
```

Bewährte Methoden für Aurora-optimierte Lesevorgänge

Nutzen Sie die folgenden bewährten Methoden für Aurora-optimierte Lesevorgänge:

- Überwachen Sie den im Instance-Speicher verfügbaren Speicherplatz anhand der CloudWatch Metrik`FreeEphemeralStorage`. Wenn der Instance-Speicher aufgrund des Workloads der DB-

Instance sein Limit erreicht, stellen Sie die Gleichzeitigkeit und die Abfragen ein, die intensiven Gebrauch von temporären Objekten machen, oder ändern Sie die Instance so, dass sie eine größere DB-Instance-Klasse verwendet.

- Überwachen Sie die CloudWatch Metrik im Hinblick auf die Cache-Trefferquote für optimierte Lesevorgänge. Operationen wie VACUUM ändern sehr schnell eine große Anzahl von Blöcken. Dies kann zu einem vorübergehenden Rückgang der Trefferrate führen. Die `pg_prewarm`-Erweiterung kann verwendet werden, um Daten in den Puffer-Cache zu laden, so dass Aurora einige dieser Blöcke proaktiv in den Cache der optimierten Lesevorgänge schreiben kann.
- Sie können das Cluster-Cache-Management (CCM) aktivieren, um den Puffer-Cache und den mehrstufigen Cache auf einem Tier-0-Reader aufzuwärmen, der als Failover-Ziel verwendet wird. Wenn CCM aktiviert ist, wird der Puffer-Cache regelmäßig gescannt, um Seiten, die bereinigt werden können, in den mehrstufigen Cache zu schreiben. Weitere Informationen zu CCM finden Sie unter Schnelle Wiederherstellung nach Failover mit der Cluster-Cacheverwaltung für Aurora PostgreSQL.

Optimieren von korrelierten Unterabfragen in Aurora PostgreSQL

Eine korrelierte Unterabfrage verweist auf Tabellenspalten aus der äußeren Abfrage. Sie wird einmal für jede Zeile ausgewertet, die von der äußeren Abfrage zurückgegeben wird. Im folgenden Beispiel verweist die Unterabfrage auf eine Spalte aus der Tabelle „ot“. Diese Tabelle ist nicht in der FROM-Klausel der Unterabfrage enthalten, wird aber in der FROM-Klausel der äußeren Abfrage referenziert. Wenn die Tabelle „ot“ 1 Million Zeilen enthält, muss die Unterabfrage 1 Million Mal ausgewertet werden.

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

> **Note**
>
> - Die Transformation einer Unterabfrage und der Unterabfragen-Cache sind in Aurora PostgreSQL ab Version 16.8 verfügbar, während Babelfish für Aurora PostgreSQL diese Funktionen ab Version 4.2.0 unterstützt.
> - Ab den Versionen 4.6.0 und 5.2.0 von Babelfish für Aurora PostgreSQL steuern die folgenden Parameter diese Funktionen:
>   - babelfishpg_tsql.apg_enable_correlated_scalar_transform
>   - babelfishpg_tsql.apg_enable_subquery_cache

Beide Parameter sind standardmäßig deaktiviert.

Verbessern der Aurora-PostgreSQL-Abfrageleistung mithilfe der Unterabfragentransformation

Aurora PostgreSQL kann korrelierte Unterabfragen beschleunigen, indem sie in äquivalente äußere Joins umgewandelt werden. Diese Optimierung gilt für die folgenden zwei Typen korrelierter Unterabfragen:

- Unterabfragen, die einen einzelnen Aggregatwert zurückgeben und in der SELECT-Liste aufgeführt sind

```
SELECT ot.a, ot.b, (SELECT AVG(it.b) FROM it WHERE it.a = ot.a) FROM ot;
```

- Unterabfragen, die einen einzelnen Aggregatwert zurückgeben und in einer WHERE-Klausel aufgeführt sind

```
SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT AVG(it.b) FROM it WHERE it.a = ot.a);
```

Aktivieren der Transformation in der Unterabfrage

Um die Transformation korrelierter Unterabfragen in äquivalente externe Joins zu ermöglichen, setzen Sie den `apg_enable_correlated_scalar_transform`-Parameter auf `ON`. Der Standardwert für diesen Parameter ist `OFF`.

Sie können die Cluster- oder Instance-Parametergruppe ändern, um diese Parameter festzulegen. Weitere Informationen hierzu finden Sie unter Parametergruppen für Amazon Aurora.

Alternativ können Sie die Einstellung mit dem folgenden Befehl ausschließlich für die aktuelle Sitzung konfigurieren:

```
SET apg_enable_correlated_scalar_transform TO ON;
```

Überprüfen der Transformation

Verwenden Sie den Befehl EXPLAIN, um zu überprüfen, ob die korrelierte Unterabfrage im Abfrageplan in einen externen Join umgewandelt wurde.

Wenn die Transformation aktiviert ist, wird der entsprechende korrelierte Teil der Unterabfrage in einen externen Join umgewandelt. Zum Beispiel:

```
postgres=> CREATE TABLE ot (a INT, b INT);
CREATE TABLE
postgres=> CREATE TABLE it (a INT, b INT);
CREATE TABLE

postgres=> SET apg_enable_correlated_scalar_transform TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);

                        QUERY PLAN
-------------------------------------------------------------
 Hash Join
   Hash Cond: (ot.a = apg_scalar_subquery.scalar_output)
   Join Filter: ((ot.b)::numeric < apg_scalar_subquery.avg)
   ->  Seq Scan on ot
   ->  Hash
         ->  Subquery Scan on apg_scalar_subquery
               ->  HashAggregate
                     Group Key: it.a
                     ->  Seq Scan on it
```

Dieselbe Abfrage wird nicht umgewandelt, wenn der GUC-Parameter auf `OFF` gesetzt wird. Der Plan hat keinen externen Join, sondern stattdessen einen untergeordneten Plan (Subplan).

```
postgres=> SET apg_enable_correlated_scalar_transform TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT
 AVG(it.b) FROM it WHERE it.a = ot.a);
                QUERY PLAN
---------------------------------------
 Seq Scan on ot
   Filter: ((b)::numeric < (SubPlan 1))
   SubPlan 1
     ->  Aggregate
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

Einschränkungen

- Die Unterabfrage muss in der SELECT-Liste oder in einer der Bedingungen der WHERE-Klausel enthalten sein. Andernfalls wird sie nicht umgewandelt.
- Die Unterabfrage muss eine Aggregatfunktion zurückgeben. Benutzerdefinierte Aggregatfunktionen unterstützen keine Transformation.
- Eine Unterabfrage, deren Return-Ausdruck keine einfache Aggregatfunktion ist, wird nicht umgewandelt.
- Die korrelierte Bedingung in den WHERE-Klauseln der Unterabfrage sollte ein einfacher Spaltenverweis sein. Andernfalls wird sie nicht umgewandelt.
- Die korrelierte Bedingung in den Where-Klauseln der Unterabfrage muss ein einfaches Gleichheitsprädikat sein.
- Die Unterabfrage darf weder eine HAVING- noch eine GROUP BY-Klausel enthalten.
- Die WHERE-Klausel in der Unterabfrage kann ein oder mehrere Prädikate in Kombination mit AND enthalten.

> **Note**
>
> Die Auswirkungen der Transformation auf die Leistung hängen von Ihrem Schema, Ihren Daten und Ihrem Workload ab. Die Ausführung korrelierter Unterabfragen mit Transformation kann die Leistung erheblich verbessern, da die Anzahl der Zeilen, die von der äußeren Abfrage erzeugt werden, zunimmt. Es wird dringend empfohlen, diese Funktion in einer Nicht-Produktionsumgebung mit Ihrem tatsächlichen Schema, Ihren Daten und Ihrem Workload zu testen, ehe Sie sie in einer Produktionsumgebung aktivieren.

Verwenden des Unterabfrage-Caches zur Verbesserung der Aurora-PostgreSQL-Abfrageleistung

Aurora PostgreSQL unterstützt einen Unterabfrage-Cache zum Speichern der Ergebnisse korrelierter Unterabfragen. Diese Funktion überspringt wiederholte Ausführungen korrelierter Unterabfragen, wenn sich die Ergebnisse der Unterabfragen bereits im Cache befinden.

Grundlegendes zu Unterabfragen

Der Memoize-Knoten von PostgreSQL ist der wichtigste Teil des Unterabfrage-Caches. Der Memoize-Knoten verwaltet eine Hash-Tabelle im lokalen Cache, um Eingabeparameterwerte den Abfrageergebniszeilen zuzuordnen. Das Speicherlimit für die Hash-Tabelle ergibt sich aus work_mem und hash_mem_multiplier. Weitere Informationen finden Sie unter Ressourcennutzung.

Während der Ausführung der Abfrage verwendet der Unterabfragen-Cache die Cache-Trefferrate (CHR), um abzuschätzen, ob der Cache die Abfrageleistung verbessert, und um zur Laufzeit der Abfrage zu entscheiden, ob der Cache weiterhin verwendet werden soll. Die CHR gibt das Verhältnis der Anzahl von Cache-Treffern zur Gesamtzahl der Anfragen an. Wenn beispielsweise eine korrelierte Unterabfrage 100 Mal ausgeführt werden muss und 70 dieser Ausführungsergebnisse aus dem Cache abgerufen werden können, beträgt die CHR 0,7.

Für jede apg_subquery_cache_check_interval-Anzahl von Cache-Fehlern wird der Nutzen des Unterabfrage-Caches ausgewertet, indem geprüft wird, ob die CHR größer ist als apg_subquery_cache_hit_rate_threshold. Wenn nicht, wird der Cache aus dem Speicher gelöscht und die Abfrageausführung kehrt zur ursprünglichen, nicht zwischengespeicherten erneuten Ausführung der Unterabfrage zurück.

Parameter, die das Verhalten des Unterabfrage-Caches steuern

In der folgenden Tabelle sind die Parameter aufgeführt, die das Verhalten des Unterabfrage-Caches steuern.

Parameter	Beschreibung	Standard	Zulässig
apg_enable_subquery_cache	Aktiviert die Verwendung von Cache für korrelierte skalare Unterabfragen	OFF	ON, OFF
apg_subquery_cache_check_interval	Legt die Häufigkeit in Anzahl der Cache-Fehler fest, um die Unterabfragen-CHR zu bewerten.	500	0 – 2147483647

Parameter	Beschreibung	Standard	Zulässig
apg_subquery_cache_hit_rate_threshold	Legt den Schwellenwert für die Unterabfragen-CHR fest.	0.3	0,0 bis 1,0

> **Note**
>
> - Höhere Werte von `apg_subquery_cache_check_interval` können die Genauigkeit der CHR-basierten Einschätzung des Cache-Nutzens verbessern, erhöhen aber den Cache-Overhead, da die CHR erst ausgewertet wird, wenn die Cache-Tabelle `apg_subquery_cache_check_interval`-Zeilen enthält.
> - Höhere Werte von `apg_subquery_cache_hit_rate_threshold` bedeuten, dass der Unterabfrage-Cache aufgegeben wurde und zur ursprünglichen, erneuten Ausführung der Unterabfrage ohne Zwischenspeicherung zurückgekehrt wird.

Sie können die Cluster- oder Instance-Parametergruppe ändern, um diese Parameter festzulegen. Weitere Informationen hierzu finden Sie unter Parametergruppen für Amazon Aurora.

Alternativ können Sie die Einstellung mit dem folgenden Befehl ausschließlich für die aktuelle Sitzung konfigurieren:

```
SET apg_enable_subquery_cache TO ON;
```

Aktivieren des Unterabfrage-Caches in Aurora PostgreSQL

Wenn der Unterabfrage-Cache aktiviert ist, nutzt Aurora PostgreSQL Cache, um die Ergebnisse der Unterabfrage zu speichern. Der Abfrageplan hat dann einen Memoize-Knoten unterhalb von SubPlan.

Die folgende Befehlssequenz zeigt beispielsweise den geschätzten Abfrageausführungsplan einer einfachen korrelierten Unterabfrage ohne Unterabfrage-Cache.

```
postgres=> SET apg_enable_subquery_cache TO OFF;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);
```

```
             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Seq Scan on it
           Filter: (a = ot.a)
```

Nach dem Aktivieren von `apg_enable_subquery_cache` enthält der Abfrageplan einen Memoize-Knoten unter dem SubPlan-Knoten, der darauf hinweist, dass die Unterabfrage beabsichtigt, den Cache zu verwenden.

```
postgres=> SET apg_enable_subquery_cache TO ON;
SET
postgres=> EXPLAIN (COSTS FALSE) SELECT ot.a, ot.b FROM ot WHERE ot.b < (SELECT it.b
 FROM it WHERE it.a = ot.a);

             QUERY PLAN
-----------------------------------
 Seq Scan on ot
   Filter: (b < (SubPlan 1))
   SubPlan 1
     ->  Memoize
           Cache Key: ot.a
           Cache Mode: binary
           ->  Seq Scan on it
                 Filter: (a = ot.a)
```

Der eigentliche Ausführungsplan für die Abfrage enthält weitere Details zum Unterabfrage-Cache, einschließlich Cache-Treffer und Cache-Fehler. Die folgende Ausgabe zeigt den tatsächlichen Abfrageausführungsplan der obigen Beispielabfrage nach dem Einfügen einiger Werte in die Tabellen.

```
postgres=> EXPLAIN (COSTS FALSE, TIMING FALSE, ANALYZE TRUE) SELECT ot.a, ot.b FROM ot
 WHERE ot.b < (SELECT it.b FROM it WHERE it.a = ot.a);
              QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on ot (actual rows=2 loops=1)
   Filter: (b < (SubPlan 1))
   Rows Removed by Filter: 8
   SubPlan 1
     ->  Memoize (actual rows=0 loops=10)
```

```
          Cache Key: ot.a
          Cache Mode: binary
          Hits: 4  Misses: 6  Evictions: 0  Overflows: 0  Memory Usage: 1kB
          ->  Seq Scan on it (actual rows=0 loops=6)
                Filter: (a = ot.a)
                Rows Removed by Filter: 4
```

Die Gesamtzahl der Cache-Treffer ist 4 und die Gesamtzahl der Cache-Fehler beträgt 6. Wenn die Gesamtzahl der Treffer und Fehler geringer ist als die Anzahl der Schleifen im Memoize-Knoten, bedeutet das, dass die CHR-Evaluierung nicht erfolgreich war und der Cache an irgendeinem Punkt bereinigt und aufgegeben wurde. Hinsichtlich der Ausführung der Unterabfrage wurde dann die ursprüngliche, erneute Ausführung ohne Zwischenspeicherung wiederaufgenommen.

Einschränkungen

Der Unterabfrage-Cache unterstützt bestimmte Muster korrelierter Unterabfragen nicht. Diese Arten von Abfragen werden ohne Cache ausgeführt, auch wenn der Unterabfrage-Cache aktiviert ist:

- Korrelierte IN/EXISTS/ANY/ALL-Unterabfragen
- Korrelierte Unterabfragen, die nicht deterministische Funktionen enthalten
- Korrelierte Unterabfragen, die auf äußere Tabellenspalten mit Datentypen verweisen, die keine Hashing- oder Gleichheitsoperationen unterstützen.

Verbessern der Abfrageleistung mit Adaptive Join

Übersicht

Adaptive Join ist eine Vorschaufunktion in Aurora PostgreSQL 17.4, die zur Verbesserung der Abfrageleistung beiträgt. Diese Funktion ist standardmäßig deaktiviert, Sie können sie jedoch mithilfe von GUC-Parametern (Global User Configuration) aktivieren. Da es sich um eine Vorschaufunktion handelt, können sich die Standardparameterwerte ändern. Wenn diese Option aktiviert ist, trägt Adaptive Join zur Optimierung der Abfrageleistung bei, indem zur Laufzeit dynamisch von einem Nested-Loop-Join zu einem Hash-Join gewechselt wird. Dieser Wechsel erfolgt, wenn der PostgreSQL-Optimierer aufgrund ungenauer Kardinalitätsschätzungen fälschlicherweise einen Nested-Loop-Join ausgewählt hat.

Konfigurieren von Adaptive Join

Sie können Adaptive Join über diese drei GUC-Parameter steuern:

Konfigurationsparameter für Adaptive Join

GUC-Parameter	Beschreibung	Standard-Konfigurationsoptionen
apg_adaptive_join_crossover_multiplier	Dieser Multiplikator bestimmt zusammen mit dem Zeilenknotenpunkt, wann von einem verschachtelten Loop zu einem Hash-Join gewechselt werden soll. Am Zeilenübergangspunkt schätzt der SQL-Optimierer, dass Vorgänge für verschachtelte Schleifen und Hash-Joins dieselben Kosten verursachen. Ein höherer Multiplikatorwert verringert die Wahrscheinlichkeit, dass Adaptive Join zu einem Hash-Join wechselt.	Steuert, ob Adaptive Join aktiviert ist • Standardwert: -1 (deaktiviert) • Gültiger Bereich: -1 bis DBL_MAX • Zum Aktivieren: Auf >= 1 festlegen
apg_adaptive_join_cost_threshold	Dieser Parameter legt einen Mindestschwellenwert für Abfragekosten fest. Adaptive Join deaktiviert sich automatisch für Abfragen, die unter diesem Schwellenwert liegen. Dadurch wird ein Leistungsaufwand bei einfachen Abfragen vermieden, bei denen die Kosten für die Planung eines adaptiven Joins die Vorteile des Wechsels von Nested Loop zu Hash-Join übersteigen könnten.	Legt den Mindestschwellenwert für Abfragekosten fest • Standardwert: 100 • Gültiger Bereich: 0 bis DBL_MAX
apg_enable_parameterized_adaptive_join	Dieser Parameter erweitert Adaptive Join-Funktionen auf parametrisierte Nested Loop Joins, wenn er aktiviert ist. Standardmäßig funktioniert Adaptive Join nur mit nicht parametrisierten Nested Loop Joins, da diese eher von der Umstellung auf Hash-	Steuert das Adaptive-Join-Verhalten für Nested Loop Joins • Standardwert: false • Zulässige Werte: true/false

GUC-Parameter	Beschreibung	Standard-Konfigurationsoptionen
	Join profitieren. Parametrisierte Nested Loop Joins schneiden in der Regel besser ab, sodass die Umstellung auf Hash-Join weniger kritisch ist.	• Bei „false“: Funktioniert nur mit nicht parametrisierten Nested Loop Joins • Wenn wahr: Funktioniert sowohl mit parametrisierten als auch mit nicht parametrisierten Nested Loop Joins **Note** `apg_adaptive_join_crossover_multiplier` muss zuerst aktiviert werden

Arbeiten mit Tabellen ohne Protokollierung in Aurora PostgreSQL

Amazon Aurora PostgreSQL unterstützt Tabellen ohne Protokollierung, die absturzsicher sind und die Datenintegrität auch nach Ausfällen oder Failover der Writer-Instance wahren. Beim Standard-PostgreSQL umgehen Tabellen ohne Protokollierung das Write-Ahead-Protokoll (WAL) bei Schreibvorgängen, was zu höheren Schreibgeschwindigkeiten führt. Dies führt jedoch zu einer geringeren Haltbarkeit, da Tabellen ohne Protokollierung nicht absturzsicher sind und nach einem Systemausfall oder unsachgemäßem Herunterfahren Daten verloren gehen können. Diese Tabellen ohne Protokollierung werden nach einem Absturz oder einem unsauberen Herunterfahren automatisch gekürzt. Ihr Inhalt und ihre Indizes werden auch nicht auf Standby-Server repliziert.

Im Gegensatz dazu behandelt Aurora PostgreSQL Tabellen ohne Protokollierung aufgrund seiner verteilten Speicherarchitektur anders. Dies liegt daran, dass das Speichersystem von Aurora aus Gründen der Haltbarkeit nicht auf das traditionelle PostgreSQL-WAL angewiesen ist. Die Leistungsvorteile, die normalerweise mit Tabellen ohne Protokollierung in Standard-PostgreSQL-Versionen einhergehen, sind in Aurora jedoch möglicherweise nicht so signifikant. Dies ist auf die verteilte Speicherarchitektur von Aurora zurückzuführen, die im Vergleich zum lokalen Speicher, der in Standard-PostgreSQL-Versionen verwendet wird, zusätzlichen Overhead mit sich bringen kann.

Wenn Sie Tabellen ohne Protokollierung in Aurora PostgreSQL verwenden, beachten Sie Folgendes:

- Sie können auf Tabellen ohne Protokollierung nur vom Writer-Knoten im Aurora-DB-Cluster aus zugreifen.
- Reader-Knoten können nur dann auf Tabellen ohne Protokollierung zugreifen, wenn sie auf den Writer-Status hochgestuft werden.
- Wenn Sie versuchen, von einem Reader-Knoten aus auf Tabellen ohne Protokollierung zuzugreifen, wird folgende Fehlermeldung angezeigt:

  ```
  cannot access temporary or unlogged relations during recovery.
  ```

Erstellen von Tabellen ohne Protokollierung

Zum Erstellen einer Tabelle ohne Protokollierung in Aurora PostgreSQL fügen Sie das Schlüsselwort UNLOGGED zu Ihrer CREATE TABLE-Anweisung hinzu:

```
CREATE UNLOGGED TABLE staging_sales_data (
    transaction_id bigint,
    customer_id bigint,
    product_id bigint,
    transaction_date date,
    amount NUMERIC
);
```

Handhaben von Tabellen ohne Protokollierung während der Migration

Bei der Vorbereitung der Migration von Daten zu Aurora PostgreSQL ist es wichtig, Tabellen ohne Protokollierung zu identifizieren und angemessen zu handhaben. Bei Tabellen ohne Protokollierung

findet keine WAL-Protokollierung statt. Zudem sind sie vom Replikations-Stream ausgeschlossen, was bedeutet, dass sie nicht auf ein Aurora-Lesereplikat repliziert werden.

Konvertieren Sie Tabellen ohne Protokollierung in Tabellen mit Protokollierung oder löschen Sie sie, wenn sie nicht verwendet werden, bevor Sie ein Aurora-Lesereplikat erstellen.

Verwenden Sie den folgenden Befehl, um zu überprüfen, ob auf den einzelnen Datenbanken auf der Instance Tabellen ohne Protokollierung vorhanden sind:

```
SELECT oid, relfilenode, relname, relpersistence, relkind
FROM pg_class
WHERE relpersistence ='u';
```

Um eine Tabelle ohne Protokollierung in eine Tabelle mit Protokollierung zu konvertieren, verwenden Sie folgenden Befehl:

```
ALTER TABLE table_name SET LOGGED;
```

Dieser Vorgang schreibt die gesamte Tabelle neu und sperrt sie bis zum Abschluss exklusiv. Bei großen Tabellen kann dies zu erheblichen Ausfallzeiten führen.

Konvertieren von Tabellen ohne Protokollierung in Tabellen mit Protokollierung

Wenn Sie eine Tabelle ohne Protokollierung in eine Tabelle mit Protokollierung konvertieren müssen, verwenden Sie folgenden Befehl:

```
ALTER TABLE table_name SET LOGGED;
```

Bei diesem Vorgang wird die gesamte Tabelle neu geschrieben und eine exklusive Sperre für sie eingerichtet, bis der Vorgang abgeschlossen ist. Bei großen Tabellen kann dies zu erheblichen Ausfallzeiten führen.

Tabellen ohne Protokollierung und logische Replikation

Tabellen ohne Protokollierung sind im Allgemeinen nicht in der logischen Replikation enthalten, da die logische Replikation zum Erfassen und Übertragen von Änderungen auf das WAL angewiesen ist. Standardmäßig findet bei Änderungen an Tabellen ohne Protokollierung keine WAL-Protokollierung statt. Zudem sind sie vom Replikations-Stream ausgeschlossen, sodass sie für Anwendungsfälle, die eine logische Replikation erfordern, nicht geeignet sind. Aurora PostgreSQL bietet jedoch einen Parameter namens `rds.logically_replicate_unlogged_tables`, mit dem Sie dieses Verhalten steuern können:

- Wenn `rds.logically_replicate_unlogged_tables` auf 0 (off) gesetzt ist, werden die Tabellen ohne Protokollierung von der logischen Replikation ausgeschlossen.
- Wenn `rds.logically_replicate_unlogged_tables` auf 1 (on) gesetzt ist, werden die Tabellen ohne Protokollierung in die logische Replikation aufgenommen.

> **Note**
>
> In Aurora PostgreSQL ist der `rds.logically_replicate_unlogged_tables`-Parameter in den Versionen 14 und niedriger standardmäßig auf 1 (on) und in den Versionen 15 und höher auf 0 (off) gesetzt.

Arbeiten mit der PostgreSQL-Selbstbereinigung in Amazon Aurora PostgreSQL

Wir empfehlen ausdrücklich die Selbstbereinigungsfunktion zu verwenden, um die Integrität Ihrer PostgreSQL-DB-Instance zu wahren. Die Selbstbereinigung automatisiert den Start der Befehle VACUUM und ANALYZE. Sie prüft auf Tabellen mit einer großen Zahl von eingefügten, aktualisierten oder gelöschten Tupeln. Nach dieser Prüfung wird Speicher durch Entfernen von überflüssigen Daten oder Tupeln aus der PostgreSQL-Datenbank freigegeben.

Standardmäßig ist die Selbstbereinigungsfunktion auf den DB-Instances für Aurora PostgreSQL aktiviert, die Sie mit einer der folgenden PostgreSQL-DB-Parametergruppe erstellen. Andere Konfigurationsparameter, die mit der Selbstbereinigungsfunktion verknüpft sind, sind ebenfalls standardmäßig festgelegt. Da diese Standardwerte generisch sind, können Sie davon profitieren,

einige der mit der Selbstbereinigungsfunktion verbundenen Parameter für Ihre spezifische Workload zu optimieren.

Im Folgenden finden Sie weitere Informationen zur Selbstbereinigung und dazu, wie Sie einige ihrer Parameter auf Ihrer DB-Instance von Aurora PostgreSQL aktivieren.

Themen

- Zuweisen von Arbeitsspeicher für die Selbstbereinigung
- Verringern der Wahrscheinlichkeit von Transaktions-ID-Wraparounds
- Ermittlung, ob die Tabellen in Ihrer Datenbank bereinigt werden müssen
- Ermittlung, für welche Tabellen derzeit eine Selbstbereinigung nötig ist
- Ermittlung, ob die Selbstbereinigung derzeit ausgeführt wird und wie lange sie dauert
- Ausführen einer manuellen Bereinigungseinfrierung
- Neuindizierung einer Tabelle während der Ausführung einer Selbstbereinigung
- Verwalten der automatischen Bereinigung mit großen Indizes
- Weitere Parameter, die sich auf die Selbstbereinigung auswirken
- Festlegen von Selbstbereinigungsparametern auf Tabellenebene
- Protokollieren von Selbstbereinigung- und Bereinigungsaktivitäten
- Verstehen des Verhaltens der Selbstbereinigung bei ungültigen Datenbanken
- Identifizieren und Beheben von Problemen mit Blockern aggressiver Bereinigungen in Aurora PostgreSQL

Zuweisen von Arbeitsspeicher für die Selbstbereinigung

Einer der wichtigsten Parameter, der sich auf die Leistung der Selbstbereinigung auswirkt, ist der Parameter `autovacuum_work_mem`. In Aurora PostgreSQL, Versionen 14 und früher, ist der `autovacuum_work_mem`-Parameter auf -1 gesetzt und zeigt somit an, dass stattdessen die `maintenance_work_mem`-Einstellung verwendet wird. Bei allen anderen Versionen wird `autovacuum_work_mem` durch GREATEST({DBInstanceClassMemory/32768}, 65536) bestimmt.

Manuelle Bereinigungsvorgänge verwenden immer die `maintenance_work_mem`-Einstellung mit einer Standardeinstellung von GREATEST({DBInstanceClassMemory/63963136*1024}, 65536). Dies kann auch auf Sitzungsebene mit dem `SET`-Befehl für gezieltere manuelle `VACUUM`-Operationen angepasst werden.

`autovacuum_work_mem` bestimmt den Speicher der automatischen Bereinigung zum Speichern von IDs toter Tupeln (`pg_stat_all_tables.n_dead_tup`) für Bereinigungsindizes.

Beachten Sie Folgendes, wenn Sie Berechnungen durchführen, um den Wert des `autovacuum_work_mem`-Parameters bestimmen:

- Wenn Sie den Wert des Parameters zu niedrig einstellen, muss die Tabelle während des Bereinigungsvorgangs möglicherweise mehrmals gescannt werden, damit der Vorgang abgeschlossen werden kann. Solche wiederholten Scans können negative Auswirkungen auf die Leistung haben. Bei größeren Instances kann das Festlegen von `maintenance_work_mem` oder `autovacuum_work_mem` auf mindestens 1 GB die Leistung bei der Bereinigung von Tabellen mit einer hohen Anzahl von toten Tupeln verbessern. In den PostgreSQL-Versionen 16 und früher ist die Speichernutzung für die Bereinigung jedoch auf 1 GB begrenzt, was ausreicht, um ungefähr 179 Millionen tote Tupel in einem einzigen Durchgang zu verarbeiten. Wenn eine Tabelle mehr tote Tupel enthält, sind mehrere Bereinigungsdurchläufe durch die Indizes der Tabelle erforderlich, was den Zeitaufwand erheblich erhöht. Ab PostgreSQL-Version 17 gibt es kein Limit von 1 GB mehr und die Selbstbereinigung kann mithilfe von Radixbäumen mehr als 179 Millionen Tupel verarbeiten.

  Ein Tupel-Bezeichner hat eine Größe von 6 Bytes. Um den Speicherbedarf für das Bereinigen eines Tabellenindex zu schätzen, fragen Sie `pg_stat_all_tables.n_dead_tup` ab, um die Anzahl der toten Tupel zu ermitteln. Multiplizieren Sie diese Zahl dann mit 6, um den Speicherbedarf zu bestimmen, der für das Bereinigen des Index in einem einzigen Durchgang erforderlich ist. Sie können die folgende Abfrage verwenden:

  ```
  SELECT
      relname AS table_name,
      n_dead_tup,
      pg_size_pretty(n_dead_tup * 6) AS estimated_memory
  FROM
      pg_stat_all_tables
  WHERE
      relname = 'name_of_the_table';
  ```

- Der Parameter `autovacuum_work_mem` funktioniert in Verbindung mit dem Parameter `autovacuum_max_workers`. Jeder Worker aus `autovacuum_max_workers` kann den von Ihnen zugeteilten Arbeitsspeicher nutzen. Wenn Sie viele kleine Tabellen haben, müssen Sie mehr `autovacuum_max_workers` und weniger `autovacuum_work_mem` zuteilen. Wenn Sie große Tabellen haben (größer als 100 GB), sollten Sie mehr Arbeitsspeicher und weniger Worker-

Prozesse zuteilen. Sie müssen genügend Arbeitsspeicher zuweisen, um den Vorgang für Ihre größte Tabelle erfolgreich ausführen zu können. Daher muss die Kombination aus Worker-Prozessen und Arbeitsspeicher dem gesamten Arbeitsspeicher entsprechen, den Sie zuteilen möchten.

Verringern der Wahrscheinlichkeit von Transaktions-ID-Wraparounds

In einigen Fällen sind Parametergruppen-Einstellungen, die sich auf die Selbstbereinigung beziehen, möglicherweise nicht aggressiv genug, um Transaktions-ID-Wraparounds zu verhindern. Um dieses Problem anzugehen, stellt Aurora PostgreSQL eine Methode bereit, mit der die Selbstbereinigungsparameter automatisch angepasst werden. Die adaptive Selbstbereinigung ist ein Feature für Aurora PostgreSQL. In der PostgreSQL-Dokumentation finden Sie eine sehr detaillierte Beschreibung von Transaktions-ID-Wraparounds.

Die adaptive Selbstbereinigung ist standardmäßig für Instances von Aurora PostgreSQL aktiviert, wobei der dynamische Parameter `rds.adaptive_autovacuum` auf ON gesetzt ist. Wir raten dringend dazu, diese Option aktiviert zu lassen. Um die adaptive Optimierung der Selbstbereinigungsparameter zu deaktivieren, stellen Sie den Parameter `rds.adaptive_autovacuum` jedoch auf 0 oder OFF ein.

Transaktions-ID-Wraparounds können selbst dann noch auftreten, wenn Aurora Amazon RDS die Selbstbereinigungsparameter optimiert. Wir raten Ihnen dazu, einen Amazon CloudWatch-Alarm für Transaktions-ID-Wraparounds zu implementieren. Weitere Informationen finden Sie im Beitrag Implement an Early Warning System for Transaction ID Wraparound in RDS für PostgreSQL im AWS-Database-Blog.

Wenn die adaptive Optimierung der Selbstbereinigungsparameter aktiviert ist, beginnt Amazon RDS mit dem Anpassen der Selbstbereinigungsparameter, wenn die CloudWatch-Metrik `MaximumUsedTransactionIDs` den Wert des `autovacuum_freeze_max_age`-Parameters oder 500 000 000 erreicht, je nachdem, welcher Wert größer ist.

Amazon RDS fährt mit dem Anpassen der Parameter für die Selbstbereinigung fort, wenn eine Tabelle weiterhin zu Transaktions-ID-Wraparounds tendiert. Jede dieser Anpassungen stellt weitere Ressourcen für die Selbstbereinigung bereit, um Wraparounds zu vermeiden. Amazon RDS aktualisiert die folgenden Parameter, die sich auf die Selbstbereinigung beziehen:

- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit

- autovacuum_work_mem
- autovacuum_naptime

RDS ändert diese Parameter nur, wenn der neue Wert die Selbstbereinigung aggressiver macht. Die Parameter werden im Arbeitsspeicher auf der DB-Instance geändert. Die Werte in der Parametergruppe werden nicht geändert. Um die aktuellen Arbeitsspeichereinstellungen anzuzeigen, verwenden Sie den PostgreSQL-SQL-Befehl SHOW.

Wenn Amazon RDS einen dieser Selbstbereinigungsparameter ändert, wird ein Ereignis für die betroffene DB-Instance erzeugt. Dieses Ereignis ist auf der AWS-Managementkonsole und über die Amazon-RDS-API sichtbar. Nachdem der Schwellenwert der zurückgegebenen CloudWatch-Metrik `MaximumUsedTransactionIDs` unterschritten wurde, setzt Amazon RDS die sich auf die Selbstbereinigung beziehenden Parameter auf die in der Parametergruppe angegebenen Werte zurück. Es generiert dann ein anderes Ereignis, das dieser Änderung entspricht.

Ermittlung, ob die Tabellen in Ihrer Datenbank bereinigt werden müssen

Sie können die folgende Abfrage verwenden, um die Anzahl der nicht eingefrorenen Transaktionen in einer Datenbank anzuzeigen. Die `datfrozenxid`-Spalte einer `pg_database`-Zeile der Datenbank ist eine Untergrenze für die normalen Transaktions-IDs, die in dieser Datenbank erscheinen. Diese Spalte ist der Mindestwert der `relfrozenxid`-Werte pro Tabelle in der Datenbank.

```
SELECT datname, age(datfrozenxid) FROM pg_database ORDER BY age(datfrozenxid) desc
 limit 20;
```

Beispielsweise könnten die Ergebnisse der Ausführung der oben gezeigten Abfrage wie folgt aussehen.

```
datname    | age
mydb       | 1771757888
template0  | 1721757888
template1  | 1721757888
rdsadmin   | 1694008527
postgres   | 1693881061
(5 rows)
```

Wenn das Alter einer Datenbank 2 Milliarden Transaktions-IDs erreicht, treten Transaktions-ID (XID)-Wraparounds auf und die Datenbank wird als schreibgeschützt festgelegt. Sie können diese Abfrage

verwenden, um eine Metrik zu erstellen und einige Male am Tag auszuführen. Standardmäßig ist die Selbstbereinigung so festgelegt, dass das Alter der Transaktionen 200 000 000 nicht überschreitet 200,000,000 (`autovacuum_freeze_max_age`).

Eine Überwachungsstrategie kann beispielsweise wie folgt aussehen:

- Stellen Sie den `autovacuum_freeze_max_age`-Wert auf 200 Millionen Transaktionen ein.
- Wenn eine Tabelle 500 Millionen nicht eingefrorene Transaktionen hat, wird ein Alarm mit niedrigem Schweregrad ausgelöst. Dies ist kein unangemessener Wert. Er könnte jedoch zu erkennen geben, dass die Selbstbereinigung nicht Schritt hält.
- Wenn eine Tabelle 1 Milliarde nicht bereinigter Transaktionen aufweist, sollte dies als ein Alarm behandelt werden, für den Maßnahmen zu ergreifen sind. Im Allgemeinen sollte aus Leistungsgründen das Alter möglichst nahe zu `autovacuum_freeze_max_age` liegen. Wir empfehlen, dass Sie eine Untersuchung unter Beachtung der folgenden Empfehlungen durchführen.
- Wenn eine Tabelle 1,5 Milliarden nicht bereinigter Transaktionen erreicht, wird ein Alarm mit hohem Schweregrad ausgelöst. Abhängig davon, wie schnell Ihre Datenbank Transaktions-IDs verwendet, kann dieser Alarm zu erkennen geben, dass die Zeit für eine Selbstbereinigung des Systems abläuft. In diesem Fall empfehlen wir, eine unmittelbare Lösung in Betracht zu ziehen.

Wenn eine Tabelle diese Schwellenwerte konstant überschreitet, ändern Sie die Selbstbereinigungsparameter weiter. Standardmäßig ist die manuelle Verwendung von VACUUM (für den kostenbasierte Verzögerungen deaktiviert sind) aggressiver als die Standardselbstbereinigung. Der Befehl hat jedoch insgesamt auch größere Auswirkungen auf das System.

Wir empfehlen Folgendes:

- Aktivieren Sie einen Überwachungsmechanismus, damit Sie über das Alter der ältesten Transaktionen informiert sind.

  Informationen zum Erstellen eines Prozesses, der Sie über Transaktions-ID-Wraparounds benachrichtigt, finden Sie im AWS-Datenbank-Blogbeitrag Implementieren eines Frühwarnsystem für den Transaktions-ID-Wraparound in Amazon RDS für PostgreSQL.
- Führen Sie für häufiger verwendete Tabellen zusätzlich zur Selbstbereinigung während Wartungsfenstern regelmäßig eine manuelle Bereinigungseinfrierung aus. Informationen zur Ausführung manueller Bereinigungseinfrierungen finden Sie unter Ausführen einer manuellen Bereinigungseinfrierung.

Ermittlung, für welche Tabellen derzeit eine Selbstbereinigung nötig ist

Häufig benötigen eine oder zwei Tabellen eine Bereinigung. Tabellen, deren `relfrozenxid`-Wert größer als die Anzahl von Transaktionen in `autovacuum_freeze_max_age` ist, sind stets Ziel der Selbstbereinigung. Wenn andernfalls die Anzahl der Tupeln, die seit dem letzten VACUUM-Befehl veraltet sind, den Bereinigungsschwellenwert überschreitet, wird die Tabelle bereinigt.

Der Selbstbereinigungsschwellenwert ist definiert als:

```
Vacuum-threshold = vacuum-base-threshold + vacuum-scale-factor * number-of-tuples
```

wobei `vacuum base threshold` `autovacuum_vacuum_threshold` ist, `vacuum scale factor` `autovacuum_vacuum_scale_factor` ist und `number of tuples` `pg_class.reltuples` ist.

Führen Sie während der Herstellung der Verbindung mit Ihrer Datenbank die folgende Abfrage aus, um eine Liste der Tabellen anzuzeigen, für die Selbstbereinigungsfunktion eine Bereinigung als notwendig betrachtet.

```
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM
pg_settings WHERE name = 'autovacuum_vacuum_threshold'),
vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM
pg_settings WHERE name = 'autovacuum_vacuum_scale_factor'),
fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE name =
 'autovacuum_freeze_max_age'),
sto AS (select opt_oid, split_part(setting, '=', 1) as param,
split_part(setting, '=', 2) as value from (select oid opt_oid, unnest(reloptions)
 setting from pg_class) opt)
SELECT '"'||ns.nspname||'"."'||c.relname||'"' as relation,
pg_size_pretty(pg_table_size(c.oid)) as table_size,
age(relfrozenxid) as xid_age,
coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age,
(coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples)
AS autovacuum_vacuum_tuples, n_dead_tup as dead_tuples FROM
pg_class c join pg_namespace ns on ns.oid = c.relnamespace
join pg_stat_all_tables stat on stat.relid = c.oid join vbt on (1=1) join vsf on (1=1)
 join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid =
 cvbt.opt_oid
```

```
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid =
 cvsf.opt_oid
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog'
AND (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
OR coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
c.reltuples <= n_dead_tup)
ORDER BY age(relfrozenxid) DESC LIMIT 50;
```

Ermittlung, ob die Selbstbereinigung derzeit ausgeführt wird und wie lange sie dauert

Wenn Sie eine Tabelle manuell bereinigen müssen, müssen Sie ermitteln, ob zurzeit eine Selbstbereinigung ausgeführt wird. Wenn dies der Fall ist, müssen Sie die Parameter anpassen, damit sie effizienter ausgeführt wird, oder die Selbstbereinigung temporär beenden, damit Sie den Befehl VACUUM manuell ausführen können.

Verwenden Sie die folgende Abfrage, um zu ermitteln, ob die Selbstbereinigung ausgeführt wird, wie lange diese bereits dauert und ob diese auf eine andere Sitzung wartet.

```
SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query
FROM pg_stat_activity
WHERE upper(query) LIKE '%VACUUM%'
ORDER BY xact_start;
```

Nach dem Ausführen der Abfrage wird Ihnen eine Ausgabe ähnlich der folgenden angezeigt.

```
 datname | usename  |  pid  | state  | wait_event |      xact_runtime       | query
 --------+----------+-------+--------+------------+-------------------------
+-------------------------------------------------------------------------------------------
 mydb    | rdsadmin | 16473 | active |            | 33 days 16:32:11.600656 |
 autovacuum: VACUUM ANALYZE public.mytable1 (to prevent wraparound)
 mydb    | rdsadmin | 22553 | active |            | 14 days 09:15:34.073141 |
 autovacuum: VACUUM ANALYZE public.mytable2 (to prevent wraparound)
 mydb    | rdsadmin | 41909 | active |            | 3 days 02:43:54.203349  |
 autovacuum: VACUUM ANALYZE public.mytable3
```

```
mydb    | rdsadmin |   618 | active |            | 00:00:00                |
SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
xact_runtime, query+
        |          |       |        |            |                         | FROM
pg_stat_activity
          +
        |          |       |        |            |                         | WHERE
query like '%VACUUM%'
         +
        |          |       |        |            |                         | ORDER BY
xact_start;
      +
```

Verschiedene Probleme können dazu führen, dass eine Selbstbereinigungssitzung über eine lange Zeit (mehrere Tage) ausgeführt wird. Das häufigste Problem besteht jedoch darin, dass der Wert Ihres Parameters `maintenance_work_mem` im Verhältnis zur Größe der Tabelle oder zur Häufigkeit der Aktualisierungen zu niedrig festgelegt wurde.

Es wird empfohlen, dass Sie die folgende Formel verwenden, um den Wert des Parameters `maintenance_work_mem` festzulegen.

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

Selbstbereinigungssitzungen, die nur kurze Zeit ausgeführt werden, können ebenfalls auf Probleme hinweisen:

- Es kann bedeuten, dass es für Ihren Workload nicht genügend `autovacuum_max_workers` gibt. In diesem Fall müssen Sie die Anzahl der Worker angeben.
- Es kann bedeuten, dass es eine Indexbeschädigung gibt (die Selbstbereinigung stürzt ab und startet bei derselben Relation neu, es gibt jedoch keine Fortschritte). Führen Sie in diesem Fall ein manuelles `vacuum freeze verbose` *`table`* aus, um die genaue Ursache zu sehen.

Ausführen einer manuellen Bereinigungseinfrierung

Möglicherweise möchten Sie eine manuelle Bereinigung für eine Tabelle ausführen, für die bereits ein Bereinigungsvorgang ausgeführt wird. Dies ist nützlich, wenn Sie eine Tabelle mit einem Alter identifiziert haben, das 2 Milliarden Transaktionen (oder einen Wert oberhalb eines von Ihnen beobachteten Schwellenwerts) erreicht.

Die folgenden Schritte sind Richtlinien mit mehreren Variationen des Prozesses. Angenommen, Sie stellen während der Überprüfung fest, dass der Wert für den Parameter `maintenance_work_mem` zu niedrig festgelegt wurde. Sie müssen sofort Maßnahmen für eine Tabelle einleiten. Sie möchten zum aktuellen Zeitpunkt jedoch keinen Bounce für die Instance auslösen. Ermitteln Sie mittels der Abfragen in vorherigen Abschnitten die Tabelle, die das Problem darstellt, und identifizieren Sie eine über einen langen Zeitraum ausgeführte Selbstbereinigungssitzung. Sie müssen die Einstellung für den Parameter `maintenance_work_mem` ändern. Sie müssen jedoch auch sofortige Maßnahmen einleiten und die betreffende Tabelle bereinigen. Das folgende Verfahren zeigt, was Sie in dieser Situation unternehmen.

So führen Sie manuell eine Bereinigungseinfrierung aus

1. Öffnen Sie zwei Sitzungen für die Datenbank, die Tabelle enthält, die Sie bereinigen möchten. Verwenden Sie für die zweite Sitzung „screen“ oder ein anderes Hilfsprogramm, das die Sitzung beibehält, wenn die Verbindung verloren geht.
2. Rufen Sie in der ersten Sitzung die Prozess-ID (PID) der Selbstbereinigungssitzung ab, die für die Tabelle ausgeführt wird.

   Führen Sie die folgende Abfrage aus, um die PID der Selbstbereinigungssitzung abzurufen.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) LIKE '%VACUUM%' ORDER BY
   xact_start;
   ```

3. Berechnen Sie in der zweiten Sitzung die Menge des Arbeitsspeichers, den Sie für diese Operation benötigen. In diesem Beispiel ermitteln Sie, dass für diese Operation bis zu 2 GB Arbeitsspeicher verwendet werden können. Daher wird für die aktuelle Sitzung `maintenance_work_mem` auf 2 GB festgelegt.

   ```
   SET maintenance_work_mem='2 GB';
   SET
   ```

4. Geben Sie in der zweiten Sitzung einen `vacuum freeze verbose`-Befehl für die Tabelle aus. Die Verbose-Einstellung ist nützlich, da Sie Aktivität sehen können, auch wenn in PostgreSQL hierfür zurzeit kein Fortschrittsbericht verfügbar ist.

   ```
   \timing on
   Timing is on.
   ```

```
vacuum freeze verbose pgbench_branches;
```

```
INFO:  vacuuming "public.pgbench_branches"
INFO:  index "pgbench_branches_pkey" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  index "pgbench_branches_test_index" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  "pgbench_branches": found 0 removable, 50 nonremovable row versions
     in 43 out of 43 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 9347 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
Time: 2.765 ms
```

5. Wenn in Sitzung 1 die Selbstbereinigung die Bereinigungssitzung blockiert hat, zeigt `pg_stat_activity` dass das Warten für Ihre Bereinigungssitzung T ist. In diesem Fall müssen Sie den Selbstbereinigungsvorgang wie folgt beenden:

```
SELECT pg_terminate_backend('the_pid');
```

> **Note**
>
> Einige niedrigere Versionen von Amazon Aurora können einen Selbstbereinigungsprozess mit dem vorherigen Befehl nicht beenden und schlagen mit dem folgenden Fehler fehl: `ERROR: 42501: must be a superuser to terminate superuser process LOCATION: pg_terminate_backend, signalfuncs.c:227`. Um die gepatchten PostgreSQL-Versionen zu finden, suchen Sie in den Updates von Amazon Aurora PostgreSQL nach folgendem Aufzählungspunkt:
>
> ```
> Allow rds_superuser to terminate backends which are not explicitly
>  associated with a role
> ```

An diesem Punkt beginnt Ihre Sitzung. Die Selbstbereinigung startet sofort neu, da diese Tabelle in der Liste der Aufgaben vermutlich ganz oben steht.

6. Initiieren Sie Ihren `vacuum freeze verbose`-Befehl in der zweiten Sitzung und beenden Sie dann den Selbstbereinigungsvorgang in der ersten Sitzung.

Neuindizierung einer Tabelle während der Ausführung einer Selbstbereinigung

Wenn ein Index beschädigt wurde, verarbeitet die Selbstbereinigung die Tabelle weiter und schlägt fehl. Wenn Sie in dieser Situation eine manuelle Bereinigung versuchen, werden Sie eine Fehlermeldung ähnlich der folgenden erhalten.

```
postgres=>  vacuum freeze pgbench_branches;
ERROR: index "pgbench_branches_test_index" contains unexpected
   zero page at block 30521
HINT: Please REINDEX it.
```

Wenn der Index beschädigt ist und eine Selbstbereinigung für die Tabelle versucht wird, konkurrieren Sie mit einer Selbstbereinigungssitzung, die bereits ausgeführt wird. Wenn Sie einen REINDEX-Befehl ausgeben, wird eine exklusive Sperre für die Tabelle ausgeführt. Schreiboperationen werden blockiert. Gleiches gilt für Lesevorgänge, die diesen spezifischen Index verwenden.

So führen Sie eine Neuindizierung für eine Tabelle aus, wenn eine Selbstbereinigung für die Tabelle ausgeführt wird

1. Öffnen Sie zwei Sitzungen für die Datenbank, die Tabelle enthält, die Sie bereinigen möchten. Verwenden Sie für die zweite Sitzung „screen“ oder ein anderes Hilfsprogramm, das die Sitzung beibehält, wenn die Verbindung verloren geht.
2. Rufen Sie in der ersten Sitzung die PID der Selbstbereinigungssitzung ab, die für die Tabelle ausgeführt wird.

   Führen Sie die folgende Abfrage aus, um die PID der Selbstbereinigungssitzung abzurufen.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) like '%VACUUM%' ORDER BY
   ```

```
xact_start;
```

3. Führen Sie in der zweiten Sitzung den Neuindizierungsbefehl aus.

```
\timing on
Timing is on.
reindex index pgbench_branches_test_index;
REINDEX
  Time: 9.966 ms
```

4. Wenn die Selbstbereinigung den Prozess blockiert hat, wird Ihnen in der ersten Sitzung in `pg_stat_activity` angezeigt, dass das Warten für Ihre Bereinigungssitzung "T" ist. In diesem Fall beenden Sie den Selbstbereinigungsprozess.

```
SELECT pg_terminate_backend('the_pid');
```

   An diesem Punkt beginnt Ihre Sitzung. Es ist wichtig, zu beachten, dass die Selbstbereinigung sofort neu gestartet wird, da diese Tabelle in der Liste der Aufgaben wahrscheinlich ganz oben steht.

5. Initiieren Sie Ihren Befehl in der zweiten Sitzung und beenden Sie dann den Selbstbereinigungsvorgang in der ersten Sitzung.

Verwalten der automatischen Bereinigung mit großen Indizes

Im Rahmen ihrer Funktion führt die automatische Bereinigung mehrere Bereinigungsphasen aus, während sie für eine Tabelle ausgeführt wird. Bevor die Tabelle bereinigt wird, werden zunächst alle Indizes bereinigt. Wenn mehrere große Indizes entfernt werden, benötigt diese Phase einen großen Zeit- und Ressourcenaufwand. Es hat sich daher bewährt, die Anzahl der Indizes in einer Tabelle zu kontrollieren und ungenutzte Indizes zu entfernen.

Überprüfen Sie für diesen Vorgang zunächst die Gesamtindexgröße. Stellen Sie dann fest, ob es möglicherweise unbenutzte Indizes gibt, die entfernt werden können, wie in den folgenden Beispielen dargestellt.

So überprüfen Sie die Größe der Tabelle und ihrer Indizes

```
postgres=> select pg_size_pretty(pg_relation_size('pgbench_accounts'));
pg_size_pretty
6404 MB
```

```
(1 row)
```

```
postgres=> select pg_size_pretty(pg_indexes_size('pgbench_accounts'));
pg_size_pretty
11 GB
(1 row)
```

In diesem Beispiel ist die Größe der Indizes größer als die Tabelle. Dieser Unterschied kann zu Leistungsproblemen führen, da die Indizes überlastet oder ungenutzt sind, was sich sowohl auf die automatische Bereinigung als auch auf Insert-Operationen auswirkt.

So prüfen Sie, ob ungenutzte Indizes vorhanden sind

Mithilfe der Ansicht pg_stat_user_indexes können Sie überprüfen, wie oft ein Index für die Spalte `idx_scan` verwendet wird. Im folgenden Beispiel haben die ungenutzten Indizes den `idx_scan`-Wert `0`.

```
postgres=> select * from pg_stat_user_indexes where relname = 'pgbench_accounts' order
 by idx_scan desc;

relid  | indexrelid | schemaname | relname          | indexrelname          | idx_scan
 | idx_tup_read | idx_tup_fetch
-------+------------+------------+------------------+-----------------------+----------
+--------------+--------------
16433  | 16454      | public     | pgbench_accounts | index_f               | 6
 | 6            | 0
16433  | 16450      | public     | pgbench_accounts | index_b               | 3
 | 199999       | 0
16433  | 16447      | public     | pgbench_accounts | pgbench_accounts_pkey | 0
 | 0            | 0
16433  | 16452      | public     | pgbench_accounts | index_d               | 0
 | 0            | 0
16433  | 16453      | public     | pgbench_accounts | index_e               | 0
 | 0            | 0
16433  | 16451      | public     | pgbench_accounts | index_c               | 0
 | 0            | 0
16433  | 16449      | public     | pgbench_accounts | index_a               | 0
 | 0            | 0
(7 rows)
```

```
postgres=> select schemaname, relname, indexrelname, idx_scan from pg_stat_user_indexes
 where relname = 'pgbench_accounts' order by idx_scan desc;

schemaname  | relname          | indexrelname          | idx_scan
------------+------------------+-----------------------+----------
public      | pgbench_accounts | index_f               | 6
public      | pgbench_accounts | index_b               | 3
public      | pgbench_accounts | pgbench_accounts_pkey | 0
public      | pgbench_accounts | index_d               | 0
public      | pgbench_accounts | index_e               | 0
public      | pgbench_accounts | index_c               | 0
public      | pgbench_accounts | index_a               | 0
(7 rows)
```

> **Note**
>
> Diese Statistiken sind ab dem Zeitpunkt, an dem die Statistiken zurückgesetzt werden, inkrementell. Angenommen, Sie haben einen Index, der nur am Ende eines Geschäftsquartals oder nur für einen bestimmten Bericht verwendet wird. Es ist möglich, dass dieser Index seit dem Zurücksetzen der Statistiken nicht mehr verwendet wurde. Weitere Informationen finden Sie unter Statistikfunktionen. Indizes, die verwendet werden, um Eindeutigkeit zu erzwingen, werden nicht gescannt und sollten nicht als ungenutzte Indizes identifiziert werden. Um die ungenutzten Indizes zu identifizieren, sollten Sie über fundierte Kenntnisse der Anwendung und ihrer Abfragen verfügen.

Um zu überprüfen, wann die Statistiken für eine Datenbank zuletzt zurückgesetzt wurden, verwenden Sie `pg_stat_database`.

```
postgres=> select datname, stats_reset from pg_stat_database where datname =
 'postgres';

datname   | stats_reset
----------+-------------------------------
postgres  | 2022-11-17 08:58:11.427224+00
(1 row)
```

Möglichst schnelles Bereinigen einer Tabelle

RDS für PostgreSQL 12 und höher

Wenn Sie zu viele Indizes in einer großen Tabelle haben, nähert sich Ihre DB-Instance möglicherweise dem Transaktions-ID-Wraparound (XID), also dem Zeitpunkt, an dem der XID-Zähler auf Null zurückgeht. Wenn diese Option nicht aktiviert ist, kann diese Situation zu Datenverlust führen. Sie können die Tabelle jedoch schnell bereinigen, ohne die Indizes zu bereinigen. In RDS für PostgreSQL 12 und höher können Sie VACUUM mit der Klausel `INDEX_CLEANUP` verwenden.

```
postgres=> VACUUM (INDEX_CLEANUP FALSE, VERBOSE TRUE) pgbench_accounts;

INFO: vacuuming "public.pgbench_accounts"
INFO: table "pgbench_accounts": found 0 removable, 8 nonremovable row versions in 1 out
 of 819673 pages
DETAIL: 0 dead row versions cannot be removed yet, oldest xmin: 7517
Skipped 0 pages due to buffer pins, 0 frozen pages.
CPU: user: 0.01 s, system: 0.00 s, elapsed: 0.01 s.
```

Wenn eine automatische Bereinigungssitzung bereits läuft, müssen Sie sie beenden, um mit dem manuellen VACUUM-Vorgang zu beginnen. Informationen zur Ausführung manueller Bereinigungseinfrierungen finden Sie unter Ausführen einer manuellen Bereinigungseinfrierung.

> **Note**
>
> Wenn Sie die Indexbereinigung regelmäßig überspringen, führt dies zu einer Aufblähung des Index, was sich negativ auf die Scanleistung auswirkt. Der Index behält tote Zeilen und die Tabelle tote Zeilenzeiger bei. Dies hat zur Folge, dass `pg_stat_all_tables.n_dead_tup` anwächst, bis eine Selbstbereinigung oder ein manueller VACUUM-Vorgang mit Indexbereinigung ausgeführt wird. Verwenden Sie dieses Verfahren am besten nur, um einen Transaktions-ID-Wraparound zu verhindern.

RDS für PostgreSQL 11 und niedriger

In RDS für PostgreSQL 11 und niedrigeren Versionen besteht die einzige Möglichkeit, den Bereinigungsvorgang schneller abzuschließen, darin, die Anzahl der Indizes in einer Tabelle zu reduzieren. Das Löschen eines Index kann sich auf Abfragepläne auswirken. Wir empfehlen, zuerst

unbenutzte Indizes zu löschen und dann die Indizes löschen, wenn der XID-Wraparound kurz bevorsteht. Nach Abschluss des Bereinigungsvorgangs können Sie diese Indizes neu erstellen.

Weitere Parameter, die sich auf die Selbstbereinigung auswirken

Die folgende Abfrage zeigt die Werte einiger Parameter an, die sich direkt auf die Selbstbereinigung und ihr Verhalten auswirken. Die Selbstbereinigungsparameter werden in der PostgreSQL-Dokumentation vollständig beschrieben.

```
SELECT name, setting, unit, short_desc
FROM pg_settings
WHERE name IN (
'autovacuum_max_workers',
'autovacuum_analyze_scale_factor',
'autovacuum_naptime',
'autovacuum_analyze_threshold',
'autovacuum_analyze_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_cost_delay',
'autovacuum_vacuum_cost_limit',
'vacuum_cost_limit',
'autovacuum_freeze_max_age',
'maintenance_work_mem',
'vacuum_freeze_min_age');
```

All diese Parameter wirken sich auf die Selbstbereinigung aus. Die wichtigsten unter ihnen sind jedoch:

- maintenance_work_mem
- autovacuum_freeze_max_age
- autovacuum_max_workers
- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit

Festlegen von Selbstbereinigungsparametern auf Tabellenebene

Sie können Speicherparameter, die sich auf die der Selbstbereinigung beziehen, auf Tabellenebene festlegen. Dies kann im Vergleich zur Änderung des Verhaltens der gesamten Datenbank ein bevorzugtes Verfahren sein. Im Fall großer Tabellen müssen möglicherweise aggressive Einstellungen festgelegt werden, und Sie möchten vielleicht nicht, dass sich die Selbstbereinigung für alle Tabellen auf diese Weise verhält.

Die folgende Abfrage zeigt, für welche Tabellen zurzeit Optionen auf Tabellenebene festgelegt wurden.

```
SELECT relname, reloptions
FROM pg_class
WHERE reloptions IS NOT null;
```

Ein Beispiel, in dem dies nützlich sein kann, sind Tabellen, die sehr viel größer als Ihre restlichen Tabellen sind. Angenommen, es gibt eine Tabelle mit 300 GB und 30 weitere Tabellen mit weniger als 1 GB. In diesem Fall würde es sich anbieten, einige spezifische Parameter nur für die große Tabelle festzulegen, um nicht das Verhalten des gesamten Systems zu ändern.

```
ALTER TABLE mytable set (autovacuum_vacuum_cost_delay=0);
```

Hierdurch wird die kostenbasierte Selbstbereinigungsverzögerung für diese Tabelle auf Kosten einer größeren Ressourcennutzung in Ihrem System deaktiviert. Normalerweise pausiert die Selbstbereinigung für `autovacuum_vacuum_cost_delay` jedes Mal, wenn `autovacuum_cost_limit` erreicht wird. Weitere Details finden Sie in der PostgreSQL-Dokumentation zum Thema kostenbasierte Bereinigung.

Protokollieren von Selbstbereinigung- und Bereinigungsaktivitäten

Informationen über Bereinigungsaktivitäten werden basierend auf der im`rds.force_autovacuum_logging_level`-Parameter angegebenen Ebene an das `postgresql.log` gesendet. Im Folgenden sind die für diesen Parameter zulässigen Werte und die PostgreSQL-Versionen aufgeführt, für die dieser Wert die Standardeinstellung ist:

- `disabled` (PostgreSQL 10, PostgreSQL 9.6)
- `debug5`, `debug4`, `debug3`, `debug2`, `debug1`

- `info` (PostgreSQL 12, PostgreSQL 11)
- `notice`
- `warning` (PostgreSQL 13 und höher)
- `error`, Protokoll, `fatal`, `panic`

Das `rds.force_autovacuum_logging_level` arbeitet mit dem `log_autovacuum_min_duration`-Parameter. Der Wert des `log_autovacuum_min_duration`-Parameters ist der Schwellenwert (in Millisekunden), über dem Selbstbereinigung-Aktionen protokolliert werden. Eine Einstellung von `-1` protokolliert nichts, während eine Einstellung von 0 alle Aktionen protokolliert. Wie bei `rds.force_autovacuum_logging_level`, Standardwerte für `log_autovacuum_min_duration` sind versionsabhängig wie folgt:

- `10000 ms` – PostgreSQL 14, PostgreSQL 13, PostgreSQL 12 und PostgreSQL 11
- `(empty)` – Kein Standardwert für PostgreSQL 10 und PostgreSQL 9.6

Wir empfehlen Ihnen, `rds.force_autovacuum_logging_level` auf `WARNING` einzustellen. Wir empfehlen auch, dass Sie `log_autovacuum_min_duration` auf einen Wert von 1000 bis 5000 einstellen. Eine Einstellung von 5000 Protokollaktivitäten, die länger als 5000 Millisekunden dauern. Jede andere Einstellung als –1 protokolliert auch Meldungen, wenn die Selbstbereinigungsaktion aufgrund einer widersprüchlichen Sperre oder gleichzeitig verworfener Beziehungen übersprungen wird. Weitere Informationen finden Sie unter Selbstbereinigung in der PostgreSQL-Dokumentation.

Um Probleme zu beheben, können Sie die `rds.force_autovacuum_logging_level`-Parameter in eine der Debugebenen ändern, von `debug1` bis zu `debug5` für die ausführlichsten Informationen. Wir empfehlen, die Debug-Einstellungen für kurze Zeiträume und nur zur Fehlerbehebung zu verwenden. Weitere Informationen finden Sie unter Zeitpunkt des Protokollierens in der PostgreSQL-Dokumentation.

> **Note**
>
> PostgreSQL ermöglicht es dem Konto `rds_superuser`, Autovakuum-Sitzungen in `pg_stat_activity` anzuzeigen. Sie können beispielsweise eine Selbstbereinigungssitzung identifizieren und beenden, die die Ausführung eines Befehls blockiert oder langsamer als ein manuell ausgegebener Bereinigungsbefehl ausgeführt wird.

Verstehen des Verhaltens der Selbstbereinigung bei ungültigen Datenbanken

Ein neuer `-2`-Wert wird in die `datconnlimit`-Spalte im `pg_database`-Katalog eingeführt, um Datenbanken, die während des DROP DATABASE-Vorgangs unterbrochen wurden, als ungültig zu kennzeichnen.

Dieser neue Wert ist in den folgenden Versionen von Aurora PostgreSQL verfügbar:

- 15.4 und alle höheren Versionen
- 14.9 und höhere Versionen
- 13.12 und höhere Versionen
- 12.16 und höhere Versionen
- 11.21 und höhere Versionen

Ungültige Datenbanken haben keinen Einfluss auf die Möglichkeit der Selbstbereinigung, die Funktionalität für gültige Datenbanken einzufrieren. Die Selbstbereinigung ignoriert ungültige Datenbanken. Folglich funktionieren reguläre Selbstbereinigungsfunktionen weiterhin ordnungsgemäß und werden effizient für alle gültigen Datenbanken in Ihrer PostgreSQL-Umgebung ausgeführt.

Themen

- Überwachen der Transaktions-ID
- Anpassen der Überwachungsanfrage
- Beheben des Problems mit ungültigen Datenbanken

Überwachen der Transaktions-ID

Die `age(datfrozenxid)`-Funktion wird häufig verwendet, um das Alter der Transaktions-ID (XID) von Datenbanken zwecks Verhinderung eines Transaktions-ID-Wrapraround zu überwachen.

Da ungültige Datenbanken von der Selbstbereinigung ausgenommen sind, kann ihr Transaktions-ID-(XID)-Zähler den Maximalwert von `2 billion` erreichen, auf `- 2 billion` zurückgehen und diesen Zyklus auf unbestimmte Zeit fortsetzen. Eine typische Abfrage zur Überwachung des Transaktions-ID-Wraparound könnte wie folgt aussehen:

```
SELECT max(age(datfrozenxid)) FROM pg_database;
```

Mit der Einführung des Werts -2 für `datconnlimit` können ungültige Datenbanken jedoch die Ergebnisse dieser Abfrage verfälschen. Da diese Datenbanken nicht gültig sind und nicht Teil regelmäßiger Wartungsprüfungen sein sollten, können sie zu einem falsch-positiven Ergebnis führen, sodass Sie glauben, dass der `age(datfrozenxid)`-Wert höher ist, als er tatsächlich ist.

Anpassen der Überwachungsanfrage

Um eine genaue Überwachung zu gewährleisten, sollten Sie Ihre Überwachungsabfrage so anpassen, dass ungültige Datenbanken ausgeschlossen werden. Folgen Sie dieser empfohlenen Abfrage:

```
SELECT
    max(age(datfrozenxid))
FROM
    pg_database
WHERE
    datconnlimit <> -2;
```

Diese Abfrage stellt sicher, dass nur gültige Datenbanken bei der `age(datfrozenxid)`-Berechnung berücksichtigt werden, sodass das Alter der Transaktions-ID in Ihrer gesamten PostgreSQL-Umgebung genau wiedergegeben wird.

Beheben des Problems mit ungültigen Datenbanken

Wenn Sie versuchen, eine Verbindung zu einer ungültigen Datenbank herzustellen, wird möglicherweise eine Fehlermeldung ähnlich der folgenden angezeigt:

```
postgres=> \c db1
connection to server at "mydb.xxxxxxxxxx.us-west-2.rds.amazonaws.com" (xx.xx.xx.xxx),
 port xxxx failed: FATAL:  cannot connect to invalid database "db1"
HINT:  Use DROP DATABASE to drop invalid databases.
Previous connection kept
```

Wenn darüber hinaus der `log_min_messages`-Parameter auf `DEBUG2` oder höher gesetzt ist, fallen Ihnen möglicherweise die folgenden Protokolleinträge auf, die darauf hinweisen, dass der Selbstbereinigungsprozess die ungültige Datenbank überspringt:

```
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db6"
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db1"
```

Um dieses Problem zu beheben, befolgen Sie den `HINT`, der während des Verbindungsversuchs ausgegeben wurde. Stellen Sie mit Ihrem RDS-Master-Konto oder einem Datenbankkonto mit der `rds_superuser`-Rolle eine Verbindung zu einer gültigen Datenbank her und verwerfen Sie ungültige Datenbanken.

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

Identifizieren und Beheben von Problemen mit Blockern aggressiver Bereinigungen in Aurora PostgreSQL

In PostgreSQL ist die Bereinigung von entscheidender Bedeutung zur Sicherstellung der Integrität der Datenbank, da durch diese Speicherplatz zurückgewonnen wird und Probleme mit Transaktions-ID-Wraparounds vermieden werden. Manchmal wird eine reibungslose Bereinigung jedoch verhindert, was zu Leistungseinbußen, einem aufgeblähten Speicher und sogar zu einer Beeinträchtigung der Verfügbarkeit Ihrer DB-Instance durch einen Transaktions-ID-Wraparound führen kann. Daher ist die Identifizierung und Lösung dieser Probleme für eine optimale Datenbankleistung und -verfügbarkeit unerlässlich. Lesen Sie Verstehen der Selbstbereinigung in Umgebungen von Amazon RDS für PostgreSQL, um mehr über die Selbstbereinigung zu erfahren.

Die `postgres_get_av_diag()`-Funktion hilft bei der Identifizierung von Problemen, die den aggressiven Bereinigungsprozess entweder verhindern oder verzögern. Es werden Vorschläge

bereitgestellt, die Befehle zur Behebung des Problems beinhalten können, wenn es identifiziert werden kann, oder Anleitungen für weitere Diagnosen, wenn das Problem nicht identifiziert werden kann. Blocker aggressiver Bereinigungen werden gemeldet, wenn das Alter den Schwellenwert für die adaptive Selbstbereinigung von RDS von 500 Millionen Transaktions-IDs überschreitet.

Wie alt ist die Transaktions-ID?

Die `age()`-Funktion für Transaktions-IDs berechnet die Anzahl der Transaktionen, die seit der ältesten nicht eingefrorenen Transaktions-ID für eine Datenbank (`pg_database.datfrozenxid`) oder Tabelle (`pg_class.relfrozenxid`) stattgefunden haben. Dieser Wert gibt die Datenbankaktivität seit dem letzten aggressiven Bereinigungsvorgang an und zeigt den wahrscheinlichen Workload für bevorstehende VACUUM-Prozesse auf.

Was ist eine aggressive Bereinigung?

Eine aggressive VACUUM-Operation führt einen umfassenden Scan aller Seiten in einer Tabelle durch, einschließlich der Seiten, die normalerweise bei regulären VACUUM-Operationen übersprungen werden. Dieser gründliche Scan zielt darauf ab, Transaktions-IDs, die sich ihrem Höchstalter nähern, „einzufrieren“ und so eine Situation, die als Transaktions-ID-Wraparound bezeichnet wird, effektiv zu verhindern.

Damit `postgres_get_av_diag()` Blocker meldet, muss der Blocker mindestens 500 Millionen Transaktionen alt sein.

Themen

- Installieren von Überwachungs- und Diagnosetools für die Selbstbereinigung in Aurora PostgreSQL
- Funktionen von postgres_get_av_diag() in Aurora PostgreSQL
- Beseitigen von identifizierbaren Bereinigungsblockern in Aurora PostgreSQL
- Beheben von nicht identifizierbaren Bereinigungsblockern in Aurora PostgreSQL
- Beheben von Leistungsproblemen bei der Bereinigung in Aurora PostgreSQL
- Erläuterung der HINWEIS-Meldungen in Aurora PostgreSQL

Installieren von Überwachungs- und Diagnosetools für die Selbstbereinigung in Aurora PostgreSQL

Die `postgres_get_av_diag()`-Funktion ist derzeit in den folgenden Versionen von Aurora PostgreSQL verfügbar:

- 17.4 und höhere 17-Versionen
- 16.7 und höhere 16-Versionen
- 15.11 und höhere 15-Versionen
- 14.16 und höhere 14-Versionen
- 13.19 und höhere 13-Versionen

Um `postgres_get_av_diag()` zu verwenden, erstellen Sie die `rds_tools`-Erweiterung.

```
postgres=> CREATE EXTENSION rds_tools ;
CREATE EXTENSION
```

Stellen Sie sicher, dass die Erweiterung installiert ist.

```
postgres=> \dx rds_tools
             List of installed extensions
   Name    | Version |  Schema   |                    Description
 ----------+---------+-----------
+------------------------------------------------------------
 rds_tools |   1.9   | rds_tools | miscellaneous administrative functions for RDS
 PostgreSQL
 1 row
```

Stellen Sie sicher, dass die Funktion erstellt ist.

```
postgres=> SELECT
    proname function_name,
    pronamespace::regnamespace function_schema,
    proowner::regrole function_owner
FROM
    pg_proc
WHERE
    proname = 'postgres_get_av_diag';
    function_name     | function_schema | function_owner
----------------------+-----------------+----------------
 postgres_get_av_diag | rds_tools       | rds_superuser
(1 row)
```

Funktionen von postgres_get_av_diag() in Aurora PostgreSQL

The `postgres_get_av_diag()`-Funktion ruft Diagnoseinformationen über Selbstbereinigungsprozesse ab, die in einer Datenbank von Aurora PostgreSQL blockiert sind oder zeitlich verzögert erfolgen. Die Abfrage muss in der Datenbank mit der ältesten Transaktions-ID ausgeführt werden, damit genaue Ergebnisse geliefert werden. Weitere Informationen zur Verwendung der Datenbank mit der ältesten Transaktions-ID finden Sie unter Keine Verbindung zur Datenbank mit dem Alter der ältesten Transaktions-ID

```
SELECT
    blocker,
    DATABASE,
    blocker_identifier,
    wait_event,
    TO_CHAR(autovacuum_lagging_by, 'FM9,999,999,999') AS autovacuum_lagging_by,
    suggestion,
    suggested_action
FROM (
    SELECT
        *
    FROM
        rds_tools.postgres_get_av_diag ()
    ORDER BY
        autovacuum_lagging_by DESC) q;
```

Die `postgres_get_av_diag()`-Funktion gibt eine Tabelle mit folgenden Informationen zurück:

blocker

Gibt die Kategorie der Datenbankaktivität an, die die Selbstbereinigung blockiert

- Aktive Anweisung
- Leerlauf in Transaktion
- Vorbereitete Transaktion
- Logischer Replikations-Slot
- Reader-Instances
- Temporäre Tabellen

Datenbank

Gibt den Namen der Datenbank an, sofern zutreffend und unterstützt. Dies ist die Datenbank, in der die Aktivität ausgeführt wird und die Selbstbereinigung blockiert oder blockieren wird. Dies ist die Datenbank, zu der Sie eine Verbindung herstellen und für die Sie Maßnahmen ergreifen müssen.

blocker_identifier

Gibt die ID der Aktivität an, die den Selbstbereinigungsprozess blockiert oder blockieren wird. Bei der ID kann es sich um eine Prozess-ID in Kombination mit einer SQL-Anweisung, einer vorbereiteten Transaktion, einer IP-Adresse einem Lesereplikat und dem Namen des logischen oder physischen Replikationsslots handeln.

wait_event

Gibt das Warteereignis der blockierenden Sitzung an und gilt für die folgenden Blocker:

- Aktive Anweisung
- Leerlauf in Transaktion

autovacum_lagging_by

Gibt pro Kategorie die Anzahl der Transaktionen an, bei denen die Selbstbereinigungsfunktion hinsichtlich der Backlog-Arbeiten im Rückstand ist

suggestion

Gibt Vorschläge zur Behebung des Blocker-Problems an. Diese Anleitungen enthalten gegebenenfalls den Namen der Datenbank, in der die Aktivität vorhanden ist, gegebenenfalls die Prozess-ID (PID) der Sitzung und die auszuführende Aktion.

suggested_action

Schlägt die Aktion vor, die ergriffen werden muss, um das Blocker-Problem zu beheben

Beseitigen von identifizierbaren Bereinigungsblockern in Aurora PostgreSQL

Der Selbstbereinigungsprozess führt aggressive Bereinigungen durch und senkt das Alter der Transaktions-IDs auf einen Wert unterhalb des Schwellenwerts, der durch den `autovacuum_freeze_max_age`-Parameter Ihrer RDS-Instance angegeben ist. Sie können dieses Alter mit der `MaximumUsedTransactionIDs`-Metrik von Amazon CloudWatch nachverfolgen.

Zum Ermitteln der Einstellung von `autovacuum_freeze_max_age` (der Standardwert ist 200 Millionen Transaktions-IDs) für Ihre Amazon-RDS-Instance eignet sich folgende Abfrage:

```
SELECT
    TO_CHAR(setting::bigint, 'FM9,999,999,999') autovacuum_freeze_max_age
FROM
    pg_settings
WHERE
    name = 'autovacuum_freeze_max_age';
```

Beachten Sie, dass `postgres_get_av_diag()` nur dann nach Blockern aggressiver Bereinigungen sucht, wenn das Alter den Schwellenwert für die adaptive Selbstbereinigung von Amazon RDS von 500 Millionen Transaktions-IDs überschreitet. Damit `postgres_get_av_diag()` Blocker erkennt, muss der Blocker mindestens 500 Millionen Transaktionen alt sein.

Die `postgres_get_av_diag()`-Funktion identifiziert die folgenden Arten von Blockern:

Themen

- Aktive Anweisung
- Leerlauf in Transaktion
- Vorbereitete Transaktion
- Logischer Replikations-Slot
- Reader-Instances
- Temporäre Tabellen

Aktive Anweisung

In PostgreSQL ist eine aktive Anweisung eine SQL-Anweisung, die aktuell von der Datenbank ausgeführt wird. Dazu gehören Abfragen, Transaktionen sowie laufende Operationen. Bei der Überwachung mit `pg_stat_activity` wird in der Statusspalte angegeben, dass der Prozess mit der entsprechenden PID aktiv ist.

Die `postgres_get_av_diag()`-Funktion liefert eine Ausgabe ähnlich der folgenden, wenn sie eine Anweisung identifiziert, die inaktiv ist.

```
blocker              | Active statement
database             | my_database
```

```
blocker_identifier    | SELECT pg_sleep(20000);
wait_event            | Timeout:PgSleep
autovacuum_lagging_by | 568,600,871
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (29621);"}
```

Vorgeschlagene Aktion

Gemäß der Empfehlung in der `suggestion`-Spalte kann der Benutzer eine Verbindung zu der Datenbank herstellen, in der sich die aktive Anweisung befindet. Wie in der `suggested_action`-Spalte angegeben, ist es ratsam, die Option zum Beenden der Sitzung sorgfältig zu prüfen. Wenn die Beendigung sicher ist, können Sie die Sitzung mit der `pg_terminate_backend()`-Funktion beenden. Diese Aktion kann von einem Administrator (wie dem RDS-Master-Konto) oder einem Benutzer mit der erforderlichen `pg_terminate_backend()`-Berechtigung durchgeführt werden.

> **Warning**
>
> Eine beendete Sitzung macht die von ihr vorgenommenen Änderungen rückgängig (`ROLLBACK`). Abhängig von Ihren Anforderungen kann es jedoch sein, dass Sie die Anweisung erneut ausführen möchten. Sie sollten dies jedoch erst tun, wenn der Selbstbereinigungsprozess die aggressive Bereinigung abgeschlossen hat.

Leerlauf in Transaktion

Eine „idle in transaction“-Anweisung bezieht sich auf jede Sitzung mit einer offenen, expliziten Transaktion (z. B. durch Ausgabe einer `BEGIN`-Anweisung), in der einige Arbeiten ausgeführt wurden und die nun darauf wartet, dass der Client entweder weitere Aufgaben übergibt oder das Ende der Transaktion durch Ausgabe von `COMMIT`, `ROLLBACK` oder `END` signalisiert (was zu einem impliziten `COMMIT` führen würde).

Die `postgres_get_av_diag()`-Funktion liefert eine Ausgabe ähnlich der folgenden, wenn sie eine `idle in transaction`-Anweisung als Blocker identifiziert.

```
blocker               | idle in transaction
database              | my_database
blocker_identifier    | INSERT INTO tt SELECT * FROM tt;
```

```
wait_event            | Client:ClientRead
autovacuum_lagging_by | 1,237,201,759
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (28438);"}
```

Vorgeschlagene Aktion

Wie in der `suggestion`-Spalte angegeben, können Sie eine Verbindung zu der Datenbank mit der „idle in transaction“-Sitzung herstellen und die Sitzung mit der `pg_terminate_backend()`-Funktion beenden. Der Benutzer kann Ihr Admin-Benutzer (RDS-Master-Konto) oder ein Benutzer mit der `pg_terminate_backend()`-Berechtigung sein.

> **Warning**
>
> Eine beendete Sitzung macht die von ihr vorgenommenen Änderungen rückgängig (`ROLLBACK`). Abhängig von Ihren Anforderungen kann es jedoch sein, dass Sie die Anweisung erneut ausführen möchten. Sie sollten dies jedoch erst tun, wenn der Selbstbereinigungsprozess die aggressive Bereinigung abgeschlossen hat.

Vorbereitete Transaktion

PostgreSQL ermöglicht Transaktionen, die Teil einer zweiphasigen Commit-Strategie sind, auch bekannt als vorbereitete Transaktionen. Diese werden aktiviert, indem der `max_prepared_transactions`-Parameter auf einen Wert ungleich Null gesetzt wird. Vorbereitete Transaktionen sollen sicherstellen, dass eine Transaktion dauerhaft ist und auch nach Datenbankabstürzen, Neustarts oder Verbindungsabbrüchen des Clients verfügbar bleibt. Genau wie reguläre Transaktionen erhalten sie eine Transaktions-ID und können die Selbstbereinigung beeinflussen. Wenn sie im Zustand „vorbereitet“ verbleiben, kann die Selbstbereinigungsfunktion keine Einfrierungen durchführen und es kann zu einem Transaktions-ID-Wraparound kommen.

Wenn Transaktionen auf unbestimmte Zeit in diesem Zustand verbleiben, ohne dass dieser von einem Transaktionsmanager aufgehoben wird, werden sie zu verwaisten vorbereiteten Transaktionen. Die einzige Möglichkeit, dieses Problem zu beheben, besteht darin, die Transaktion mit den Befehlen `COMMIT PREPARED` oder `ROLLBACK PREPARED` festzuschreiben bzw. rückgängig zu machen.

> **Note**
>
> Beachten Sie, dass ein Backup, das während einer vorbereiteten Transaktion erstellt wurde, diese Transaktion auch nach der Wiederherstellung noch enthält. Sehen Sie sich die folgenden Informationen genauer an, um mehr über das Finden und Beenden solcher Transaktionen zu erfahren.

Die `postgres_get_av_diag()`-Funktion liefert die folgende Ausgabe, wenn sie eine vorbereitete Transaktion als Blocker identifiziert.

```
blocker               | Prepared transaction
database              | my_database
blocker_identifier    | myptx
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database" and consider either COMMIT
 or ROLLBACK the prepared transaction using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"COMMIT PREPARED 'myptx';",[OR],"ROLLBACK PREPARED 'myptx';"}
```

Vorgeschlagene Aktion

Stellen Sie, wie in der Vorschlagsspalte angegeben, eine Verbindung zu der Datenbank her, auf der sich die vorbereitete Transaktion befindet. Prüfen Sie basierend auf der `suggested_action`-Spalte sorgfältig, ob Sie entweder `COMMIT` oder ausführen `ROLLBACK` möchten und welche Aktion angemessen ist.

Zur allgemeinen Überwachung vorbereiteter Transaktionen bietet PostgreSQL eine Katalogansicht mit der Bezeichnung `pg_prepared_xacts`. Sie können die folgende Abfrage verwenden, um vorbereitete Transaktionen zu finden.

```
SELECT
    gid,
    prepared,
    owner,
    database,
    transaction AS oldest_xmin
FROM
    pg_prepared_xacts
```

```
ORDER BY
    age(transaction) DESC;
```

Logischer Replikations-Slot

Zweck eines Replikations-Slots ist es, nicht genutzte Änderungen so lange aufzubewahren, bis sie auf einen Zielserver repliziert wurden. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation unter Logische Replikation.

Es gibt zwei Arten von logischen Replikations-Slots.

Inaktive logische Replikations-Slots

Wenn die Replikation beendet wird, können nicht genutzte Transaktionsprotokolle nicht entfernt werden und der Replikations-Slot wird inaktiv. Obwohl ein inaktiver logischer Replikations-Slot derzeit nicht von einem Subscriber verwendet wird, verbleibt er auf dem Server, was zur Aufbewahrung von WAL-Dateien führt und verhindert, dass alte Transaktionsprotokolle entfernt werden. Dies kann die Festplattennutzung erhöhen und insbesondere verhindern, dass der Selbstbereinigungsprozess interne Katalogtabellen bereinigt, da das System verhindern muss, dass LSN-Informationen überschrieben werden. Wird dieses Problem nicht behoben, kann das zu einem Aufblähen des Katalogs, zu Leistungseinbußen und zu einem erhöhten Risiko für eine Wraparound-Bereinigung führen, was wiederum Transaktionsausfallzeiten nach sich ziehen kann.

Aktive, aber langsame logische Replikations-Slots

Manchmal verzögert sich das Entfernen toter Katalog-Tupel aufgrund von Leistungseinbußen bei der logischen Replikation. Diese Verzögerung bei der Replikation verlangsamt die Aktualisierung von `catalog_xmin`und kann zu einem aufgeblähten Katalog und einer Wraparound-Bereinigung führen.

Die `postgres_get_av_diag()`-Funktion liefert eine Ausgabe ähnlich der folgenden, wenn sie einen logischen Replikations-Slot als Blocker identifiziert.

```
blocker               | Logical replication slot
database              | my_database
blocker_identifier    | slot1
wait_event            | Not applicable
autovacuum_lagging_by | 1,940,103,068
suggestion            | Ensure replication is active and resolve any lag for the slot
 if active. If inactive, consider dropping it using the command in suggested_action.
 For more information, see Working with PostgreSQL autovacuum in the Amazon RDS User
 Guide.
```

```
suggested_action      | {"SELECT pg_drop_replication_slot('slot1') FROM
 pg_replication_slots WHERE active = 'f';"}
```

Vorgeschlagene Aktion

Überprüfen Sie zur Behebung des Problems die Replikationskonfiguration auf Probleme mit dem Zielschema oder Daten, die den Anwendungsprozess möglicherweise beenden. Die häufigsten Gründe sind die folgenden:

- fehlende Spalten
- inkompatibler Datentyp
- nicht übereinstimmende Daten
- fehlende Tabelle

Wenn das Problem mit Infrastrukturproblemen zusammenhängt:

- Netzwerkprobleme – Wie behebe ich Probleme mit einer Amazon-RDS-Datenbank, die sich in einem inkompatiblen Netzwerkstatus befindet?
- Die Datenbank oder DB-Instance ist aus folgenden Gründen nicht verfügbar:
  - Die Replikat-Instance hat keinen Speicherplatz mehr – Sehen Sie unter Wie behebe ich Probleme, die auftreten, wenn den Amazon-RDS-DB-Instances der Speicherplatz ausgeht? nach, um mehr über das Hinzufügen von Speicherplatz zu erfahren.
  - Incompatible-parameters – Sehen Sie unter Wie behebe ich Fehler bei einer Amazon-RDS-DB-Instance, die im Status „incompatible-parameters“ feststeckt? nach, um mehr über die Behebung des Problems zu erfahren.

Wenn sich Ihre Instance außerhalb des AWS-Netzwerks oder auf AWS-EC2 befindet, wenden Sie sich an Ihren Administrator, um zu erfahren, wie Sie die Verfügbarkeits- oder Infrastrukturprobleme lösen können.

Löschen des inaktiven Slots

> **Warning**
>
> Achtung: Vergewissern Sie sich vor dem Löschen eines Replikations-Slots, dass dieser keine laufende Replikation aufweist, inaktiv ist und sich im Zustand „nicht wiederherstellbar“

befindet. Das vorzeitige Löschen eines Slots kann zu Replikationsunterbrechungen oder Datenverlust führen.

Nachdem Sie sich vergewissert haben, dass der Replikations-Slot nicht mehr benötigt wird, löschen Sie diesen, damit die Selbstbereinigung fortgesetzt werden kann. Die `active = 'f'`-Bedingung sorgt dafür, dass nur ein inaktiver Slot gelöscht wird.

```
SELECT pg_drop_replication_slot('slot1') WHERE active ='f'
```

Reader-Instances

Wenn die Einstellung „hot_standby_feedback“ aktiviert ist, verhindert sie, dass der Selbstbereinigungsprozess auf der Writer-Instance tote Zeilen entfernt, die möglicherweise noch von Abfragen benötigt werden, die auf der Reader-Instance ausgeführt werden. Dieses Verhalten ist notwendig, da Abfragen, die auf der Reader-Instance ausgeführt werden (gilt auch für Reader-Instances in Aurora Global Database), erfordern, dass diese Zeilen auf der Writer-Instance verfügbar bleiben, um Abfragekonflikte und -abbrüche zu vermeiden.

> **Note**
>
> `hot_standby_feedback` ist standardmäßig aktiviert und kann in Aurora PostgreSQL nicht geändert werden.

Die `postgres_get_av_diag()`-Funktion liefert eine Ausgabe ähnlich der folgenden, wenn sie ein Lesereplikat mit einem physischen Replikations-Slot als Blocker identifiziert.

```
blocker               | Oldest query running on aurora reader
database              | Not applicable
blocker_identifier    | my-aurora-reader-2
wait_event            | Not applicable
autovacuum_lagging_by | 540,122,859
suggestion            | Run the following query on the reader "my-aurora-reader-2" to
 find the long running query:

                      | SELECT * FROM pg_catalog.pg_stat_activity WHERE
 backend_xmin::text::bigint = 523476310;
```

```
                      | Review carefully and you may consider terminating the query on
 reader using suggested_action.
suggested_action      | {"SELECT pg_terminate_backend(pid) FROM
 pg_catalog.pg_stat_activity WHERE backend_xmin::text::bigint = 523476310;","

                      | [OR]


                      | ","Delete the reader if not needed"}
```

Wie in der `suggested_action`-Spalte empfohlen, sollten Sie diese Optionen sorgfältig prüfen, um die Blockierung des Selbstbereinigungsprozesses aufzuheben.

- Abfrage beenden – Gemäß der Empfehlung in der Vorschlagspalte können Sie eine Verbindung zum Lesereplikat herstellen. Laut der suggested_action-Spalte empfiehlt es sich jedoch, die Option zum Beenden der Sitzung sorgfältig zu prüfen. Wenn die Beendigung als sicher erachtet wird, können Sie die Sitzung mit der `pg_terminate_backend()`-Funktion beenden. Diese Aktion kann von einem Administrator (wie dem RDS-Master-Konto) oder einem Benutzer mit der erforderlichen pg_terminate_backend()-Berechtigung durchgeführt werden.

  Sie können den folgenden SQL-Befehl auf dem Lesereplikat ausführen, um die Abfrage zu beenden, die verhindert, dass der Bereinigungsprozess auf der primären Datenbank alte Zeilen löscht. Der Wert von `backend_xmin` wird in der Ausgabe der Funktion gemeldet:

  ```
  SELECT
      pg_terminate_backend(pid)
  FROM
      pg_catalog.pg_stat_activity
  WHERE
      backend_xmin::text::bigint = backend_xmin;
  ```

- Nicht benötigte Reader-Instances löschen – Wenn die Reader-Instance nicht mehr benötigt wird, können Sie sie löschen. Dadurch entfällt der damit verbundene Replikations-Overhead und die primäre Datenbank kann die Transaktionsprotokolle recyceln, ohne von der Instance zurückgehalten zu werden.

Temporäre Tabellen

Temporäre Tabellen, die mit dem `TEMPORARY`-Schlüsselwort erstellt wurden, befinden sich im temporären Schema, z. B. in pg_temp_xxx, und sind nur für die Sitzung zugänglich, in der sie erstellt

wurden. Temporäre Tabellen werden gelöscht, wenn die Sitzung endet. Diese Tabellen sind jedoch für den Selbstbereinigungsprozess von PostgreSQL unsichtbar und müssen manuell von der Sitzung bereinigt werden, von der sie erstellt wurden. Der Versuch, die temporäre Tabelle von einer anderen Sitzung aus bereinigen zu lassen, hat keine Wirkung.

In seltenen Fällen kann es eine temporäre Tabelle geben, die keiner aktiven Sitzung angehört. Wenn die innehabende Sitzung aufgrund eines fatalen Absturzes, eines Netzwerkproblems oder eines ähnlichen Ereignisses unerwartet beendet wird, wird die temporäre Tabelle möglicherweise nicht bereinigt, sodass sie als „verwaiste“ Tabelle zurückbleibt. Wenn der Selbstbereinigungsprozess von PostgreSQL eine solche verwaiste temporäre Tabelle entdeckt, protokolliert er die folgende Meldung:

```
LOG: autovacuum: found orphan temp table \"%s\".\"%s\" in database \"%s\"
```

Die `postgres_get_av_diag()`-Funktion liefert eine Ausgabe ähnlich der folgenden, wenn sie eine temporäre Tabelle als Blocker identifiziert. Damit die Funktion die Ausgabe in Bezug auf temporäre Tabellen korrekt anzeigt, muss sie auf der Datenbank ausgeführt werden, auf der diese Tabellen vorhanden sind.

```
blocker               | Temporary table
database              | my_database
blocker_identifier    | pg_temp_14.ttemp
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database". Review carefully, you
 may consider dropping temporary table using command in suggested_action. For more
 information, see Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"DROP TABLE ttemp;"}
```

Vorgeschlagene Aktion

Folgen Sie den Anleitungen aus der `suggestion`-Spalte der Ausgabe, um die temporäre Tabelle zu identifizieren und zu entfernen, die die Ausführung des Selbstbereinigungsprozesses verhindert. Verwenden Sie den folgenden Befehl, um die temporäre Tabelle zu löschen, die von `postgres_get_av_diag()` gemeldet wurde. Ersetzen Sie den Tabellennamen auf der Grundlage der von der `postgres_get_av_diag()`-Funktion gelieferten Ausgabe.

```
DROP TABLE my_temp_schema.my_temp_table;
```

Mit der folgenden Abfrage können temporäre Tabellen identifiziert werden:

```
SELECT
    oid,
    relname,
    relnamespace::regnamespace,
    age(relfrozenxid)
FROM
    pg_class
WHERE
relpersistence = 't'
ORDER BY
    age(relfrozenxid) DESC;
```

Beheben von nicht identifizierbaren Bereinigungsblockern in Aurora PostgreSQL

In diesem Abschnitt werden weitere Gründe für ein Verhindern des Fortschritts bei der Bereinigung untersucht. Diese Probleme können derzeit nicht direkt mit der `postgres_get_av_diag()`-Funktion identifiziert werden.

Themen

- Index-Inkonsistenz
- Außergewöhnlich hohe Transaktionsrate

Index-Inkonsistenz

Ein logisch inkonsistenter Index kann verhindern, dass der Selbstbereinigungsprozess voranschreitet. Die folgenden oder ähnliche Fehler werden entweder während der Bereinigungsphase des Index oder beim Zugriff auf den Index durch SQL-Anweisungen protokolliert.

```
ERROR: right sibling's left-link doesn't match:block 5 links to 10 instead of expected
 2 in index ix_name
```

```
ERROR: failed to re-find parent key in index "XXXXXXXXXX" for deletion target page XXX
CONTEXT:  while vacuuming index index_name of relation schema.table
```

Empfehlung

Erstellen Sie den Index neu oder überspringen Sie Indizes mittels `INDEX_CLEANUP` bei einem manuellen `VACUUM FREEZE`.

- Verwenden der CONCURRENTLY-Option – Vor PostgreSQL-Version 12 erforderte die Neuerstellung eines Index eine exklusive Tabellensperre, die den Zugriff auf die Tabelle einschränkte. In PostgreSQL-Version 12 und höheren Versionen ermöglicht die CONCURRENTLY-Option das Sperren auf Zeilenebene, wodurch die Verfügbarkeit der Tabelle erheblich verbessert wird. Dies ist der Befehl:

  ```
  REINDEX INDEX ix_name CONCURRENTLY;
  ```

  CONCURRENTLY ist zwar weniger störend, kann aber bei stark frequentierten Tabellen langsamer sein. Erwägen Sie, den Index nach Möglichkeit in Zeiten mit geringem Datenverkehr zu erstellen. Weitere Informationen finden Sie unter REINDEX in der PostgreSQL-Dokumentation.
- Verwenden der INDEX_CLEANUP FALSE-Option – Wenn die Indizes groß sind und vermutlich viel Zeit für die Fertigstellung benötigen, können Sie die Blockierung der Selbstbereinigung aufheben, indem Sie einen manuellen VACUUM FREEZE-Vorgang ausführen und dabei Indizes ausschließen. Diese Funktionalität ist in PostgreSQL-Version 12 und höheren Versionen verfügbar.

  Durch das Umgehen von Indizes können Sie den Bereinigungsprozess des inkonsistenten Index überspringen und das Wraparound-Problem abmildern. Das zugrundeliegende Problem mit der ungültigen Seite wird dadurch jedoch nicht gelöst. Um das Problem mit der ungültigen Seite vollständig zu beheben, müssen Sie den Index dennoch neu erstellen.

Außergewöhnlich hohe Transaktionsrate

In PostgreSQL können hohe Transaktionsraten die Leistung der Selbstbereinigung erheblich beeinträchtigen, was zu einer langsameren Bereinigung toter Tupel und einem erhöhten Risiko für einen Transaktions-ID-Wraparound führt. Sie können die Transaktionsrate überwachen, indem Sie die Differenz hinsichtlich `max(age(datfrozenxid))` zwischen zwei Zeiträumen, typischerweise pro Sekunde, messen. Darüber hinaus können Sie die folgenden Zählermetriken von RDS Performance Insights verwenden, um die Transaktionsrate (die Summe aus xact_commit und xact_rollback) zu messen, die die Gesamtzahl der Transaktionen darstellt.

Zähler	Typ	Einheit	Metrik
xact_commit	Transaktionen	Commits pro Sekunde	db.Transactions.xact_commit

Zähler	Typ	Einheit	Metrik
xact_rollback	Transaktionen	Rollbacks pro Sekunde	db.Transactions.xact_rollback

Ein schneller Anstieg deutet auf eine hohe Transaktionslast hin, die den Selbstbereinigungsvorgang überlasten und zu Aufblähungen, Sperrkonflikten und potenziellen Leistungsproblemen führen kann. Dies kann sich auf unterschiedliche Art und Weise negativ auf den Selbstbereinigungsprozess auswirken:

- Tabellenaktivität: Bei der spezifischen Tabelle, die bereinigt wird, kann es zu einem hohen Transaktionsvolumen kommen, was möglicherweise Verzögerungen nach sich zieht.
- Systemressourcen: Das gesamte System ist möglicherweise überlastet, sodass der Selbstbereinigungsprozess möglicherweise nur schwer auf die erforderlichen Ressourcen zugreifen kann, die für eine effiziente Durchführung erforderlich sind.

Ziehen Sie die folgenden Strategien in Betracht, damit der Selbstbereinigungsprozess effektiver verlaufen und seine Aufgaben ohne große Verzögerungen erfüllen kann.

1. Reduzieren Sie nach Möglichkeit die Transaktionsrate. Erwägen Sie, ähnliche Transaktionen nach Möglichkeit zu Stapeln zusammenzufassen oder zu gruppieren.
2. Nehmen Sie häufig aktualisierte Tabellen ins Visier und verlegen Sie deren manuelle nächtliche, wöchentlich oder zweiwöchentliche `VACUUM FREEZE`-Operationen auf Zeiträume außerhalb der Spitzenzeiten.
3. Erwägen Sie, Ihre Instance-Klasse hochzuskalieren, um mehr Systemressourcen für das hohe Transaktionsvolumen und die Selbstbereinigung zur Verfügung zu haben.

Beheben von Leistungsproblemen bei der Bereinigung in Aurora PostgreSQL

In diesem Abschnitt werden Faktoren erörtert, die häufig zu einer schlechteren Bereinigungsleistung führen. Zudem wird beschrieben, wie diese Probleme behoben werden können.

Themen

- Bereinigen großer Indizes
- Zu viele Tabellen oder Datenbanken zum Bereinigen

- Es läuft eine aggressive Bereinigung (um ein Wraparound zu verhindern)

Bereinigen großer Indizes

Der VACUUM-Prozess durchläuft aufeinanderfolgende Phasen: Initialisierung, Heap-Scanning, Index- und Heap-Bereinigung, Indexsäuberung, Heap-Kürzung und abschließende Bereinigung. Während des Heap-Scans bereinigt der Prozess Seiten, defragmentiert sie und friert sie ein. Nach Abschluss des Heap-Scans bereinigt VACUUM Indizes, gibt leere Seiten an das Betriebssystem zurück und führt abschließende Aufgaben wie das Bereinigen der freien Speicherzuweisung und das Aktualisieren von Statistiken durch.

Das Bereinigen von Indizes kann mehrere Durchläufe erfordern, wenn `maintenance_work_mem` (oder `autovacuum_work_mem`) nicht ausreicht, um den Index zu verarbeiten. In PostgreSQL 16 und früheren Versionen bedingte ein Speicherlimit von 1 GB für das Speichern toter Tupel-IDs häufig mehrere Durchläufe großer Indizes. PostgreSQL 17 führt `TidStore` ein, sodass kein Array mit einer einzelnen Zuweisung verwendet, sondern Speicher dynamisch zugewiesen wird. Dadurch wird die Beschränkung auf 1 GB aufgehoben, der Arbeitsspeicher effizienter genutzt und der Bedarf an mehreren Index-Scans pro Index reduziert.

Große Indizes können in PostgreSQL 17 immer noch mehrere Durchläufe erfordern, wenn der verfügbare Speicher nicht die gesamte Indexverarbeitung auf einmal bewältigen kann. In der Regel enthalten größere Indizes mehr tote Tupel, für die mehrere Durchläufe erforderlich sind.

Erkennen langsamer Bereinigungsvorgänge

Die `postgres_get_av_diag()`-Funktion erkennt, wenn Bereinigungsvorgänge aufgrund unzureichenden Speichers langsam ausgeführt werden. Weitere Informationen zu dieser Funktionen finden Sie unter Installieren von Überwachungs- und Diagnosetools für die Selbstbereinigung in Aurora PostgreSQL.

Die `postgres_get_av_diag()`-Funktion gibt die folgenden Hinweise aus, wenn der verfügbare Speicher nicht ausreicht, um die Indexbereinigung in einem einzigen Durchgang abzuschließen.

`rds_tools` 1.9

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_work_mem is XX might not be sufficient.
 Consider increasing the setting to XXX, and if necessary, scaling up the RDS instance
 class for more
       memory. The suggested value is an estimate based on the current number of dead
 tuples for the table being vacuumed, which might not fully reflect the latest state.
 Additionally, review the possibility of manual
       vacuum with exclusion of indexes using (VACUUM (INDEX_CLEANUP FALSE, VERBOSE
 TRUE) table_name;). For more information, see

       Working with PostgreSQL autovacuum in the Amazon Aurora User Guide.
```

> **Note**
>
> Die `postgres_get_av_diag()`-Funktion stützt sich auf `pg_stat_all_tables.n_dead_tup`, um die Menge an Speicher zu schätzen, der für die Indexbereinigung benötigt wird.

Wenn die `postgres_get_av_diag()`-Funktion einen langsamen Bereinigungsvorgang feststellt, für den mehrere Index-Scans erforderlich sind, weil `autovacuum_work_mem` nicht ausreicht, wird folgende Meldung generiert:

```
NOTICE: Your vacuum is performing multiple index scans due to insufficient
 autovacuum_work_mem:XXX for index vacuuming.
       For more information, see Working with PostgreSQL autovacuum in the Amazon
 Amazon RDS User Guide.
```

Empfehlung

Sie können die folgenden Abhilfemaßnahmen mittels manuellem `VACUUM FREEZE` anwenden, um das Einfrieren der Tabelle zu beschleunigen.

Erhöhen des Speichers für die Bereinigung

Wie von der `postgres_get_av_diag()`-Funktion vorgeschlagen ist es ratsam, den `autovacuum_work_mem`-Parameter zu erhöhen, um möglichen Speicherbeschränkungen auf Instance-Ebene zu begegnen. Zwar ist `autovacuum_work_mem` ein dynamischer Parameter, dennoch muss beachtet werden, dass der Daemon für die Selbstbereinigung seine Worker neu starten muss, damit die neuen Speichereinstellungen wirksam werden. So können Sie dies erreichen:

1. Bestätigen Sie, dass die neue Einstellung festgelegt wurde.
2. Beenden Sie die Prozesse, die derzeit die Selbstbereinigung durchführen.

Dieser Ansatz stellt sicher, dass die angepasste Speicherzuweisung auf neue Selbstbereinigungsvorgänge angewendet wird.

Um unmittelbarere Ergebnisse zu erzielen, sollten Sie erwägen, manuell eine `VACUUM FREEZE`-Operation mit einer höheren `maintenance_work_mem`-Einstellung innerhalb Ihrer Sitzung durchzuführen:

```
SET maintenance_work_mem TO '1GB';
VACUUM FREEZE VERBOSE table_name;
```

Wenn Sie Amazon RDS verwenden und feststellen, dass Sie zusätzlichen Speicher benötigen, um höhere Werte für `maintenance_work_mem` oder `autovacuum_work_mem` unterstützen zu können, sollten Sie ein Upgrade auf eine Instance-Klasse mit mehr Speicher in Betracht ziehen. Dadurch können die erforderlichen Ressourcen bereitgestellt werden, um sowohl die manuellen als auch die automatischen Bereinigungsvorgänge zu verbessern, was insgesamt zu einer verbesserten Bereinigungs- und Datenbankleistung führt.

Deaktivieren von INDEX_CLEANUP

Ein manuelles `VACUUM` in PostgreSQL-Version 12 und höher ermöglicht das Überspringen der Indexbereinigungsphase, während die Notfall-Selbstbereinigung in PostgreSQL-Version 14 und höher dies automatisch auf der Grundlage des Parameters `vacuum_failsafe_age` erledigt.

> **Warning**
>
> Das Überspringen der Indexbereinigung kann zu einer Aufblähung des Index führen und sich negativ auf die Abfrageleistung auswirken. Um dem entgegenzuwirken, sollten Sie erwägen, die betroffenen Indizes während eines Wartungsfensters neu zu indizieren oder zu bereinigen.

Weitere Anleitungen zum Umgang mit großen Indizes finden Sie in der Dokumentation zu Verwalten der automatischen Bereinigung mit großen Indizes .

Parallele Indexbereinigung

Ab PostgreSQL 13 können Indizes standardmäßig mit manuellem `VACUUM` parallel bereinigt und gesäubert werden, wobei jedem Index ein Bereinigungs-Worker zugewiesen wird. Damit PostgreSQL jedoch feststellen kann, ob ein Bereinigungsvorgang für eine parallele Ausführung infrage kommt, müssen bestimmte Kriterien erfüllt sein:

- Es müssen mindestens zwei Indizes vorhanden sein.
- Der `max_parallel_maintenance_workers`-Parameter sollte auf mindestens 2 gesetzt werden.
- Die Indexgröße muss den `min_parallel_index_scan_size`-Grenzwert überschreiten, der standardmäßig 512 KB beträgt.

Sie können die `max_parallel_maintenance_workers`-Einstellung auf der Grundlage der Anzahl der auf Ihrer Amazon-RDS-Instance verfügbaren vCPUs und der Anzahl der Indizes in der Tabelle anpassen, um die Bearbeitungszeit für das Bereinigen zu optimieren.

Weitere Informationen finden Sie unter Parallele Selbstbereinigung in Amazon RDS für PostgreSQL und Amazon Aurora PostgreSQL.

Zu viele Tabellen oder Datenbanken zum Bereinigen

Wie in der PostgreSQL-Dokumentation Der Selbstbereinigungsdaemon angegeben, durchläuft der Bereinigungsdaemon mehrere Prozesse. Dazu gehört auch ein persistenter Launcher für die Selbstbereinigung, der für das Starten von Selbstbereinigungs-Worker-Prozessen für jede Datenbank innerhalb des Systems verantwortlich ist. Der Launcher plant, dass diese Worker ungefähr alle `autovacuum_naptime` Sekunden pro Datenbank initiiert werden.

Bei „N“ Datenbanken startet ungefähr alle [`autovacuum_naptime`/N Sekunden] ein neuer Worker. Die Gesamtzahl der gleichzeitigen Worker ist jedoch durch die `autovacuum_max_workers`-Einstellung begrenzt. Wenn die Anzahl der Datenbanken oder Tabellen, die bereinigt werden müssen, diesen Grenzwert überschreitet, wird die nächste Datenbank oder Tabelle verarbeitet, sobald ein Worker verfügbar ist.

Wenn viele große Tabellen oder Datenbanken gleichzeitig bereinigt werden müssen, kann es sein, dass alle verfügbaren Bereinigungs-Worker für einen längeren Zeitraum ausgelastet sind, was die Wartung anderer Tabellen und Datenbanken verzögert. In Umgebungen mit hohen Transaktionsraten kann dieser Engpass schnell eskalieren und möglicherweise zu komplexeren Bereinigungsproblemen innerhalb Ihrer Amazon-RDS-Instance führen.

Wenn `postgres_get_av_diag()` eine große Anzahl von Tabellen oder Datenbanken erkennt, wird folgende Empfehlung bereitgestellt:

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_max_workers:3 might not be sufficient.
 Consider increasing the setting and, if necessary, consider scaling up the Amazon RDS
 instance class for more workers.
```

Empfehlung

Erhöhen von autovacuum_max_workers

Um das Bereinigen zu beschleunigen, empfehlen wir, den `autovacuum_max_workers`-Parameter so anzupassen, dass mehr gleichzeitige Worker möglich sind. Wenn weiterhin Leistungsengpässe bestehen, sollten Sie erwägen, Ihre Amazon-RDS-Instance auf eine Klasse mit mehr vCPUs hochzuskalieren, wodurch die Möglichkeiten der Parallelverarbeitung weiter optimiert werden können.

Es läuft eine aggressive Bereinigung (um ein Wraparound zu verhindern)

Das Alter der Datenbank (MaximumUsedTransactionIDs) in PostgreSQL nimmt nur ab, wenn erfolgreich eine aggressive Bereinigung (zur Wraparound-Verhinderung) durchgeführt wird. Bis diese Bereinigung beendet ist, wird das Alter je nach Transaktionsrate weiter zunehmen.

Die `postgres_get_av_diag()`-Funktion generiert folgenden `NOTICE`, wenn sie eine aggressive Bereinigung erkennt. Sie löst diese Ausgabe jedoch erst aus, wenn die Bereinigung mindestens zwei Minuten lang aktiv war.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

Weitere Informationen zu aggressivem Bereinigungen finden Sie unter Wenn bereits eine aggressive Bereinigung läuft.

Mit der folgenden Abfrage können Sie überprüfen, ob eine aggressive Bereinigung läuft:

```
SELECT
    a.xact_start AS start_time,
    v.datname "database",
    a.query,
    a.wait_event,
    v.pid,
    v.phase,
```

```
    v.relid::regclass,
    pg_size_pretty(pg_relation_size(v.relid)) AS heap_size,
    (
        SELECT
            string_agg(pg_size_pretty(pg_relation_size(i.indexrelid)) || ':' ||
 i.indexrelid::regclass || chr(10), ', ')
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS index_sizes,
    trunc(v.heap_blks_scanned * 100 / NULLIF(v.heap_blks_total, 0)) AS step1_scan_pct,
    v.index_vacuum_count || '/' || (
        SELECT
            count(*)
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS step2_vacuum_indexes,
    trunc(v.heap_blks_vacuumed * 100 / NULLIF(v.heap_blks_total, 0)) AS
 step3_vacuum_pct,
    age(CURRENT_TIMESTAMP, a.xact_start) AS total_time_spent_sofar
FROM
    pg_stat_activity a
    INNER JOIN pg_stat_progress_vacuum v ON v.pid = a.pid;
```

Sie können anhand der Abfragespalte in der Ausgabe feststellen, ob es sich um eine aggressive Bereinigung (zur Wraparound-Verhinderung) handelt. Der Ausdruck „zur Wraparound-Verhinderung“ weist darauf hin, dass es sich um eine aggressive Bereinigung handelt.

```
query                 | autovacuum: VACUUM public.t3 (to prevent wraparound)
```

Angenommen, Sie haben einen Blocker mit einem Transaktionsalter von 1 Milliarde und eine Tabelle, die eine aggressive Bereinigung erfordert, um einen Wraparound im gleichen Transaktionsalter zu verhindern. Darüber hinaus gibt es einen weiteren Blocker mit einem Transaktionsalter von 750 Millionen. Nach der Aufhebung des Blockers mit einem Transaktionsalter von 1 Milliarde wird das Transaktionsalter nicht sofort auf 750 Millionen sinken. Es wird so lange hoch bleiben, bis die Tabelle, für die die aggressive Bereinigung erforderlich ist, oder eine andere Transaktion mit einem Alter von über 750 Millionen abgeschlossen ist. Während dieses Zeitraums wird das Transaktionsalter Ihres PostgreSQL-Clusters weiter steigen. Sobald der Bereinigungsvorgang abgeschlossen ist, wird das

Transaktionsalter auf 750 Millionen sinken, aber es wird wieder steigen, bis das weitere Bereinigen abgeschlossen ist. Dieser Zyklus wird so lange fortgesetzt, wie diese Bedingungen bestehen, bis das Transaktionsalter schließlich auf das für Ihre Amazon-RDS-Instance konfigurierte Niveau fällt, das durch `autovacuum_freeze_max_age` angegeben ist.

Erläuterung der HINWEIS-Meldungen in Aurora PostgreSQL

Die `postgres_get_av_diag()`-Funktion stellt die folgenden HINWEIS-Meldungen bereit:

Wenn für das Alter der Überwachungsschwellenwert noch nicht erreicht ist

Die Überwachungsschwellenwert für `postgres_get_av_diag()` zur Identifizierung von Blockern liegt standardmäßig bei 500 Millionen Transaktionen. Wenn `postgres_get_av_diag()` den folgenden HINWEIS generiert, bedeutet dies, dass das Transaktionsalter diesen Schwellenwert noch nicht erreicht hat.

```
NOTICE: postgres_get_av_diag() checks for blockers that prevent aggressive vacuums
 only, it does so only after exceeding dvb_threshold which is 500,000,000 and age of
 this PostgreSQL cluster is currently at 2.
```

Keine Verbindung zur Datenbank mit dem Alter der ältesten Transaktions-ID

Die `postgres_get_av_diag()`-Funktion liefert die genaueste Ausgabe, wenn sie mit der Datenbank mit dem Alter der ältesten Transaktions-ID verbunden ist. Die Datenbank mit dem ältesten von `postgres_get_av_diag()` gemeldeten Transaktions-ID-Alter unterscheidet sich in Ihrem Fall von „my_database“. Wenn Sie nicht mit der richtigen Datenbank verbunden sind, wird der folgende HINWEIS generiert:

```
NOTICE: You are not connected to the database with the age of oldest transaction
 ID. Connect to my_database database and run postgres_get_av_diag() for accurate
 reporting.
```

Aus folgenden Gründen ist es wichtig, eine Verbindung zur Datenbank mit dem ältesten Transaktionsalter herzustellen:

- Identifizierung temporärer Tabellenblocker: Da die Metadaten für temporäre Tabellen für jede Datenbank spezifisch sind, befinden sie sich normalerweise in der Datenbank, in der sie erstellt wurden. Wenn jedoch eine temporäre Tabelle der primäre Blocker ist und sich in der Datenbank mit der ältesten Transaktion befindet, könnte dies zu falschen Annahmen führen. Durch die Verbindung mit der richtigen Datenbank wird die genaue Identifizierung des Blockers für temporäre Tabellen gewährleistet.

- Diagnose langsamer Bereinigungen: Die Index-Metadaten und die Informationen zur Anzahl der Tabellen sind datenbankspezifisch und für die Diagnose von Problemen im Zusammenhang mit langsamen Bereinigungen erforderlich.

Die Datenbank mit der ältesten Transaktion nach Alter befindet sich in einer rdsadmin- oder template0-Datenbank

In bestimmten Fällen kann die `rdsadmin`- oder die `template0`-Datenbank als die Datenbank mit dem ältesten Transaktions-ID-Alter identifiziert werden. In diesem Fall gibt `postgres_get_av_diag()` folgenden HINWEIS aus:

```
NOTICE: The database with the age of oldest transaction ID is rdsadmin or template0,
 reach out to support if the reported blocker is in rdsadmin or template0.
```

Stellen Sie sicher, dass der aufgelistete Blocker nicht aus einer dieser beiden Datenbanken stammt. Wenn gemeldet wird, dass der Blocker in `rdsadmin` oder `template0` vorhanden ist, wenden Sie sich an den Support, da Benutzer nicht auf diese Datenbanken zugreifen können und ein Eingreifen erforderlich ist.

Es ist sehr unwahrscheinlich, dass die `rdsadmin`-Datenbank oder die `template0`-Datenbank einen primären Blocker enthält.

Wenn bereits eine aggressive Bereinigung läuft

Die `postgres_get_av_diag()`-Funktion ist so konzipiert, dass sie es meldet, wenn ein aggressiver Bereinigungsprozess läuft, löst diese Ausgabe jedoch erst aus, wenn die Bereinigungsfunktion seit mindestens 1 Minute aktiv ist. Diese absichtliche Verzögerung trägt dazu bei, die Wahrscheinlichkeit falsch-positiver Ergebnisse zu verringern. Durch das Warten stellt die Funktion sicher, dass nur effektive, signifikante Bereinigungen gemeldet werden, was zu einer genaueren und zuverlässigeren Überwachung der Bereinigungsaktivität führt.

Die `postgres_get_av_diag()`-Funktion generiert den folgenden HINWEIS, wenn sie feststellt, dass eine oder mehrere aggressive Bereinigungen laufen.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

Wie im HINWEIS angegeben sollten Sie die Leistung der Bereinigung weiterhin überwachen. Mehr über aggressive Bereinigungen erfahren Sie unter Es läuft eine aggressive Bereinigung (um ein Wraparound zu verhindern)

Wenn die Selbstbereinigung deaktiviert ist

Die `postgres_get_av_diag()`-Funktion generiert den folgenden HINWEIS, wenn die Selbstbereinigung auf Ihrer Datenbank-Instance deaktiviert ist:

```
NOTICE: Autovacuum is OFF, we strongly recommend to enable it, no restart is
 necessary.
```

Die Selbstbereinigung ist eine wichtige Funktion Ihrer Instance von Aurora PostgreSQL, die einen reibungslosen Datenbankbetrieb gewährleistet. Sie entfernt automatisch alte Zeilenversionen, fordert Speicherplatz zurück, verhindert ein Aufblähen von Tabellen und trägt so dazu bei, dass Tabellen und Indizes zwecks optimaler Leistung effizient bleiben. Darüber hinaus schützt sie vor einem Transaktions-ID-Wraparound, wodurch Transaktionen auf Ihrer Amazon-RDS-Instance gestoppt werden können. Die Deaktivierung der Selbstbereinigung kann zu langfristigen Einbußen bei der Datenbankleistung und -stabilität führen. Wir empfehlen Ihnen, sie ständig aktiviert zu lassen. Weitere Informationen finden Sie unter Grundlegendes zur Selbstbereinigung in Aurora PostgreSQL-Umgebungen.

> **Note**
>
> Durch das Deaktivieren der Selbstbereinigung werden aggressive Bereinigungen nicht gestoppt. Diese treten weiterhin auf, sobald Ihre Tabellen den `autovacuum_freeze_max_age`-Schwellenwert erreichen.

Die Anzahl der verbleibenden Transaktionen ist äußerst gering

Die `postgres_get_av_diag()`-Funktion generiert den folgenden HINWEIS, wenn eine Wraparound-Bereinigung unmittelbar bevorsteht. Dieser HINWEIS wird ausgegeben, wenn Ihre Amazon-RDS-Instance 100 Millionen Transaktionen davon entfernt ist, neue Transaktionen möglicherweise abzulehnen.

```
WARNING: Number of transactions remaining is critically low, resolve issues with
 autovacuum or perform manual VACUUM FREEZE before your instance stops accepting
 transactions.
```

Ihr sofortiges Handeln ist erforderlich, um Ausfallzeiten der Datenbank zu vermeiden. Sie sollten Ihre Bereinigungsvorgänge genau überwachen und erwägen, manuell einen `VACUUM FREEZE`-Vorgang in der betroffenen Datenbank zu initiieren, um Transaktionsfehler zu vermeiden.

Verwaltung von TOAST-OID-Konflikten in

TOAST (The Oversized-Attribute Storage Technique) ist eine PostgreSQL-Funktion, die für die Verarbeitung großer Datenwerte entwickelt wurde, die die typische Datenbankblockgröße von 8 KB überschreiten. PostgreSQL erlaubt nicht, dass sich physische Zeilen über mehrere Blöcke erstrecken. Die Blockgröße dient als Obergrenze für die Zeilengröße. TOAST überwindet diese Einschränkung, indem große Feldwerte in kleinere Blöcke aufgeteilt werden. Es speichert sie separat in einer speziellen TOAST-Tabelle, die mit der Haupttabelle verknüpft ist. Weitere Informationen finden Sie in der Dokumentation zum PostgreSQL TOAST-Speichermechanismus und zur Implementierung.

Themen

- TOAST-Operationen verstehen
- Identifizierung von Leistungsherausforderungen
- Empfehlungen
- Überwachen

TOAST-Operationen verstehen

TOAST führt eine Komprimierung durch und speichert große Feldwerte außerhalb der Zeile. TOAST weist jedem Block übergroßer Daten, die in der TOAST-Tabelle gespeichert sind, eine eindeutige OID (Object Identifier) zu. Die Haupttabelle speichert die TOAST-Wert-ID und die Beziehungs-ID auf der Seite, um auf die entsprechende Zeile in der TOAST-Tabelle zu verweisen. Dadurch kann PostgreSQL diese TOAST-Chunks effizient lokalisieren und verwalten. Wenn die TOAST-Tabelle jedoch wächst, besteht die Gefahr, dass das System die verfügbaren OIDs Ressourcen ausschöpft, was sowohl zu Leistungseinbußen als auch zu potenziellen Ausfallzeiten aufgrund der OID-Erschöpfung führt.

Objektkennungen in TOAST

Ein Object Identifier (OID) ist ein systemweiter eindeutiger Identifier, der von PostgreSQL verwendet wird, um Datenbankobjekte wie Tabellen, Indizes und Funktionen zu referenzieren. Diese Identifikatoren spielen eine wichtige Rolle bei den internen Vorgängen von PostgreSQL und ermöglichen es der Datenbank, Objekte effizient zu lokalisieren und zu verwalten.

Für Tabellen mit Datensätzen, die für das Toasten in Frage kommen, weist PostgreSQL jedem übergroßen Datenblock, der in der zugehörigen TOAST-Tabelle gespeichert ist, eine eindeutige

Identifizierung OIDs zu. Das System verknüpft jeden Chunk mit einem`chunk_id`, was PostgreSQL hilft, diese Chunks effizient in der TOAST-Tabelle zu organisieren und zu lokalisieren.

Identifizierung von Leistungsherausforderungen

Das OID-Management von PostgreSQL stützt sich auf einen globalen 32-Bit-Zähler, sodass es nach der Generierung von 4 Milliarden eindeutigen Werten einen Umbruch durchführt. Der Datenbank-Cluster teilt sich diesen Zähler, aber die OID-Zuweisung umfasst bei TOAST-Vorgängen zwei Schritte:

- Globaler Zähler für die Zuweisung — Der globale Zähler weist dem gesamten Cluster eine neue OID zu.
- Lokale Suche nach Konflikten — Die TOAST-Tabelle stellt sicher, dass die neue OID nicht mit der in der jeweiligen Tabelle OIDs bereits verwendeten OID in Konflikt steht.

Leistungseinbußen können auftreten, wenn:

- Die TOAST-Tabelle weist eine hohe Fragmentierung oder eine hohe OID-Auslastung auf, was zu Verzögerungen bei der Zuweisung der OID führt.
- OIDs In Umgebungen mit hoher Datenfluktuation oder großen Tabellen, in denen TOAST häufig verwendet wird, werden vom System häufig Zuweisungen vorgenommen und wiederverwendet.

Weitere Informationen finden Sie in der Dokumentation zur PostgreSQL-TOAST-Tabellengröße und zur OID-Zuweisung:

Ein globaler Zähler generiert die Werte OIDs und schließt sie etwa alle 4 Milliarden ab, sodass das System von Zeit zu Zeit erneut einen bereits verwendeten Wert generiert. PostgreSQL erkennt das und versucht es erneut mit der nächsten OID. Ein langsames INSERT kann auftreten, wenn es eine sehr lange Reihe von verwendeten OID-Werten ohne Lücken in der TOAST-Tabelle gibt. Diese Probleme werden immer ausgeprägter, je größer der OID-Speicherplatz ist, was zu langsameren Einfügungen und Aktualisierungen führt.

Identifizierung des Problems

- Einfache `INSERT` Aussagen dauern auf inkonsistente und zufällige Weise deutlich länger als gewöhnlich.
- Verzögerungen treten nur bei `INSERT` und bei `UPDATE` Anweisungen auf, die TOAST-Operationen beinhalten.

- Die folgenden Protokolleinträge erscheinen in PostgreSQL-Protokollen, wenn das System Schwierigkeiten hat, verfügbare OIDs TOAST-Tabellen zu finden:

```
LOG: still searching for an unused OID in relation "pg_toast_20815"
DETAIL: OID candidates have been checked 1000000 times, but no unused OID has been
 found yet.
```

- Performance Insights weist auf eine hohe Anzahl von durchschnittlichen aktiven Sitzungen (AAS) hin, die mit Ereignissen verknüpft sind `LWLock:buffer_io` und `LWLock:OidGenLock` warten.

  Sie können die folgende SQL-Abfrage ausführen, um lang andauernde INSERT-Transaktionen mit Warteereignissen zu identifizieren:

```
SELECT
    datname AS database_name,
    usename AS database_user,
    pid,
    now() - pg_stat_activity.xact_start AS transaction_duration,
    concat(wait_event_type, ':', wait_event) AS wait_event,
    substr(query, 1, 30) AS TRANSACTION,
    state
FROM
    pg_stat_activity
WHERE (now() - pg_stat_activity.xact_start) > INTERVAL '60 seconds'
    AND state IN ('active', 'idle in transaction', 'idle in transaction (aborted)',
 'fastpath function call', 'disabled')
    AND pid <> pg_backend_pid()
AND lower(query) LIKE '%insert%'
ORDER BY
    transaction_duration DESC;
```

  Beispiel für Abfrageergebnisse, die INSERT-Operationen mit längeren Wartezeiten anzeigen:

```
 database_name |  database_user  |  pid  | transaction_duration |     wait_event
 |          transaction           | state
---------------+-----------------+-------+----------------------
+---------------------+--------------------------------+--------
 postgres       | db_admin_user| 70965 | 00:10:19.484061      | LWLock:buffer_io    |
 INSERT INTO "products" (......... | active
 postgres       | db_admin_user| 69878 | 00:06:14.976037      | LWLock:buffer_io    |
 INSERT INTO "products" (......... | active
```

```
postgres       | db_admin_user| 68937 | 00:05:13.942847      | :                  |
INSERT INTO "products" (......... | active
```

Das Problem isolieren

- Test small insert — Fügt einen Datensatz ein, der kleiner als der `toast_tuple_target` Schwellenwert ist. Denken Sie daran, dass die Komprimierung vor der TOAST-Speicherung angewendet wird. Wenn dies ohne Leistungsprobleme funktioniert, hängt das Problem mit TOAST-Vorgängen zusammen.
- Neue Tabelle testen — Erstellen Sie eine neue Tabelle mit derselben Struktur und fügen Sie einen Datensatz ein, der größer als `toast_tuple_target` ist. Wenn das ohne Probleme funktioniert, ist das Problem auf die OID-Zuordnung der Originaltabelle beschränkt.

Empfehlungen

Die folgenden Lösungsansätze können dabei helfen, Probleme mit TOAST-OID-Konflikten zu lösen.

- Datenbereinigung und Archivierung — Überprüfen und löschen Sie alle veralteten oder unnötigen Daten, um sie OIDs für die future Verwendung freizugeben, oder archivieren Sie sie. Erwägen Sie die folgenden Einschränkungen:
  - Eingeschränkte Skalierbarkeit, da eine future Bereinigung möglicherweise nicht immer möglich ist.
  - Möglicher lang andauernder VACUUM-Vorgang, um die resultierenden toten Tupel zu entfernen.
- In eine neue Tabelle schreiben — Erstellen Sie eine neue Tabelle für future Einfügungen und verwenden Sie eine `UNION ALL` Ansicht, um alte und neue Daten für Abfragen zu kombinieren. In dieser Ansicht werden die kombinierten Daten aus alten und neuen Tabellen dargestellt, sodass Abfragen auf sie als eine einzige Tabelle zugreifen können. Erwägen Sie die folgenden Einschränkungen:
  - Aktualisierungen der alten Tabelle können immer noch zu einer Überlastung der OID führen.
- Partition oder Shard — Partitionieren Sie die Tabellen- oder Shard-Daten, um die Skalierbarkeit und Leistung zu verbessern. Erwägen Sie die folgenden Einschränkungen:
  - Höhere Komplexität bei der Abfragelogik und Wartung, potenzielle Notwendigkeit von Anwendungsänderungen, um partitionierte Daten korrekt verarbeiten zu können.

Überwachen

Verwendung von Systemtabellen

Sie können die Systemtabellen von PostgreSQL verwenden, um das Wachstum der OID-Nutzung zu überwachen.

> **Warning**
>
> Je nach Anzahl der Einträge OIDs in der TOAST-Tabelle kann es einige Zeit dauern, bis der Vorgang abgeschlossen ist. Wir empfehlen, die Überwachung außerhalb der Geschäftszeiten einzuplanen, um die Auswirkungen zu minimieren.

Der folgende anonyme Block zählt die Anzahl der in jeder TOAST-Tabelle OIDs verwendeten Zeichen und zeigt die Informationen zur übergeordneten Tabelle an:

```
DO $$
DECLARE
    r record;
    o bigint;
    parent_table text;
    parent_schema text;
BEGIN
    SET LOCAL client_min_messages TO notice;
    FOR r IN
    SELECT
        c.oid,
        c.oid::regclass AS toast_table
    FROM
        pg_class c
    WHERE
        c.relkind = 't'
        AND c.relowner != 10 LOOP
            -- Fetch the number of distinct used OIDs (chunk IDs) from the TOAST table
            EXECUTE 'SELECT COUNT(DISTINCT chunk_id) FROM ' || r.toast_table INTO o;
            -- If there are used OIDs, find the associated parent table and its schema
            IF o <> 0 THEN
                SELECT
                    n.nspname,
                    c.relname INTO parent_schema,
                    parent_table
```

```
                FROM
                    pg_class c
                    JOIN pg_namespace n ON c.relnamespace = n.oid
                WHERE
                    c.reltoastrelid = r.oid;
                -- Raise a concise NOTICE message
                RAISE NOTICE 'Parent schema: % | Parent table: % | Toast table: %
 | Number of used OIDs: %', parent_schema, parent_table, r.toast_table, TO_CHAR(o,
 'FM9,999,999,999,999');
            END IF;
        END LOOP;
END
$$;
```

Beispielausgabe mit OID-Nutzungsstatistiken nach TOAST-Tabelle:

```
NOTICE:  Parent schema: public | Parent table: my_table | Toast table:
 pg_toast.pg_toast_16559 | Number of used OIDs: 45,623,317
NOTICE:  Parent schema: public | Parent table: my_table1 | Toast table:
 pg_toast.pg_toast_45639925 | Number of used OIDs: 10,000
NOTICE:  Parent schema: public | Parent table: my_table2 | Toast table:
 pg_toast.pg_toast_45649931 | Number of used OIDs: 1,000,000
DO
```

Der folgende anonyme Block ruft die maximal zugewiesene OID für jede nicht leere TOAST-Tabelle ab:

```
DO $$
DECLARE
    r record;
    o bigint;
    parent_table text;
    parent_schema text;
BEGIN
    SET LOCAL client_min_messages TO notice;
    FOR r IN
    SELECT
        c.oid,
        c.oid::regclass AS toast_table
    FROM
        pg_class c
    WHERE
        c.relkind = 't'
```

```
        AND c.relowner != 10 LOOP
            -- Fetch the max(chunk_id) from the TOAST table
            EXECUTE 'SELECT max(chunk_id) FROM ' || r.toast_table INTO o;
            -- If there's at least one TOASTed chunk, find the associated parent table
 and its schema
            IF o IS NOT NULL THEN
                SELECT
                    n.nspname,
                    c.relname INTO parent_schema,
                    parent_table
                FROM
                    pg_class c
                    JOIN pg_namespace n ON c.relnamespace = n.oid
                WHERE
                    c.reltoastrelid = r.oid;
                -- Raise a concise NOTICE message
                RAISE NOTICE 'Parent schema: % | Parent table: % | Toast table:
 % | Max chunk_id: %', parent_schema, parent_table, r.toast_table, TO_CHAR(o,
 'FM9,999,999,999,999');
            END IF;
        END LOOP;
END
$$;
```

Beispielausgabe, die den maximalen Teil für TOAST-Tabellen IDs anzeigt:

```
NOTICE:  Parent schema: public | Parent table: my_table | Toast table:
 pg_toast.pg_toast_16559 | Max chunk_id: 45,639,907
NOTICE:  Parent schema: public | Parent table: my_table1 | Toast table:
 pg_toast.pg_toast_45639925 | Max chunk_id: 45,649,929
NOTICE:  Parent schema: public | Parent table: my_table2 | Toast table:
 pg_toast.pg_toast_45649931 | Max chunk_id: 46,649,935
DO
```

Verwendung von Performance-Insights

Die Warteereignisse `LWLock:buffer_io` `LWLock:OidGenLock` werden in Performance Insights bei Vorgängen angezeigt, für die neue Objektkennungen () OIDs zugewiesen werden müssen. High Average Active Sessions (AAS) für diese Ereignisse deuten in der Regel auf Konflikte bei der OID-Zuweisung und der Ressourcenverwaltung hin. Dies ist besonders häufig in Umgebungen mit hoher Datenfluktuation, umfangreicher Datennutzung oder häufiger Objekterstellung der Fall.

LWLockDies ist besonders häufig in Umgebungen mit hoher Datenfluktuation, umfangreicher Datennutzung oder häufiger Objekterstellung der Fall. ----sep----:buffer_io

`LWLock:buffer_io`ist ein Wartungsereignis, das auftritt, wenn eine PostgreSQL-Sitzung auf den Abschluss von I/O Vorgängen in einem gemeinsam genutzten Puffer wartet. Dies passiert normalerweise, wenn die Datenbank Daten von der Festplatte in den Speicher liest oder geänderte Seiten vom Speicher auf die Festplatte schreibt. Das `BufferIO` Wait-Ereignis gewährleistet Konsistenz, indem es verhindert, dass mehrere Prozesse während laufender I/O Operationen auf denselben Puffer zugreifen oder ihn ändern. Ein häufiges Auftreten dieses Warteereignisses kann auf Festplattenengpässe oder übermäßige I/O Aktivität bei der Datenbank-Arbeitslast hinweisen.

Während TOAST-Vorgängen:

- PostgreSQL weist OIDs große Objekte zu und stellt deren Einzigartigkeit sicher, indem es den Index der TOAST-Tabelle scannt.
- Große TOAST-Indizes erfordern möglicherweise den Zugriff auf mehrere Seiten, um die Eindeutigkeit der OID zu überprüfen. Dies führt zu mehr Festplatten-E/A, insbesondere wenn der Pufferpool nicht alle erforderlichen Seiten zwischenspeichern kann.

Die Größe des Indexes wirkt sich direkt auf die Anzahl der Pufferseiten aus, auf die während dieser Operationen zugegriffen werden muss. Selbst wenn der Index nicht aufgebläht ist, kann seine bloße Größe die Puffer-I/O erhöhen, insbesondere in Umgebungen mit hoher Parallelität oder hoher Kundenabwanderung. Weitere Informationen finden Sie unter: Anleitung zur Fehlerbehebung bei BufferiO-WarteereignissenLWLock.

LWLock:OidGenLock

`OidGenLock`ist ein Wartungsereignis, das auftritt, wenn eine PostgreSQL-Sitzung darauf wartet, einen neuen Objektbezeichner (OID) zuzuweisen. Diese Sperre stellt sicher, dass OIDs sie sequentiell und sicher generiert werden, sodass jeweils nur ein Prozess generiert OIDs werden kann.

Während TOAST-Vorgängen:

- OID-Zuweisung für Chunks in der TOAST-Tabelle — PostgreSQL weist Chunks in TOAST-Tabellen OIDs zu, wenn große Datensätze verwaltet werden. Jede OID muss eindeutig sein, um Konflikte im Systemkatalog zu vermeiden.
- Hohe Parallelität — Da der Zugriff auf den OID-Generator sequentiell erfolgt, kann es zu Konflikten kommen, wenn mehrere Sitzungen gleichzeitig Objekte erstellen OIDs, die dies erfordern.

`OidGenLock` Dies erhöht die Wahrscheinlichkeit, dass Sitzungen auf den Abschluss der OID-Zuweisung warten.

- Abhängigkeit vom Systemkatalogzugriff — Für die Zuweisung OIDs sind Aktualisierungen der gemeinsam genutzten Systemkatalogtabellen wie `pg_class` und `pg_type` erforderlich. Wenn diese Tabellen stark beansprucht werden (aufgrund häufiger DDL-Operationen), kann sich der Sperrkonflikt für erhöhen. `OidGenLock`
- Hohe Nachfrage nach OID-Zuweisungen — TOAST-Arbeitslasten mit großen Datensätzen erfordern eine konstante OID-Zuweisung, was zu mehr Konflikten führt.

Zusätzliche Faktoren, die den OID-Konflikt erhöhen:

- Häufige Objekterstellung — Workloads, bei denen häufig Objekte erstellt und gelöscht werden, wie z. B. temporäre Tabellen, verstärken den Konflikt auf dem globalen OID-Zähler.
- Globales Sperren von Leistungsindikatoren — Auf den globalen OID-Zähler wird seriell zugegriffen, um die Eindeutigkeit sicherzustellen, wodurch in Umgebungen mit hoher Parallelität ein einziger Streitpunkt entsteht.

Verwenden von Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL erweitert Ihren DB-Cluster von Aurora PostgreSQL um die Möglichkeit, Datenbankverbindungen von SQL-Server-Clients zu akzeptieren. Mit Babelfish können Anwendungen, die ursprünglich für SQL Server entwickelt wurden, direkt mit Aurora PostgreSQL mit wenigen Codeänderungen im Vergleich zu einer herkömmlichen Migration und ohne Änderung der Datenbanktreiber arbeiten. Weitere Informationen zur Migration von in eine VPC finden Sie unter Migrieren einer SQL-Server-Datenbank zu Babelfish for Aurora PostgreSQL.

Babelfish bietet einen zusätzlichen Endpunkt für einen Aurora-PostgreSQL-Datenbankcluster, der es erlaubt, das SQL-Server-Wire-Level-Protokoll und häufig verwendete SQL-Server-Anweisungen zu verstehen. Clientanwendungen, die das Wire-Protokoll Tabular Data Stream (TDS) verwenden, können sich nativ mit dem TDS-Listener-Port von Aurora PostgreSQL verbinden. Weitere Informationen zu TDS finden Sie unter [MS-TDS]: Tabellarisches Datenstromprotokoll auf der Microsoft-Website.

> **Note**
>
> Babelfish for Aurora PostgreSQL unterstützt TDS-Versionen 7.1 bis 7.4.

Babelfish bietet auch Zugriff auf Daten über die PostgreSQL-Verbindung. Standardmäßig sind beide von Babelfish unterstützten SQL-Dialekte über ihre nativen Wire-Protokolle an den folgenden Ports verfügbar:

- Der SQL-Server-Dialekt (T-SQL), Clients stellen eine Verbindung mit Port 1433 her.
- PostgreSQL-Dialekt (PL/pgSQL), Clients stellen eine Verbindung mit Port 5432 her.

Babelfish führt die Transact-SQL-Sprache (T-SQL) mit einigen Unterschieden aus. Weitere Informationen finden Sie unter Unterschiede zwischen Babelfish für Aurora PostgreSQL und SQL Server.

In den folgenden Abschnitten finden Sie Informationen zum Einrichten und Verwenden eines DB-Clusters von Babelfish for Aurora PostgreSQL.

Themen

- Babelfish-Einschränkungen
- Grundlagen der Babelfish-Architektur und -Konfiguration

- Erstellen eines DB-Clusters von Babelfish for Aurora PostgreSQL
- Migrieren einer SQL-Server-Datenbank zu Babelfish for Aurora PostgreSQL
- Datenbankauthentifizierung mit Babelfish für Aurora PostgreSQL
- Verbinden mit einem Babelfish-DB-Cluster
- Arbeiten mit Babelfish
- Fehlerbehebung bei Babelfish
- Deaktivieren von Babelfish
- Verwalten von Versionsupdates von Babelfish für Aurora PostgreSQL
- Referenz für Babelfish for Aurora PostgreSQL

Babelfish-Einschränkungen

Bei Babelfish für Aurora PostgreSQL gelten aktuell die folgenden Einschränkungen:

- Folgende Aurora-Funktionen werden von Babelfish nicht unterstützt:
  - AWS Identity and Access Management
  - Datenbank-Aktivitätsstreams (DAS)
  - RDS-Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellt
  - RDS-Proxy mit RDS für SQL Server
  - SCRAM (Salted Challenge Response Authentication Mechanism)
  - Abfrage-Editor
- Babelfish bietet keine Unterstützung für die folgenden Anforderungen der Client-Treiber-API:
  - API-Anforderungen mit den Verbindungsattributen im Zusammenhang mit Microsoft Distributed Transaction Coordinator (MSDTC) werden nicht unterstützt. Dazu gehören XA-Aufrufe durch die SQLServer XAResource Klasse im SQL Server-JDBC-Treiber.
- Derzeit werden folgende Aurora-PostgreSQL-Erweiterungen von Babelfish nicht unterstützt:
  - `bloom`
  - `btree_gin`
  - `btree_gist`
  - `citext`
  - `cube`
  - `hstore`
  - `hypopg`
  - Logische Replikation mithilfe von `pglogical`
  - `ltree`
  - `pgcrypto`
  - Abfrageplanverwaltung mithilfe von `apg_plan_mgmt`

  Weitere Informationen zu PostgreSQL-Erweiterungen finden Sie unter Arbeiten mit Erweiterungen und Fremddaten-Wrappern.
- Der Open-Source-JTDS-Treiber, der als Alternative zum JDBC-Treiber von Microsoft konzipiert wurde, wird nicht unterstützt.

Grundlagen der Babelfish-Architektur und -Konfiguration

Sie verwalten den DB-Cluster der mit Aurora PostgreSQL kompatiblen Edition, auf dem Babelfish ausgeführt wird, genau wie jeden anderen Aurora-DB-Cluster. Sie profitieren also von der Skalierbarkeit, der hohen Verfügbarkeit mit Failover-Unterstützung und der integrierten Replikation, die von einem Aurora-DB-Cluster bereitgestellt wird. Weitere Informationen zu diesen Funktionen finden Sie unter Verwalten von Performance und Skalierung für einen Aurora-DB-Cluster, Hohe Verfügbarkeit für Amazon Aurora und Replikation mit Amazon Aurora. Sie haben auch Zugriff auf viele andere AWS-Tools und -Dienstprogramme, einschließlich der folgenden:

- Amazon CloudWatch ist ein Überwachungs- und Beobachtungsservice, der Ihnen Daten und umsetzbare Erkenntnisse bietet. Weitere Informationen finden Sie unter Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch.
- Performance Insights ist eine Funktion zur Optimierung und Überwachung der Datenbank-Performance, mit der Sie die Belastung Ihrer Datenbank schnell beurteilen können. Weitere Informationen hierzu finden Sie unter Überwachung mit Performance Insights auf .
- Globale Aurora-Datenbanken umfassen mehrere AWS-Regionen, was globale Lesevorgänge mit niedriger Latenz ermöglicht und eine schnelle Wiederherstellung nach einem seltenen Ausfall ermöglicht, der sich auf die gesamte AWS-Region auswirken könnte. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.
- Das automatische Software-Patching hält Ihre Datenbank mit den neuesten Sicherheits- und Feature-Patches auf dem neuesten Stand, wenn sie verfügbar sind.
- Amazon-RDS-Ereignisse informieren Sie per E-Mail oder SMS-Nachricht über wichtige Datenbankereignisse wie ein automatisiertes Failover. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora-Ereignissen.

Im Folgenden erfahren Sie mehr über die Babelfish-Architektur und wie die von Ihnen migrierten SQL-Server-Datenbanken von Babelfish behandelt werden. Wenn Sie Ihren Babelfish-DB-Cluster erstellen, müssen Sie vorab einige Entscheidungen im Hinblick auf eine oder mehrere Datenbanken, Sortierungen und andere Details treffen.

Themen

- Babelfish-Architektur
- Einstellungen der DB-Cluster-Parametergruppe für Babelfish
- Verstehen der Kollationen in Babelfish für Aurora PostgreSQL

- Verwalten der Babelfish-Fehlerbehandlung mit Escape-Schraffuren

Babelfish-Architektur

Wenn Sie einen Aurora-PostgreSQL-Cluster erstellen, bei dem Babelfish aktiviert ist, stellt Aurora den Cluster mit einer PostgreSQL-Datenbank namens `babelfish_db` bereit. In dieser Datenbank befinden sich alle migrierten SQL Server-Objekte und Strukturen.

> **Note**
>
> In einem Aurora-PostgreSQL-Cluster ist der Datenbankname `babelfish_db` für Babelfish reserviert. Wenn Sie eine eigene „babelfish_db“-Datenbank auf einem Babelfish-DB-Cluster erstellen, wird verhindert, dass Aurora Babelfish erfolgreich bereitstellt.

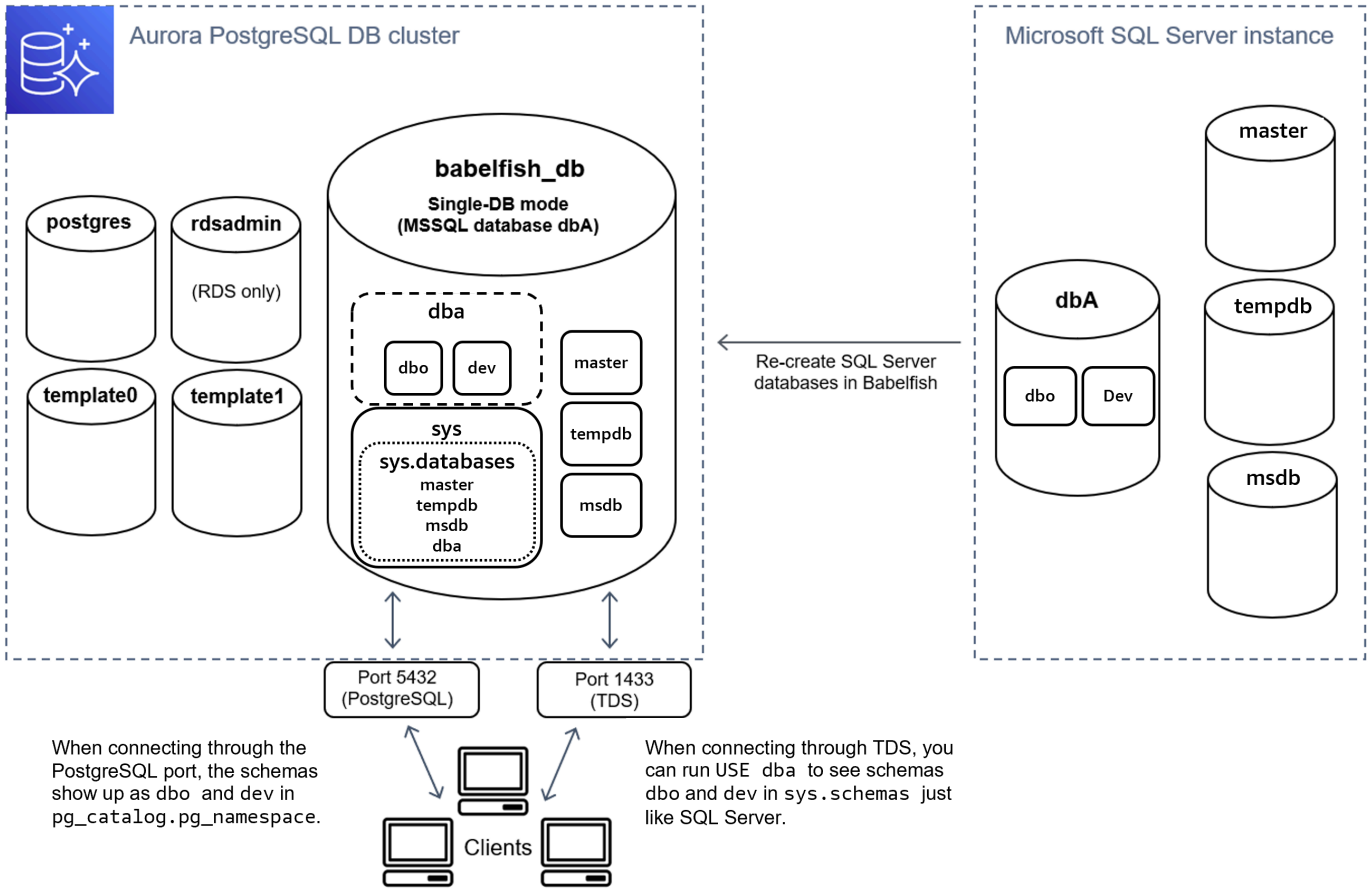
Wenn Sie eine Verbindung zum TDS-Port herstellen, wird die Sitzung in der `babelfish_db`-Datenbank. Von T-SQL sieht die Struktur ähnlich aus wie eine Verbindung mit einer SQL Server-Instanz. Die `master`-, `msdb`- und `tempdb`-Datenbanken und der `sys.databases`-Katalog sind zu sehen. Sie können zusätzliche Benutzerdatenbanken erstellen und mit der USE-Anweisung zwischen Datenbanken wechseln. Wenn Sie eine SQL Server-Benutzerdatenbank erstellen, wird sie in die `babelfish_db` PostgreSQL-Datenbank eingebracht. Ihre Datenbank behält datenbankübergreifende Syntax und Semantik bei, die denen von SQL Server entsprechen oder ähnlich sind.

Verwenden von Babelfish mit einer einzigen Datenbank oder mehreren Datenbanken

Wenn Sie einen Aurora-PostgreSQL-Cluster erstellen, der mit Babelfish verwendet werden soll, wählen Sie zwischen der Verwendung einer einzelnen SQL Server-Datenbank für sich selbst oder mehrere SQL Server-Datenbanken zusammen. Ihre Wahl wirkt sich darauf aus, wie die Namen von SQL Server-Schemas innerhalb der `babelfish_db`-Datenbank aus Aurora PostgreSQL erscheinen. Der Migrationsmodus wird im `migration_mode`-Parameter gespeichert. Sie dürfen diesen Parameter nicht ändern, nachdem Sie Ihren Cluster erstellt haben, da Sie ggf. den Zugriff auf alle zuvor erstellten SQL-Objekte verlieren.

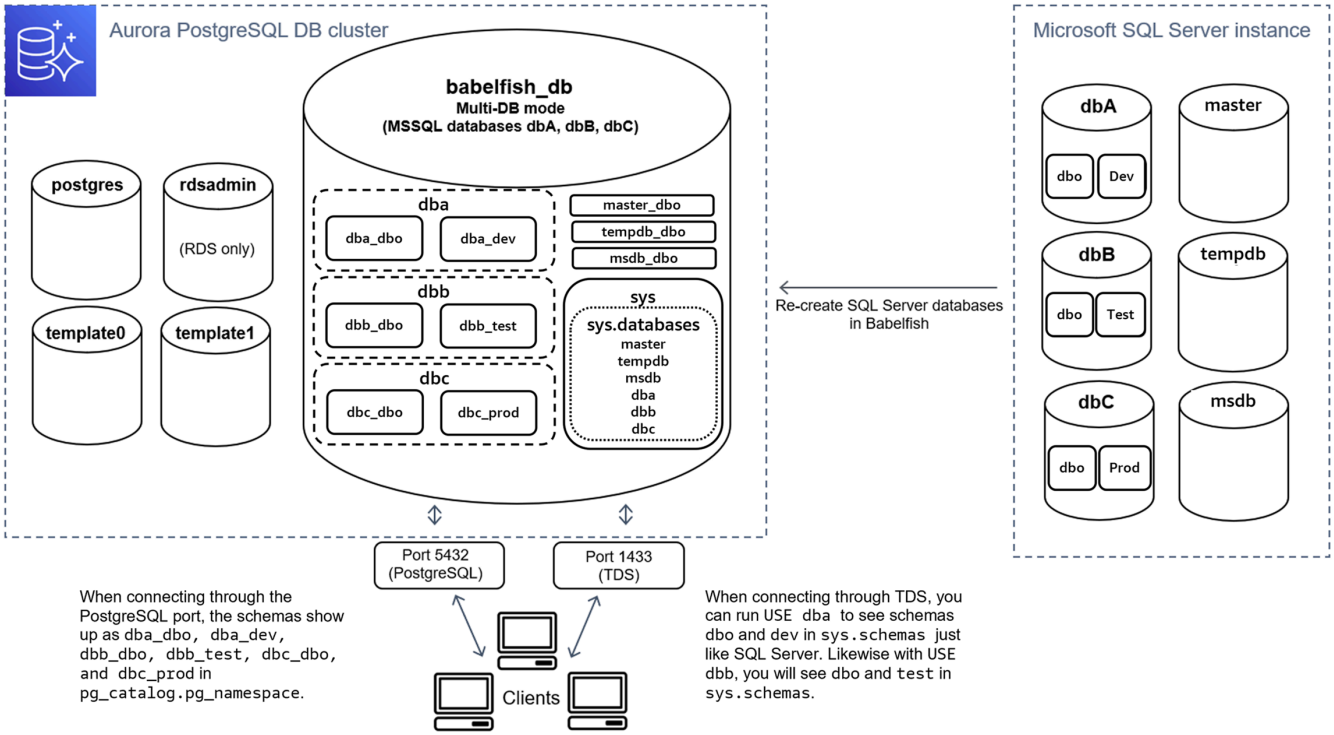
Im Einzeldatenbank-Modus bleiben die Schemanamen der SQL-Server-Datenbank in der `babelfish_db`-Datenbank von PostgreSQL unverändert. Wenn Sie nur eine einzelne Datenbank migrieren möchten, können die Schemanamen der migrierten Benutzerdatenbank in PostgreSQL mit

denselben Namen referenziert werden, die in SQL Server verwendet werden. Zum Beispiel: Das dbo- und `smith`-Schema befinden sich in der dbA-Datenbank.



Wenn Sie sich über TDS verbinden, können Sie `USE dba` starten, um Schemas dbo und `dev` von T-SQL zu sehen, wie Sie es in SQL Server tun würden. Die unveränderten Schemanamen sind auch von PostgreSQL aus sichtbar.

Im Mehrfachdatenbankmodus werden die Schemanamen von Benutzerdatenbanken zu `dbname_schemaname`, wenn von PostgreSQL aus darauf zugegriffen wird. Die Schemanamen bleiben gleich, wenn von T-SQL aus darauf zugegriffen wird.



Wie im Bild gezeigt, sind der Mehrfachdatenbankmodus und der Einzeldatenbankmodus identisch mit SQL Server, wenn Sie eine Verbindung über den TDS-Port herstellen und T-SQL verwenden. Beispielsweise listet `USE dbA` Schemata dbo und `dev` genau wie in SQL Server auf. Die zugeordneten Schemanamen wie `dba_dbo` und `dba_dev` sind von PostgreSQL aus sichtbar.

Jede Datenbank enthält immer noch Ihre Schemas. Der Name jeder Datenbank wird dem Namen des SQL Server-Schemas vorangestellt, wobei ein Unterstrich als Trennzeichen verwendet wird, z. B.:

- `dba` enthält `dba_dbo` und `dba_dev`.
- dbb enthält `dbb_dbo` und `dbb_test`.
- `dbc` enthält `dbc_dbo` und `dbc_prod`.

Innerhalb der `babelfish_db`-Datenbank muss der T-SQL-Benutzer immer noch `USE dbname` ausführen, um den Datenbankkontext zu ändern, so dass das Erscheinungsbild ähnlich wie bei SQL Server bleibt.

Auswahl eines Migrationsmodus

Jeder Migrationsmodus hat Vor- und Nachteile. Wählen Sie Ihren Migrationsmodus basierend auf der Anzahl der Benutzerdatenbanken, die Sie haben, und Ihren Migrationsplänen aus. Nachdem Sie

einen Cluster für die Verwendung mit Babelfish erstellt haben, dürfen Sie den Migrationsmodus nicht mehr ändern, da Sie ggf. den Zugriff auf alle zuvor erstellten SQL-Objekte verlieren. Berücksichtigen Sie bei der Auswahl eines Migrationsmodus die Anforderungen Ihrer Benutzerdatenbanken und Clients.

Wenn Sie einen Cluster zur Verwendung mit Babelfish erstellen, erstellt Aurora PostgreSQL die Systemdatenbanken `master` und `tempdb`. Wenn Sie Objekte in den Systemdatenbanken erstellt oder geändert haben (`master` oder `tempdb`), stellen Sie sicher, dass Sie diese Objekte in Ihrem neuen Cluster neu erstellen. Im Gegensatz zu SQL Server initialisiert Babelfish `tempdb` nach einem Neustart des Clusters nicht neu.

Verwenden Sie in den folgenden Fällen den Migrationsmodus einer einzelnen Datenbank:

- Wenn Sie eine einzelne SQL Server-Datenbank migrieren. Im Einzeldatenbankmodus sind migrierte Schemanamen identisch mit den ursprünglichen SQL-Server-Schemanamen. Dadurch werden Codeänderungen an vorhandenen SQL-Abfragen reduziert, wenn Sie sie für die Ausführung mit einer PostgreSQL-Verbindung optimieren möchten.
- Wenn Ihr Endziel eine vollständige Migration zur nativen Aurora PostgreSQL ist. Konsolidieren Sie Ihre Schemas vor der Migration in einem einzigen Schema (`dbo`) und migrieren Sie dann in einen einzelnen Cluster, um die erforderlichen Änderungen zu verringern.

Verwenden Sie in den folgenden Fällen den Migrationsmodus für mehrere Datenbanken:

- Wenn Sie die standardmäßige SQL-Server-Umgebung mit mehreren Benutzerdatenbanken in derselben Instance wünschen.
- Wenn mehrere Benutzerdatenbanken zusammen migriert werden müssen.

Einstellungen der DB-Cluster-Parametergruppe für Babelfish

Wenn Sie einen Aurora-PostgreSQL-DB-Cluster erstellen und Turn on Babelfish (Babelfish aktivieren) auswählen, wird beim Klicken auf Create new (Neu erstellen) automatisch eine DB-Cluster-Parametergruppe für Sie erstellt. Diese DB-Cluster-Parametergruppe basiert auf der DB-Cluster-Parametergruppe von Aurora PostgreSQL für die Aurora-PostgreSQL Version, die für die Installation ausgewählt wurde, z. B. Aurora PostgreSQL Version 14. Sie wird unter Verwendung der folgenden allgemeinen Konvention benannt:

```
custom-aurora-postgresql14-babelfish-compat-3
```

Sie können die folgenden Einstellungen während des Cluster-Erstellungsprozesses ändern, aber einige davon können nicht geändert werden, sobald sie in der benutzerdefinierten Parametergruppe gespeichert sind. Wählen Sie daher Folgendes sorgfältig aus:

- Einzelne Datenbank oder mehrere Datenbanken
- Standardgebietsschema für Sortierungen
- Name der Sortierung
- DB-Parametergruppe

Wenn Sie eine vorhandene DB-Cluster-Parametergruppe von Aurora PostgreSQL Version 13 oder höher verwenden möchten, bearbeiten Sie die Gruppe und legen Sie den `babelfish_status`-Parameter auf on fest. Geben Sie alle Babelfish-Optionen an, bevor Sie Ihren Aurora-PostgreSQL-Cluster erstellen. Weitere Informationen hierzu finden Sie unter Parametergruppen für Amazon Aurora.

Die folgenden Parameter steuern die Präferenzen von Babelfish. Sofern in der Beschreibung nicht anders angegeben, können Parameter geändert werden. Der Standardwert ist in der Beschreibung enthalten. Gehen Sie wie folgt vor, um die zulässigen Werte für jeden Parameter anzuzeigen:

> **Note**
>
> Wenn Sie eine neue DB-Parametergruppe mit einer DB-Instance verknüpfen, werden die geänderten statischen und dynamischen Parameter erst nach Neustart der DB-Instance angewendet. Wenn Sie jedoch dynamische Parameter in der DB-Parametergruppe ändern, nachdem Sie sie der DB-Instance zugeordnet haben, werden diese Änderungen sofort ohne Neustart angewendet.

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie die Parametergruppe für Ihren `default.aurora-postgresql14`-DB-Cluster in der Liste aus.
4. Geben Sie den Namen eines Parameters in das Suchfeld ein. Geben Sie beispielsweise `babelfishpg_tsql.default_locale` in das Suchfeld ein, um diesen Parameter, seinen Standardwert und die zulässigen Einstellungen anzuzeigen.

> **Note**
>
> Die globalen Datenbanken von Babelfish für Aurora PostgreSQL funktionieren nur in sekundären Regionen, wenn die folgenden Parameter in diesen Regionen aktiviert sind.

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tsql.apg_enable_correlated_scalar_transform	Ermöglicht dem Planer, korrelierte skalare Unterabfragen in Babelfish zu transformieren (Standardwert: on) (Zulässig: on, off)	dynamisch	true
babelfishpg_tsql.apg_enable_subquery_cache	Aktiviert die Verwendung von Cache für korrelierte skalare Unterabfragen in Babelfish (Standardwert: on) (Zulässig: on, off)	dynamisch	true
babelfishpg_tds.tds_default_numeric_scale	Legt den Standardmaßstab des numerischen Typs fest, der in	dynamisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
	den Metadaten der TDS-Spalte gesendet werden soll, wenn die Engine keine angibt. (Standardwert: 8) (Zulässig: 0–38)		
babelfishpg_tds.tds_default_numeric_precision	Eine Ganzzahl, die die Standardgenauigkeit des numerischen Typs festlegt, der in den Metadaten der TDS-Spalte gesendet werden soll, wenn die Engine keine angibt. (Standardwert: 38) (Zulässig: 1–38)	dynamisch	true
babelfishpg_tds.tds_default_packet_size	Eine Ganzzahl, die die Standardpaketgröße für die Verbindung von SQL-Server-Clients festlegt. (Standardwert: 4096) (Zulässig: 512–32767)	dynamisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tds.tds_default_protocol_version	Eine Ganzzahl, die eine Standardversion des TDS-Protokolls zum Verbinden von Clients festlegt. (Standardwert: DEFAULT) (Zulässig: TDSv7.0, TDSv7.1, TDSv7.1.1, TDSv7.2, TDSv7.3A, TDSv7.3B, TDSv7.4, DEFAULT)	dynamisch	true
babelfishpg_tds.default_server_name	Ein String, der den Standardnamen des Babelfish-Servers bezeichnet. (Standardwert: Microsoft SQL Server) (Zulässig: null)	dynamisch	true
babelfishpg_tds.tds_debug_log_level	Eine Ganzzahl, die die Protokollierungsstufe in TDS festlegt; 0 schaltet die Protokollierung aus. (Standardwert: 1) (Zulässig: 0, 1, 2, 3)	dynamisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tds.listen_adressen	Ein String, der den Host-Namen oder die IP-Adresse oder -Adressen festlegt, an denen TDS überwacht werden soll. Dieser Parameter kann nicht geändert werden, nachdem der Babelfish-DB-Cluster erstellt wurde. (Standardwert: *) (Zulässig: null)	–	false
babelfishpg_tds.port	Eine Ganzzahl, die den TCP-Port festlegt, der für Anfragen in der SQL-Server-Syntax verwendet wird. (Standardwert: 1433) (Zulässig: 1–65535)	statisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tds.tds_ssl_encrypt	Ein Boolescher Wert, der die Verschlüsselung für Daten, die den TDS-Listener-Port durchqueren, ein- (0) oder ausschaltet (1). Ausführliche Informationen zur Verwendung von SSL für Clientverbindungen finden Sie unter SSL-Einstellungen und Clientverbindungen für Babelfish. (Standardwert: 0) (Zulässig: 0, 1)	dynamisch	true
babelfishpg_tds.tds_ssl_max_protocol_version	Ein String, der die höchste SSL/TLS-Protokollversion angibt, die für die TDS-Sitzung verwendet werden soll. (Standardwert: 'TLSv1.2') (Zulässig: 'TLSv1', 'TLSv1.1', 'TLSv1.2')	dynamisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tds.tds_ssl_min_protocol_version	Ein String, der die niedrigste SSL/TLS-Protokollversion angibt, die für die TDS-Sitzung verwendet werden soll. (Standardwert: „TLSv1.2“ von Aurora-PostgreSQL-Version 16, „TLSv1“ für Versionen älter als Aurora-PostgreSQL-Version 16) (Zulässig: „TLSv1“, „TLSv1.1“, „TLSv1.2“)	dynamisch	true
babelfishpg_tds.unix_socket_directories	Ein String, der das Unix-Socket-Verzeichnis des TDS-Servers bezeichnet. Dieser Parameter kann nicht geändert werden, nachdem der Babelfish-DB-Cluster erstellt wurde. (Standardwert: /tmp) (Zulässig: null)	–	false

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tds.unix_socket_group	Ein String, der die Unix-Socket-Gruppe des TDS-Servers bezeichnet. Dieser Parameter kann nicht geändert werden, nachdem der Babelfish-DB-Cluster erstellt wurde. (Standardwert: rdsdb) (Zulässig: null)	–	false
babelfishpg_tsql.default_locale	Ein String, der das Standardgebietsschema für Babelfish-Sortierungen angibt. Das Standardgebietsschema ist nur das Gebietsschema und enthält keine Qualifikatoren. Stellen Sie diesen Parameter ein, wenn Sie einen Babelfish DB-Cluster bereitstellen. Nachdem der DB-Cluster bereitgestellt wurde, werden Änderungen an diesem Parameter ignoriert. (Standardwert: en_US) (Zulässig: Siehe - Tabellen)	statisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tsql.migration_mode	Eine nicht änderbare Liste, die die Unterstützung für Einzel- oder Mehrbenutzerdatenbanken angibt. Stellen Sie diesen Parameter ein, wenn Sie einen Babelfish DB-Cluster bereitstellen. Nach der Bereitstellung des DB-Clusters können Sie den Wert dieses Parameters nicht mehr ändern. (Standardwert: multi-db ab Aurora-PostgreSQL-Version 16, single-db für Versionen älter als Aurora-PostgreSQL-Version 16) (Zulässig: single-db, multi-db, null)	statisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
babelfishpg_tsql.server_colation_name	Ein String, der den Namen der Sortierung angibt, die für Aktionen auf Serverebene verwendet wird. Stellen Sie diesen Parameter ein, wenn Sie einen Babelfish DB-Cluster bereitstellen. Ändern Sie nach der Bereitstellung des DB-Clusters den Wert dieses Parameters nicht. (Standardwert: bbf_unicode_general_ci_as) (Zulässig: Siehe -Tabellen)	statisch	true
babelfishpg_tsql.version	Ein String, der die Ausgabe der Variablen @@VERSION festlegt. Ändern Sie diesen Wert nicht für Aurora-PostgreSQL-DB-Cluster. (Standardwert: null) (Zulässig: default)	dynamisch	true

Parameter	Beschreibung	Typ anwenden	Ist anpassbar
rds.babelfish_status	Ein String, der den Status der Babelfish-Funktionalität festlegt. Wenn dieser Parameter auf `datatypes only` gesetzt ist, ist Babelfish ausgeschaltet, aber SQL Server-Datentypen sind weiterhin verfügbar. (Standardwert: off) (Zulässig: on, off, datatypesonly)	statisch	true
unix_socket_permissions	Eine Ganzzahl, die die Unix-Socket-Berechtigungen des TDS-Servers festlegt. Dieser Parameter kann nicht geändert werden, nachdem der Babelfish-DB-Cluster erstellt wurde. (Standardwert: 0700) (Zulässig: 0–511)	–	false

SSL-Einstellungen und Clientverbindungen für Babelfish

Zum Erzwingen von SSL/TLS-Verbindungen zu Ihrem DB-Cluster von Babelfish für Aurora PostgreSQL verwenden Sie den Parameter `rds.force_ssl`.

- Zum Erzwingen von SSL/TLS-Verbindungen setzen Sie den Wert des Parameters `rds.force_ssl` auf 1 (on).
- Zum Deaktivieren erzwungener SSL/TLS-Verbindungen setzen Sie den Wert des Parameters `rds.force_ssl` auf 0 (off).

Der Standardwert dieses Parameters hängt von der Aurora-PostgreSQL-Version ab:

- Für Aurora-PostgreSQL-Versionen 17 und höher: Der Standardwert ist 1 (on).
- Für Aurora-PostgreSQL-Versionen 16 und niedriger: Der Standardwert ist 0 (off).

> **Note**
>
> Wenn Sie ein Hauptversions-Upgrade von Aurora-PostgreSQL-Version 16 oder früher auf Version 17 oder höher durchführen, ändert sich der Standardwert des Parameters von 0 (off) auf 1 (on). Diese Änderung kann zu Verbindungsfehlern bei Anwendungen führen, die nicht für SSL konfiguriert sind. Sie können zum vorherigen Standardverhalten zurückkehren, indem Sie diesen Parameter auf 0 (off) setzen.

Treiberspezifische Einzelheiten finden Sie unter Verbinden mit einem Babelfish-DB-Cluster.

Wenn ein Client eine Verbindung mit dem TDS-Port (Standardwert `1433`) herstellt, vergleicht Babelfish die Secure Sockets Layer (SSL)-Einstellung, die während des Client-Handshake gesendet wurde, mit der Babelfish-SSL-Parametereinstellung (`tds_ssl_encrypt`). Babelfish bestimmt dann, ob eine Verbindung zulässig ist. Wenn eine Verbindung zulässig ist, wird das Verschlüsselungsverhalten je nach Ihren Parametereinstellungen und der Unterstützung für die Verschlüsselung des Clients entweder erzwungen oder nicht.

Die folgende Tabelle zeigt, wie sich Babelfish für jede Kombination verhält.

Client SSL-Einstellungen	SSL-Einstellung für Babelfish	rds.force_ssl	Verbindung zulässig?	Wert wird an den Client zurückgegeben
ENCRYPT_ON	Any	Any	Zulässig, die gesamte Verbindung ist verschlüsselt	ENCRYPT_ON
ENCRYPT_OFF	tds_ssl_encrypt=1	Any	Zulässig, die gesamte Verbindung ist verschlüsselt	ENCRYPT_REQ
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=0	Zulässig, das Anmeldepaket ist verschlüsselt	ENCRYPT_OFF
ENCRYPT_OFF	tds_ssl_encrypt=0	rds.force_ssl=1	Nein, Verbindung geschlossen	ENCRYPT_OFF
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=0	Ja	ENCRYPT_NOT_SUP
ENCRYPT_NOT_SUP	tds_ssl_encrypt=1	Any	Nein, Verbindung geschlossen	ENCRYPT_REQ
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	rds.force_ssl=1	Nein, Verbindung geschlossen	ENCRYPT_NOT_SUP
ENCRYPT_CLIENT_CERT	Any	Any	Nein, Verbindung geschlossen	Nicht unterstützt

Verstehen der Kollationen in Babelfish für Aurora PostgreSQL

Wenn Sie einen Aurora-PostgreSQL-DB-Cluster mit Babelfish erstellen, wählen Sie eine Sortierung für Ihre Daten aus. Eine Sortierung gibt die Sortierreihenfolge und Bitmuster an, die den Text oder die Zeichen in einer bestimmten menschlichen Schriftsprache erzeugen. Eine Sortierung enthält Regeln zum Vergleich von Daten für einen bestimmten Satz von Bitmustern. Die Sortierung bezieht sich auf die Lokalisierung. Verschiedene Gebietsschemas wirken sich auf die Zeichenzuordnung, die Sortierreihenfolge und dergleichen aus. Sortierattribute spiegeln sich in den Namen verschiedener Sortierungen wider. Weitere Informationen zu den Attributen finden Sie unter Babelfish collation attributes table.

Babelfish ordnet SQL Server-Sortierungen vergleichbaren Sortierungen von Babelfish zu. Babelfish definiert Unicode-Sortierungen mit kulturell sensiblen Zeichenfolgenvergleiche und Sortierreihenfolge vorab. Babelfish bietet auch eine Möglichkeit, die Sortierungen in Ihrer SQL Server-DB in die am engsten übereinstimmende Babelfish-Sollation zu übersetzen. Lokalspezifische Sortierungen werden für verschiedene Sprachen und Regionen bereitgestellt.

Einige Sortierungen geben eine Codepage an, die einer clientseitigen Kodierung entspricht. Babelfish wird abhängig von der Sortierung jeder Ausgabespalte automatisch von der Servercodierung in die Clientcodierung übersetzt.

Babelfish unterstützt die Sortierungen, die in der Babelfish supported collations table aufgeführt sind. Babelfish ordnet SQL Server-Sortierungen vergleichbaren Sortierungen von Babelfish zu.

Babelfish verwendet Version 153.80 der ICU-Sortierbibliothek (International Components for Unicode). Weitere Informationen zu ICU-Sortierungen finden Sie unter Sortierung in der ICU-Dokumentation. Weitere Informationen zu PostgreSQL und Sortierungen finden Sie unter Unterstützung von Sortierungen in der PostgreSQL-Dokumentation.

Themen

- DB-Cluster-Parameter, die die Sortierung und das Gebietsschema steuern
- Deterministische und nicht deterministische Sortierungen in Babelfish
- Auf Datenbankebene unterstützte Sortierungen in Babelfish
- Server- und Objektsortierungen in Babelfish
- Standardsortierungsverhalten in Babelfish
- Verwalten von Sortierungen
- Einschränkungen und Verhaltensunterschiede von Sortierungen

DB-Cluster-Parameter, die die Sortierung und das Gebietsschema steuern

Die folgenden Parameter wirken sich auf das Sortierverhalten aus.

babelfishpg_tsql.default_locale

Dieser Parameter gibt das Standardgebietsschema an, das von der Sortierung verwendet wird. Dieser Parameter wird in Kombination mit den Attributen in der Babelfish collation attributes table verwendet, um Sortierungen für eine bestimmte Sprache und Region anzupassen. Der Standardwert für diesen Parameter ist `en-US`.

Das Standardgebietsschema gilt für alle Babelfish-Sortierungen, die mit den Buchstaben „BBF“ beginnen, und für alle SQL-Server-Sortierungen, die Babelfish-Sortierungen zugeordnet sind. Wenn die Einstellung für diesen Parameter auf einem vorhandenen Babelfish-DB-Cluster geändert wird, hat dies keinen Einfluss auf das Gebietsschema vorhandener Sortierungen. Eine Liste der Sortierungen finden Sie in der Babelfish supported collations table.

babelfishpg_tsql.server_colation_name

Dieser Parameter gibt die Standardsortierung für den Server (DB-Cluster-Instance von Aurora PostgreSQL) und die Datenbank an. Der Standardwert ist `sql_latin1_general_cp1_ci_as`. Die `server_collation_name`muss ein sein `CI_AS`-Sortierung, da in T-SQL die Serversortierung bestimmt, wie Bezeichner verglichen werden.

Wenn Sie Ihren Babelfish-DB-Cluster erstellen, wählen Sie den Collation name (Name der Sortierung) aus der auswählbaren Liste aus. Dazu gehören die Sortierungen in der Babelfish supported collations table. Ändern Sie nicht das `server_collation_name` nachdem die Babelfish-Datenbank erstellt wurde.

Die Einstellungen, die Sie beim Erstellen Ihres DB-Clusters von Babelfish for Aurora PostgreSQL auswählen, werden in der DB-Cluster-Parametergruppe gespeichert, die dem Cluster für diese Parameter zugeordnet ist, und legen dessen Sortierverhalten fest.

Deterministische und nicht deterministische Sortierungen in Babelfish

Babelfish unterstützt deterministische und nicht deterministische Kollationen:

- Eine deterministische Sortierung wertet Zeichen mit identischen Bytesequenzen als gleich aus. Dies bedeutet, dass `x` und `X` bei einer deterministischen Sortierung nicht gleich sind. Deterministische Sortierungen können Groß- und Kleinschreibung (CS) und Akzente (AS) berücksichtigen.

- Eine nicht deterministische Sortierung erfordert keine identische Übereinstimmung. Eine nicht deterministische Kollation bewertet x und X gleich. Nicht deterministische Sortierungen berücksichtigen Groß- und Kleinschreibung (CI), Akzente (AI) oder beides nicht.

In der folgenden Tabelle finden Sie einige Verhaltensunterschiede zwischen Babelfish und PostgreSQL im Hinblick auf die Verwendung deterministischer Sortierungen.

Babelfish	PostgreSQL
Unterstützt die LIKE-Klausel bei CI_AS-Sortierungen.	Unterstützt die LIKE-Klausel bei nicht deterministischen Sortierungen nicht.
Unterstützt die LIKE-Klausel nur bei den folgenden AI-Sortierungen ab Babelfish-Version 4.2.0: • bbf_unicode_cp1250_ci_ai • bbf_unicode_cp1250_cs_ai • bbf_unicode_cp1257_ci_ai • bbf_unicode_cp1257_cs_ai • bbf_unicode_cp1_ci_ai • bbf_unicode_cp1_cs_ai • estonian_ci_ai • finnish_swedish_ci_ai • french_ci_ai • latin1_general_ci_ai • latin1_general_cs_ai • modern_spanish_ci_ai • polish_ci_ai • sql_latin1_general_cp1_ci_ai • sql_latin1_general_cp1_cs_ai • traditional_spanish_ci_ai	Unterstützt die LIKE-Klausel bei nicht deterministischen Sortierungen nicht.

Eine Liste anderer Einschränkungen und Verhaltensunterschiede für Babelfish im Vergleich zu SQL Server und PostgreSQL finden Sie unter Einschränkungen und Verhaltensunterschiede von Sortierungen.

Babelfish und SQL Server folgen einer Benennungskonvention für Sortierungen, die die Sortierattribute beschreiben, wie in der folgenden Tabelle dargestellt.

Attribut	Beschreibung
AI	Akzentunempfindlich.
AS	Akzentsensibel.
BIN2	BIN2 fordert die Sortierung von Daten in Codepunkt-Reihenfolge an. Die Reihenfolge der Unicode-Codepunkte entspricht der gleichen Zeichenre ihenfolge für UTF-8-, UTF-16- und UCS-2-Kodierungen. Die Codepunkt-Reihenfolge ist eine schnelle deterministische Sortierung.
CI	Berücksichtigt Groß- und Kleinschreibung nicht.
CS	Groß-/Kleinschreibung ist zu beachten.
PREF	Verwenden Sie eine PREF-Sortierung, um Großbuchstaben vor Kleinbuch staben zu sortieren. Wenn beim Vergleich die Groß- und Kleinschreibung nicht berücksichtigt wird, wird die Großbuchstabenversion vor der Kleinbuch stabenversion sortiert, wenn keine andere Unterscheidung vorliegt. Die ICU-Bibliothek unterstützt Großbuchstaben mit `colCaseFirst=upper` , aber nicht für CI_AS-Sortierungen. PREF kann nur auf `CS_AS` deterministische Kollationen angewendet werden.

Auf Datenbankebene unterstützte Sortierungen in Babelfish

Die folgenden Sortierungen werden in Babelfish auf Datenbankebene unterstützt:

- bbf_unicode_bin2
- bbf_unicode_cp1_ci_ai
- bbf_unicode_cp1_ci_as

- bbf_unicode_cp1250_ci_ai
- bbf_unicode_cp1250_ci_as
- bbf_unicode_cp1257_ci_ai
- bbf_unicode_cp1257_ci_as
- estonian_ci_ai
- estonian_ci_as
- finnish_swedish_ci_ai
- finnish_swedish_ci_as
- french_ci_ai
- french_ci_as
- latin1_general_bin2
- latin1_general_ci_ai
- latin1_general_ci_as
- latin1_general_90_bin2
- latin1_general_100_bin2
- latin1_general_140_bin2
- modern_spanish_ci_ai
- modern_spanish_ci_as
- polish_ci_ai
- polish_ci_as
- sql_latin1_general_cp1_ci_ai
- sql_latin1_general_cp1_ci_as
- sql_latin1_general_cp1250_ci_as
- sql_latin1_general_cp1251_ci_as
- sql_latin1_general_cp1257_ci_as
- traditional_spanish_ci_ai
- traditional_spanish_ci_as

> **Note**
>
> Stellen Sie zur Verwendung einer anderen Sortierung auf Datenbankebene sicher, dass sie mit der Sortierung auf Serverebene übereinstimmt. Weitere Informationen finden Sie unter Server- und Objektsortierungen in Babelfish

Server- und Objektsortierungen in Babelfish

Verwenden Sie die folgenden Sortierungen als Serversortierung oder Objektsortierung.

Sortier-ID	Hinweise
bbf_unicode_general_ci_as	Unterstützt den Vergleich zwischen Groß- und Kleinschreibung und den LIKE Operator.
bbf_unicode_cp1_ci_as	Nichtdeterministische Kollation auch bekannt als CP1252.
bbf_unicode_CP1250_ci_as	Nichtdeterministische Kollation wird verwendet, um Texte in mitteleuropäischen und osteuropäischen Sprachen darzustellen, die lateinische Schrift verwenden.
bbf_unicode_CP1251_ci_as	Nichtdeterministische Kollation für Sprachen, die das kyrillische Skript verwenden.
bbf_unicode_cp1253_ci_as	Nichtdeterministische Kollation repräsentierte früher das moderne Griechisch.
bbf_unicode_cp1254_ci_as	Nichtdeterministische Kollation, die Türkisch unterstützt.
bbf_unicode_cp1255_ci_as	Nichtdeterministische Kollation, die Hebräisch unterstützt.
bbf_unicode_cp1256_ci_as	Nichtdeterministische Kollation wird verwendet, um Sprachen zu schreiben, die arabische Schrift verwenden.

Sortier-ID	Hinweise
bbf_unicode_cp1257_ci_as	Nichtdeterministische Kollationwird verwendet, um estnische, lettische und litauische Sprachen zu unterstützen.
bbf_unicode_cp1258_ci_as	Nichtdeterministische Kollation wird verwendet, um vietnamesische Schriftzeichen zu schreiben.
bbf_unicode_cp874_ci_as	Nichtdeterministische Kollation wrid verwendet, um thailändische Charaktere zu schreiben.
sql_latin1_general_cp1250_ci_as	Nicht deterministische Einzelbyte-Zeichencodierung wird verwendet, um lateinische Zeichen darzustellen.
sql_latin1_general_cp1251_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.
sql_latin1_general_cp1_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.
sql_latin1_general_cp1253_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.
sql_latin1_general_cp1254_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.
sql_latin1_general_cp1255_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.
sql_latin1_general_cp1256_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.

Sortier-ID	Hinweise
sql_latin1_general_cp1257_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.
sql_latin1_general_cp1258_ci_as	Nichtdeterministische Kollation, die lateinische Zeichen unterstützt.
chinese_prc_ci_as	Nicht deterministische Sortierung, die Chinesisch (VR China) unterstützt.
cyrillic_general_ci_as	Nicht deterministische Kollation, die Kyrillisch unterstützt.
finnish_swedish_ci_as	Nicht deterministische Kollatierung, die Finnisch unterstützt.
french_ci_as	Nicht deterministische Kollatierung, die Französisch unterstützt.
japanese_ci_as	Nicht deterministische Sortierung, die Japanisch unterstützt. Wird in Babelfish 2.1.0 und höheren Versionen unterstützt.
korean_wansung_ci_as	Nicht deterministische Sortierung, die Koreanisch unterstützt (mit Wörterbuchsortierung).
latin1_general_ci_as	Nicht deterministische Sortierung, die lateinische Zeichen unterstützt.
modern_spanish_ci_as	Nicht deterministische Kollatierung, die das moderne Spanisch unterstützt.
polish_ci_as	Nicht deterministische Kollatierung, die Polnisch unterstützt.
thai_ci_as	Nicht deterministische Kollatierung, die Thai unterstützt.

Sortier-ID	Hinweise
traditional_spanish_ci_as	Nicht deterministische Sortierung, die Spanisch unterstützt (traditionelle Sortierung).
turkish_ci_as	Nicht deterministische Kollatierung, die Türkisch unterstützt.
ukrainisch_ci_as	Nicht deterministische Kollatierung, die Ukrainisch unterstützt.
vietnamesisch_ci_as	Nicht deterministische Kollatierung, die Vietnamesen unterstützt.

Sie können die folgenden Sortierungen als Objektsortierungen verwenden.

Dialekt	Deterministische Optionen	Nicht deterministische Optionen
Arabisch	Arabic_CS_AS	Arabic_CI_AS Arabic_CI_AI
Arabische Schrift	BBF_Unicode_CP1256_CS_AS BBF_Unicode_Pref_CP1256_CS_AS	BBF_Unicode_CP1256_CI_AI BBF_Unicode_CP1256_CS_AI
Binär	latin1_general_bin2 BBF_Unicode_BIN2	–
Mitteleur opäische und osteuropäische Sprachen, die lateinisc	BBF_Unicode_CP1250_CS_AS BBF_Unicode_Pref_CP1250_CS_AS	BBF_Unicode_CP1250_CI_AI BBF_Unicode_CP1250_CS_AI

Dialekt	Deterministische Optionen	Nicht deterministische Optionen
he Schrift verwenden		
Chinesisch	Chinese_PRC_CS_AS	Chinese_PRC_CI_AS Chinese_PRC_CI_AI
Kyrillic_ Allgemein	Cyrillic_General_CS_AS	Cyrillic_General_CI_AS Cyrillic_General_CI_AI
Kyrillische Schrift	BBF_Unicode_CP1251_CS_AS BBF_Unicode_Pref_CP1251_CS_AS	BBF_Unicode_CP1251_CI_AI BBF_Unicode_CP1251_CS_AI
Estnisch	Estonian_CS_AS	Estonian_CI_AS Estonian_CI_AI
Estnisch, Lettisch und Litauisch	BBF_Unicode_CP1257_CS_AS BBF_Unicode_Pref_CP1257_CS_AS	BBF_Unicode_CP1257_CI_AI BBF_Unicode_CP1257_CS_AI
Finnish_S chwedisch	Finnish_Swedish_CS_AS	Finnish_Swedish_CI_AS Finnish_Swedish_CI_AI
Französisch	französisch_cs_as	French_CI_AS French_CI_AI
Griechisch	Greek_CS_AS	Greek_CI_AS Greek_CI_AI

Dialekt	Deterministische Optionen	Nicht deterministische Optionen
Hebräisch	BBF_Unicode_CP1255_CS_AS BBF_Unicode_Pref_CP1255_CS_AS Hebrew_CS_AS	BBF_Unicode_CP1255_CI_AI BBF_Unicode_CP1255_CS_AI Hebrew_CI_AS Hebrew_CI_AI
Japanisch (Babelfish 2.1.0 und höher)	Japanese_CS_AS	Japanese_CI_AI Japanese_CI_AS
Korean_Wamsung	Korean_wamsung_cs_as	Korean_Wamsung_CI_AS Korean_Wamsung_CI_AI
Lateinische Zeichen für die Codeseite CP1252	latin1_general_cs_as BBF_Unicode_General_CS_AS BBF_Unicode_General_Pref_CS_AS BBF_Unicode_Pref_CP1_CS_AS BBF_Unicode_CP1_CS_AS	latin1_general_ci_as latin1_general_ci_ai latin1_general_cs_ai BBF_Unicode_General_CI_AI BBF_Unicode_General_CS_AI BBF_Unicode_CP1_CI_AI BBF_Unicode_CP1_CS_AI
Neugriechisch	BBF_Unicode_CP1253_CS_AS BBF_Unicode_Pref_CP1253_CS_AS	BBF_Unicode_CP1253_CI_AI BBF_Unicode_CP1253_CS_AI

Dialekt	Deterministische Optionen	Nicht deterministische Optionen
Modern_Spanisch	Modern_Spanish_CS_AS	Modern_Spanish_CI_AS Modern_Spanish_CI_AI
Mongolisch	Mongolisch_CS_AS	Mongolian_CI_AS Mongolian_CI_AI
Polnisch	Polish_CS_AS	Polish_CI_AS Polish_CI_AI
Thailändisch	BBF_Unicode_CP874_CS_AS BBF_Unicode_Pref_CP874_CS_AS Thai_cs_as	BBF_Unicode_CP874_CI_AI BBF_Unicode_CP874_CS_AI Thai_CI_AS, Thai_CI_AI
Traditional_Spanisch	Traditional_Spanish_CS_AS	Traditional_Spanish_CI_AS Traditional_Spanish_CI_AI
Türkisch	BBF_Unicode_CP1254_CS_AS BBF_Unicode_Pref_CP1254_CS_AS Türkisch_CS_AS	BBF_Unicode_CP1254_CI_AI BBF_Unicode_CP1254_CS_AI Turkish_ci_as, Türkisch_CI_AI
Ukrainisch	Ukranian_CS_AS	Ukranian_CI_AS Ukranian_CI_AI

Dialekt	Deterministische Optionen	Nicht deterministische Optionen
Vietnamesisch	BBF_Unicode_CP1258_CS_AS	BBF_Unicode_CP1258_CI_AI
	BBF_Unicode_Pref_CP1258_CS_AS	BBF_Unicode_CP1258_CS_AI
	Vietnamese_CS_AS	Vietnamese_CI_AS
		Vietnamese_CI_AI

Standardsortierungsverhalten in Babelfish

Bisher lautete die Standardsortierung der sortierbaren Datentypen `pg_catalog.default`. Die Datentypen und Objekte, die von diesen Datentypen abhängen, werden unter Berücksichtigung der Groß- und Kleinschreibung sortiert. Diese Bedingung wirkt sich möglicherweise auf die T-SQL-Objekte des Datensatzes aus, der ohne Berücksichtigung der Groß- und Kleinschreibung sortiert wird. Ab Babelfish 2.3.0 entspricht die Standardsortierung für die sortierbaren Datentypen (außer TEXT und NTEXT) der Sortierung im Parameter `babelfishpg_tsql.server_collation_name`. Wenn Sie ein Upgrade auf Babelfish 2.3.0 durchführen, wird die Standardsortierung bei der Erstellung des DB-Clusters automatisch ausgewählt, was keine sichtbaren Auswirkungen hat.

Verwalten von Sortierungen

Die ICU-Bibliothek bietet Sortierversionsverfolgung, um sicherzustellen, dass Indizes, die von Sortierungen abhängen, neu dexiert werden können, wenn eine neue Version der ICU verfügbar wird. Um festzustellen, ob Ihre aktuelle Datenbank Sortierungen enthält, die aktualisiert werden müssen, können Sie die folgende Abfrage verwenden, nachdem Sie eine Verbindung mit `psql` oder `pgAdmin` hergestellt haben:

```
SELECT pg_describe_object(refclassid, refobjid,
    refobjsubid) AS "Collation",
    pg_describe_object(classid, objid, objsubid) AS "Object"
    FROM pg_depend d JOIN pg_collation c ON refclassid = 'pg_collation'::regclass
    AND refobjid = c.oid WHERE c.collversion <> pg_collation_actual_version(c.oid)
    ORDER BY 1, 2;
```

Diese Abfrage gibt beispielsweise die folgende Ausgabe zurück:

```
 Collation | Object
```

```
-----------+--------
(0 rows)
```

In diesem Beispiel müssen keine Sortierungen aktualisiert werden.

Zum Abrufen einer Auflistung der vordefinierten Sortierungen in Ihrer Babelfish-Datenbank können Siepsql oder pgAdmin mit der folgenden Abfrage verwenden:

```
SELECT * FROM pg_collation;
```

Vordefinierte Sortierungen werden in der sys.fn_helpcollations-Tabelle gespeichert. Mit dem folgenden Befehl können Sie Informationen über eine Sortierung anzeigen (z. B. die Flags „lcid“, „style“ und „collatate“). Wenn Sie eine Auflistung aller Sortierungen mit sqlcmd abrufen möchten, verbinden Sie sich mit dem T-SQL-Port (standardmäßig 1433) und führen Sie die folgende Abfrage aus:

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                  description
------------------------------------- ----------------------------------------
arabic_cs_as                          Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                          Arabic, case-insensitive, accent-insensi
arabic_ci_as                          Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                      Unicode-General, case-sensitive, accent-
bbf_unicode_cp1250_ci_ai              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_cs_ai              Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as              Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as         Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai              Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as              Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai              Default locale, code page 1254, case-ins
...
(124 rows affected)
```

Die im Beispiel gezeigten Zeilen 1 und 2 schränken die Ausgabe nur zu Lesbarkeitszwecken der Dokumentation ein.

```
1> SELECT SERVERPROPERTY('COLLATION')
```

```
2> GO
serverproperty
--------------------------------------------------------------------------------
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>
```

Einschränkungen und Verhaltensunterschiede von Sortierungen

Babelfish verwendet die ICU-Bibliothek zur Sortierunterstützung. PostgreSQL ist mit einer bestimmten Version der ICU erstellt und kann höchstens mit einer Version einer Sortierung übereinstimmen. Variationen zwischen Versionen sind unvermeidbar, ebenso geringfügige Schwankungen im Laufe der Zeit, während sich die Sprachen entwickeln. Im folgenden Abschnitt werden einige der bekannten Einschränkungen und Verhaltensvarianten von Babelfish-Sortierungen aufgeführt:

- Indizes und Sortierungstypabhängigkeit – Ein Index für einen benutzerdefinierten Typ, der von der International Components for Unicode (ICU)-Sortierungsbibliothek (die von Babelfish verwendete Bibliothek) abhängt, wird nicht ungültig, wenn sich die Version der Bibliothek ändert.
- COLLATIONPROPERTY-Funktion – Sortiereigenschaften werden nur für die unterstützten Babelfish-BBF-Sortierungen implementiert. Weitere Informationen hierzu finden Sie unter Babelfish supported collations table.
- Unterschiede bei Unicode-Sortierregeln – SQL-Sortierungen für SQL Server sortieren mit Unicode codierte Daten (`nchar` und `nvarchar`) anders als Daten, die nicht mit Unicode codiert sind (`char` und `varchar`). Babelfish-Datenbanken sind immer UTF-8-codiert und wenden Unicode-Sortierregeln unabhängig vom Datentyp immer einheitlich an, sodass die Sortierreihenfolge für `char` oder `varchar` die gleiche ist wie für `nchar` oder `nvarchar`.
- Sekundär gleiche Sortierungen und Sortierverhalten – Die Standard-ICU-Unicode-Sortierung sekundär gleich (`CI_AS`) sortiert Satzzeichen und andere nicht alphanumerische Zeichen vor numerischen Zeichen und numerische Zeichen vor alphabetischen Zeichen. Die Reihenfolge der Satzzeichen und anderer Sonderzeichen ist jedoch unterschiedlich.
- Teritiäre Sortierung, Umgehung für ORDER BY – SQL-Sortierungen wie `SQL_Latin1_General_Pref_CP1_CI_AS` unterstützen die `TERTIARY_WEIGHTS`-Funktion und die Möglichkeit, Strings zu sortieren, die gleichermaßen in einer `CI_AS`-Sortierung verglichen werden, die zuerst in Großbuchstaben sortiert werden soll: `ABC`, `ABc`, `AbC`, `Abc`, `aBC`, `aBc`, `abC` und schließlich `abc`. Daher bewertet die analytische Funktion `DENSE_RANK OVER (ORDER`

`BY column`) diese Strings als denselben Rang, ordnet sie jedoch zuerst in einer Partition in Großbuchstaben an.

Sie können ein ähnliches Ergebnis mit Babelfish erzielen, indem Sie eine `COLLATE`-Klausel zu einer `ORDER BY`-Klausel hinzufügen, die eine Tertiäre `CS_AS` angibt,, welche `@colCaseFirst=upper` spezifiziert. Der `colCaseFirst`-Modifizierer gilt jedoch nur für Strings, die tertiär gleich sind (anstatt sekundär gleich wie bei einer `CI_AS`-Sortierung). Daher können Sie tertiäre SQL-Sollationen nicht mit einer einzigen ICU-Sortierung emulieren.

Als Problemumgehung empfehlen wir, Anwendungen zu ändern, die eine `SQL_Latin1_General_Pref_CP1_CI_AS`-Kollation zur Verwendung der `BBF_SQL_Latin1_General_CP1_CI_AS`-Kollation nutzt. Dann fügen Sie `COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS` zu einer beliebigen `ORDER BY`-Klausel für diese Spalte hinzu.

- Zeichenerweiterung – Eine Zeichenerweiterung behandelt ein einzelnes Zeichen als eine Zeichenfolge auf der Primärebene. Die `CI_AS`-Standardsortierung von SQL Server unterstützt die Zeichenerweiterung. ICU-Sortierungen unterstützen die Zeichenerweiterung nur für akzentunempfindliche Sortierungen.

  Wenn eine Zeichenerweiterung erforderlich ist, verwenden Sie eine `AI`-Sortierung für Vergleiche. Solche Sortierungen werden derzeit jedoch vom LIKE Operator nicht unterstützt.

- Codierung von char und varchar – Wenn SQL-Sortierungen für `char`- oder `varchar`-Datentypen verwendet werden, wird die Sortierreihenfolge für Zeichen vor ASCII 127 durch die spezifische Code Page für diese SQL-Sortierung bestimmt. Bei SQL-Sortierungen können Strings, die als `char` oder `varchar` deklariert sind, anders sortiert werden als Strings, die als `nchar` oder `nvarchar` deklariert sind.

  PostgreSQL kodiert alle Strings mit der Datenbankkodierung, damit alle Zeichen in UTF-8 konvertiert und nach Unicode-Regeln sortiert werden.

  Da SQL-Sollationen nchar- und nvarchar-Datentypen mithilfe von Unicode-Regeln sortieren, codiert Babelfish alle Strings auf dem Server mit UTF-8. Babelfish sortiert nchar- und nvarchar-Strings genauso wie Char- und Varchar-Strings unter Verwendung von Unicode-Regeln.

- Ergänzende Zeichen – Die SQL-Server-Funktionen `NCHAR`, `UNICODE` und `LEN` unterstützen Zeichen für Codepunkte außerhalb der Unicode Basic Multilingual Plane (BMP). Im Gegensatz dazu verwenden Nicht-SC-Sollationen Ersatzpaarzeichen, um zusätzliche Zeichen zu behandeln.

Für Unicode-Datentypen kann SQL Server bis zu 65.535 Zeichen mit UCS-2 oder dem gesamten Unicode-Bereich (1.114.114 Zeichen) darstellen, wenn zusätzliche Zeichen verwendet werden.

- Kana-sensible (KS) Sortierungen – Eine kana-sensible (KS) Sortierung behandelt japanische `Hiragana`- und `Katakana`-Kana-Zeichen anders. Die Intensivstation unterstützt den japanischen Sortierstandard `JIS X 4061`. Der jetzt veraltete `colhiraganaQ [on | off]` lokale Modifikator bietet möglicherweise die gleiche Funktionalität wie KS-Sortierungen. KS-Sortierungen mit demselben Namen wie SQL Server werden derzeit jedoch nicht von Babelfish unterstützt.
- Breitenempfindliche (WS) Sortierungen – Wenn ein Einzelbyte-Zeichen (halbe Breite) und das gleiche Zeichen, das als Doppelbyte-Zeichen (volle Breite) dargestellt wird, unterschiedlich behandelt werden, wird die Sortierung als breitenempfindlich (width-sensitive, WS) bezeichnet. WS-Sortierungen mit dem gleichen Namen wie SQL Server werden derzeit von Babelfish nicht unterstützt.
- Variationsauswahl-sensible (VSS) Sortierungen – Variationsauswahl-sensible (VSS) Sortierungen unterscheiden zwischen ideographischen Variationsauswahlen in japanischen Sortierungen `Japanese_Bushu_Kakusu_140` und `Japanese_XJIS_140`. Eine Variationssequenz besteht aus einem Basiszeichen plus einem zusätzlichen Variationsselektor. Wenn Sie nicht die `_VSS`-Option wählen, wird der Variationsselektor im Vergleich nicht berücksichtigt.

  VSS-Sortierungen werden derzeit nicht von Babelfish unterstützt.
- BIN- und BIN2-Sortierungen – Eine BIN2-Sortierung sortiert Zeichen nach Codepunkt-Reihenfolge. Die binäre Byte-für-Byte-Reihenfolge von UTF-8 behält die Reihenfolge der Unicode-Codepunkte bei, daher ist dies wahrscheinlich auch die Sortierung mit der besten Leistung. Wenn die Unicode-Codepunkt-Reihenfolge für eine Anwendung funktioniert, sollten Sie eine BIN2-Sortierung verwenden. Die Verwendung einer BIN2-Sollation kann jedoch dazu führen, dass Daten auf dem Client in einer kulturell unerwarteten Reihenfolge angezeigt werden. Im Laufe der Zeit werden Unicode neue Zuordnungen zu Kleinbuchstaben hinzugefügt, sodass die `LOWER`-Funktion auf verschiedenen Versionen der Intensivstation unterschiedlich funktionieren kann. Dies ist ein Sonderfall des allgemeineren Sortierungsproblems und nicht als etwas, das für die BIN2-Sollation spezifisch ist.

  Babelfish bietet eine `BBF_Latin1_General_BIN2`-Kollation mit der Babelfish-Distribution, um in Unicode-Codepunkt-Reihenfolge zusammenzufassen. In einer BIN-Sollation wird nur das erste Zeichen als wchar sortiert. Die verbleibenden Zeichen werden byte für Byte sortiert, effektiv in Codepunkt-Reihenfolge entsprechend ihrer Kodierung. Dieser Ansatz folgt nicht den Unicode-Sollationsregeln und wird von Babelfish nicht unterstützt.

- Nicht deterministische Sortierungen und CHARINDEX-Einschränkung – Für Babelfish-Versionen, die älter als Version 2.1.0 sind, können Sie CHARINDEX nicht mit nicht deterministischen Sortierungen verwenden. Standardmäßig verwendet Babelfish eine Sortierung ohne Berücksichtigung der Groß- und Kleinschreibung (nicht deterministisch). Die Verwendung von CHARINDEX für ältere Versionen von Babelfish löst den folgenden Laufzeitfehler aus:

```
nondeterministic collations are not supported for substring searches
```

> **Note**
>
> Diese Einschränkung und Problemumgehung gelten nur für Babelfish Version 1.x (13.x-Versionen von Aurora PostgreSQL). Babelfish 2.1.0 und höhere Releases haben dieses Problem nicht.

Dieses Problem können Sie mit einer der folgenden Methoden umgehen:

- Konvertieren Sie den Ausdruck explizit in eine Kollatierung mit Berücksichtigung der Groß- und Kleinschreibung und wandeln Sie die Groß- oder Kleinschreibung beider Argumente um, indem Sie LOWER oder UPPER anwenden. Der Aufruf `SELECT charindex('x', a) FROM t1` führt beispielsweise zu folgender Ausgabe:

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- Erstellen Sie eine SQL-Funktion f_charindex und ersetzen Sie CHARINDEX-Aufrufe durch Aufrufe der folgenden Funktion:

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
WHILE len(@s2) >= len(@s1)
BEGIN
  if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
  set @i += 1
  set @s2 = substring(@s2,2,999999999)
END
return 0
END
```

```
go
```

Verwalten der Babelfish-Fehlerbehandlung mit Escape-Schraffuren

Babelfish ahmt das SQL-Verhalten in Bezug auf den Kontrollfluss und den Transaktionsstatus nach Möglichkeit nach. Wenn Babelfish auf einen Fehler stößt, gibt es nach Möglichkeit einen Fehlercode zurück, der dem SQL Server-Fehlercode ähnelt. Wenn Babelfish den Fehler keinem SQL-Server-Code zuordnen kann, gibt es einen festen Fehlercode (`33557097`) zurück und führt spezifische Aktionen basierend auf der Art des Fehlers wie folgt aus:

- Bei Kompilierzeitfehlern führt Babelfish ein Rollback für die Transaktion aus.
- Wenn es sich um einen Laufzeitfehler handelt, beendet Babelfish den Stapel und führt ein Rollback für die Transaktion aus.
- Bei einem Protokollfehler zwischen Client und Server wird kein Rollback der Transaktion ausgeführt.

Wenn ein Fehlercode nicht einem äquivalenten Code zugeordnet werden kann und der Code für einen ähnlichen Fehler verfügbar ist, wird der Fehlercode dem alternativen Code zugeordnet. Zum Beispiel die Verhaltensweisen, die SQL Server-Codes `8143` und `8144` verursachen, sind beide als `8143` mappiert.

Fehler, die nicht mappiert werden können, respektieren ein `TRY... CATCH`-Konstrukt nicht.

Sie können `@@ERROR` nutzen, um einen SQL Server-Fehlercode zurückzugeben, oder die `@@PGERROR`-Funktion, um einen PostgreSQL-Fehlercode zurückzugeben. Sie können auch die `fn_mapped_system_error_list`-Funktion nutzen, um eine Liste der zugeordneten Fehlercodes zurückzugeben. Weitere Informationen zu PostgreSQL-Fehlercodes finden Sie unter Die PostgreSQL-Website.

Ändern der Escape-Schraffureinstellungen in Babelfish

Um besser mit Anweisungen umzugehen, die scheitern könnten, definiert Babelfish bestimmte Optionen, die als Escape-Schraffuren bezeichnet werden. Eine Escape-Luke ist eine Option, die das Verhalten von Babelfish angibt, wenn es auf eine nicht unterstütztes Funktion oder eine nicht unterstützte Syntax stößt.

Sie können die `sp_babelfish_configure` gespeicherte Prozedur zur Steuerung der Einstellungen einer Escape-Schraffur nutzen. Verwenden Sie das Skript, um die Escape-Schraffur auf `ignore` oder `strict` zu setzen. Wenn es auf `strict` eingestellt ist, gibt Babelfish einen Fehler zurück, den Sie korrigieren müssen, bevor Sie fortfahren.

Schließen Sie das `server`-Schlüsselwort ein, um die Änderungen auf die aktuelle Sitzung und auf Cluster-Ebene anzuwenden.

Die Verwendung ist wie folgt:

- Um alle Escape-Schraffuren und ihren Status sowie Nutzungsinformationen aufzulisten, führen Sie `sp_babelfish_configure` aus.
- Führen Sie den Befehl `sp_babelfish_configure '`*`hatch_name`*`'` aus, um die benannten Escape-Schraffuren und ihre Werte für die aktuelle oder clusterweite Sitzung aufzulisten, wo *`hatch_name`* die Kennung einer oder mehrerer Escape-Schraffuren *`hatch_name`* SQL-Platzhalter wie '%' verwenden kann.
- Um eine oder mehrere Escape-Schraffuren auf den angegebenen Wert festzulegen, führen Sie `sp_babelfish_configure ['`*`hatch_name`*`' [, 'strict'|'ignore' [, 'server']]` aus. Um die Einstellungen auf einer clusterweiten Ebene dauerhaft zu machen, schließen Sie das `server`-Schlüsselwort ein, wie im Folgenden dargestellt:

  ```
  EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
  ```

  Um sie nur für die aktuelle Sitzung festzulegen, verwenden Sie `server` nicht.
- Führen Sie `sp_babelfish_configure 'default'` (Babelfish 1.2.0 und höher) aus, um alle Escape-Schraffuren auf ihre Standardwerte zurückzusetzen.

Die Zeichenfolge, die die Schraffur (oder Schraffuren) identifiziert, kann SQL-Platzhalter enthalten. Im Folgenden werden beispielsweise alle Syntax-Escape-Schraffuren festgelegt auf `ignore` für den Aurora-PostgreSQL-Cluster.

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

In der folgenden Tabelle finden Sie Beschreibungen und Standardwerte für die vordefinierten Escape-Schraffuren von Babelfish.

Escape-Luke	Beschreibung	Standard
`escape_hatch_checkpoint`	Erlaubt die Verwendung der CHECKPOINT-Anweisung im prozeduralen Code, die	ignore

Escape-Luke	Beschreibung	Standard
	CHECKPOINT-Anweisung ist derzeit jedoch nicht implementiert.	
`escape_hatch_constraint_name_for_default`	Steuert das Verhalten von Babelfish im Zusammenhang mit Standardbeschränkungsnamen.	ignore
`escape_hatch_database_misc_options`	Steuert das Verhalten von Babelfish im Zusammenhang mit den folgenden Optionen in CREATE DATABASE: CONTAINMENT, DB_CHAINING, TRUSTWORTHY, PERSISTENT_LOG_BUFFER	ignore
`escape_hatch_for_replication`	Steuert das Verhalten von Babelfish im Zusammenhang mit der [NOT] FOR REPLICATION-Klausel beim Erstellen oder Ändern einer Tabelle.	strict
`escape_hatch_fulltext`	Steuert das Verhalten von Babelfish im Zusammenhang mit FULLTEXT-Features wie DEFAULT_FULLTEXT_LANGUAGE in CREATE/ALTER DATABASE, CREATE FULLTEXT INDEX oder sp_fulltext_database.	ignore

Escape-Luke	Beschreibung	Standard
`escape_hatch_ignore_dup_key`	Steuert das Verhalten von Babelfish im Zusammenhang mit CREATE/ALTER TABLE und CREATE INDEX. Wenn IGNORE_DUP_KEY=ON gilt, wird ein Fehler ausgelöst, wenn die Einstellung `strict` (Standard wert) festgelegt ist, oder der Fehler ignoriert, wenn die Einstellung `ignore` lautet (Babelfish Version 1.2.0 und höher).	strict
`escape_hatch_index_clustering`	Steuert das Verhalten von Babelfish im Zusammenh ang mit den Schlüsselwörtern CLUSTERED oder NONCLUSTE RED für Indizes und PRIMARY KEY- oder UNIQUE-Einschränku ngen. Wenn CLUSTERED ignoriert wird, wird der Index oder die Einschränkung immer noch so erstellt, als wäre NONCLUSTE RED angegeben worden.	ignore
`escape_hatch_index_columnstore`	Steuert das Verhalten von Babelfish im Zusammenhang mit der COLUMNSTORE-Klause l. Wenn Sie `ignore` angeben, erstellt Babelfish einen regulären B-Baum-Index.	strict

Escape-Luke	Beschreibung	Standard
`escape_hatch_join_hints`	Steuert das Verhalten von Schlüsselwörtern in einem JOIN-Operator: LOOP, HASH, MERGE, REMOTE, REDUCATE, REDISTRIBUTE, REPLICATE.	ignore
`escape_hatch_language_non_english`	Steuert das Verhalten von Babelfish im Zusammenhang mit anderen Sprachen als Englisch für Nachrichten auf dem Bildschirm. Babelfish unterstützt derzeit nur `us_english` für Nachrichten auf dem Bildschirm. SET LANGUAGE verwendet möglicherweise eine Variable, die den Sprachnamen enthält, sodass die tatsächlich festgelegte Sprache nur zur Laufzeit erkannt werden kann.	strict
`escape_hatch_login_hashed_password`	Wenn es ignoriert wird, unterdrückt der Fehler für das Schlüsselwort `HASHED` für `CREATE LOGIN` und `ALTER LOGIN`.	strict
`escape_hatch_login_misc_options`	Wenn es ignoriert wird, unterdrückt der Fehler bei anderen Schlüsselwörtern außer `HASHED`, `´MUST_CHANGE` , `OLD_PASSWORD` und `UNLOCK` für `CREATE LOGIN` und `ALTER LOGIN`.	strict

Escape-Luke	Beschreibung	Standard
`escape_hatch_login_old_password`	Wenn es ignoriert wird, unterdrückt der Fehler für das Schlüsselwort `OLD_PASSWORD` für `CREATE LOGIN` und `ALTER LOGIN`.	strict
`escape_hatch_login_password_must_change`	Wenn es ignoriert wird, unterdrückt der Fehler für das Schlüsselwort `MUST_CHANGE` für `CREATE LOGIN` und `ALTER LOGIN`.	strict
`escape_hatch_login_password_unlock`	Wenn es ignoriert wird, unterdrückt der Fehler für das Schlüsselwort `UNLOCK` für `CREATE LOGIN` und `ALTER LOGIN`.	strict
`escape_hatch_nocheck_add_constraint`	Steuert das Verhalten von Babelfish im Zusammenhang mit der WITH CHECK- oder NOCHECK-Klausel für Einschränkungen.	strict
`escape_hatch_nocheck_existing_constraint`	Steuert das Verhalten von Babelfish im Zusammenhang mit FOREIGN KEY- oder CHECK-Einschränkungen.	strict

Escape-Luke	Beschreibung	Standard
`escape_hatch_query_hints`	Steuert das Verhalten von Babelfish im Zusammenhang mit Abfragehinweisen. Wenn diese Option auf ignorieren eingestellt ist, ignoriert der Server Hinweise, die die OPTION (...) -Klausel verwenden, um Aspekte der Abfrageverarbeitung anzugeben. Beispiele sind SELECT FROM... OPTION (JOIN-HASH ZUSAMMENFÜHREN, MAXRECURSION 10)).	ignore
`escape_hatch_rowversion`	Steuert das Verhalten der Datentypen ROWVERSION und TIMESTAMP. Weitere Informationen zur Nutzung finden Sie unter Verwenden von Babelfish-Funktionen mit eingeschränkter Implementierung.	strict
`escape_hatch_schemabinding_function`	Steuert das Verhalten von Babelfish im Zusammenhang mit der WITH SCHEMABINDING-Klausel. Standardmäßig wird die WITH SCHEMABINDING-Klausel ignoriert, wenn sie mit dem Befehl CREATE oder ALTER FUNCTION angegeben wird.	ignore

Escape-Luke	Beschreibung	Standard
`escape_hatch_schemabinding_procedure`	Steuert das Verhalten von Babelfish im Zusammenhang mit der WITH SCHEMABINDING-Klausel. Standardmäßig wird die WITH SCHEMABINDING-Klausel ignoriert, wenn sie mit dem Befehl CREATE oder ALTER PROCEDURE angegeben wird.	ignore
`escape_hatch_rowguidcol_column`	Steuert das Verhalten von Babelfish im Zusammenhang mit der ROWGUIDCOL-Klausel beim Erstellen oder Ändern einer Tabelle.	strict
`escape_hatch_schemabinding_trigger`	Steuert das Verhalten von Babelfish im Zusammenhang mit der WITH SCHEMABINDING-Klausel. Standardmäßig wird die WITH SCHEMABINDING-Klausel ignoriert, wenn sie mit dem Befehl CREATE oder ALTER TRIGGER angegeben wird.	ignore
`escape_hatch_schemabinding_view`	Steuert das Verhalten von Babelfish im Zusammenhang mit der WITH SCHEMABINDING-Klausel. Standardmäßig wird die WITH SCHEMABINDING-Klausel ignoriert, wenn sie mit dem Befehl CREATE oder ALTER VIEW angegeben wird.	ignore

Escape-Luke	Beschreibung	Standard
`escape_hatch_session_settings`	Steuert das Verhalten von Babelfish in Richtung nicht unterstützter SET-Anweisungen auf Sitzungsebene.	ignore
`escape_hatch_showplan_all`	Steuert das Verhalten von Babelfish in Bezug auf SET SHOWPLAN_ALL und SET STATISTICS PROFILE. Wenn sie auf „ignore“ eingestellt sind, verhalten sie sich wie SET BABELFISH_SHOWPLAN_ALL und SET BABELFISH_STATISTICS PROFILE; wenn sie auf „strict“ festgelegt sind, werden sie stillschweigend ignoriert.	strict
`escape_hatch_storage_on_partition`	Steuert das Verhalten von Babelfish im Zusammenhang mit der `ON partition_scheme column` -Klausel beim Definieren der Partitionierung. Babelfish implementiert derzeit keine Partitionierung.	strict

Escape-Luke	Beschreibung	Standard
`escape_hatch_storage_options`	Escape-Schraffur für jede Speicheroption, die in CREATE, ALTER DATABASE, TABLE, INDEX verwendet. Dazu gehören Klauseln (LOG) ON, TEXTIMAGE _ON, FILESTREAM_ON, die Speicherorte (Partitionen, Dateigruppen) für Tabellen, Indizes und Einschränkungen sowie für eine Datenbank definiere n. Diese Escape-Schraffurei nstellung gilt für alle diese Klauseln (einschließlich ON [PRIMARY] und ON „DEFAULT“). Die Ausnahme ist, wenn eine Partition für eine Tabelle oder einen Index mit ON partition _scheme (Spalte) angegeben wird.	ignore
`escape_hatch_table_hints`	Steuert das Verhalten von Tabellenhinweisen, die mit der WITH (...) -Klausel angegeben wurden.	ignore

Escape-Luke	Beschreibung	Standard
`escape_hatch_unique_constraint`	Bei Einstellung auf „strict“ (streng) kann ein obskurer semantisc her Unterschied zwischen SQL Server und PostgreSQL bei der Handhabung von NULL-Wert en in indizierten Spalten Fehler auslösen. Der semantische Unterschied tritt nur in unrealist ischen Anwendungsfällen auf, sodass Sie diese Escape-Schraffur auf „ignore“ (Ignorieren) festlegen können, um den Fehler nicht anzuzeigen. In den folgenden Versionen veraltet: 3.6.0 und höhere Versionen, 4.2.0 und höhere Versionen	strict

Erstellen eines DB-Clusters von Babelfish for Aurora PostgreSQL

Diese Version von Babelfish for Aurora PostgreSQL wird von Aurora PostgreSQL Version 13.4 und höher unterstützt.

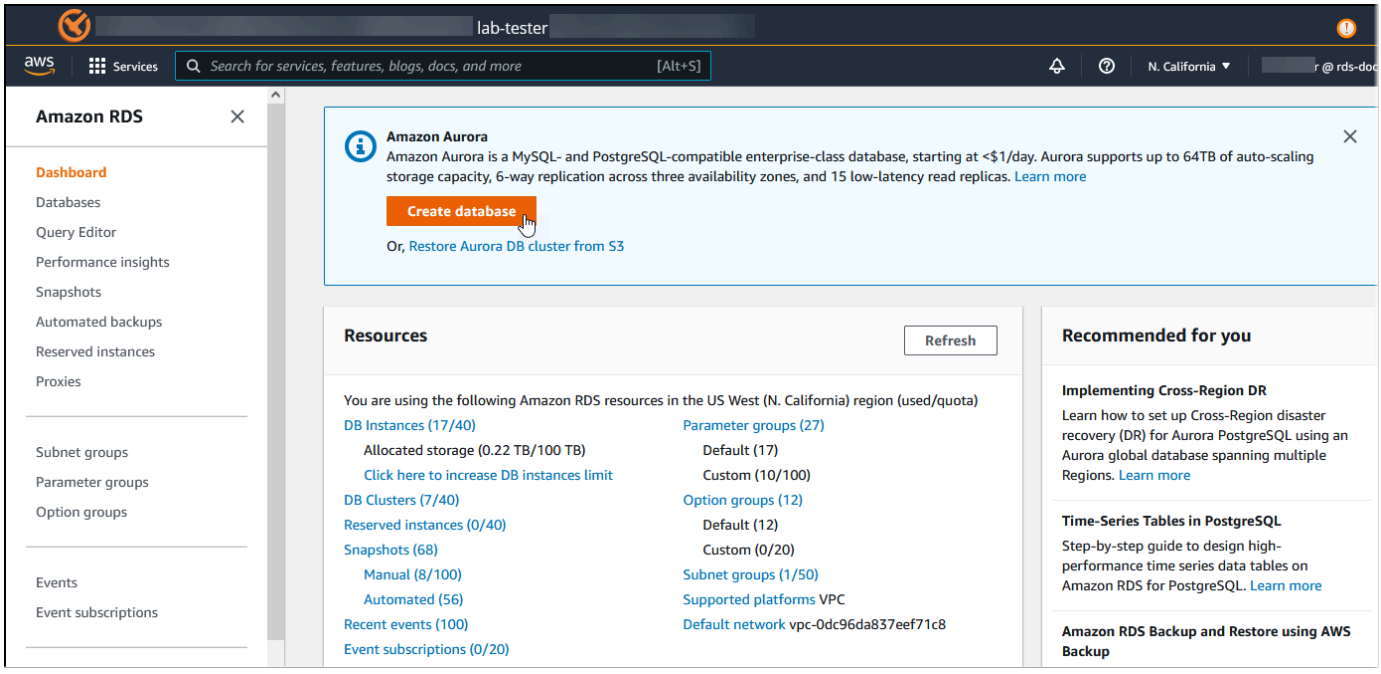
Sie können AWS-Managementkonsole oder AWS CLI verwenden, um einen Aurora-PostgreSQL-Cluster mit Babelfish zu erstellen.

> **Note**
>
> In einem Aurora-PostgreSQL-Cluster ist der Datenbankname `babelfish_db` für Babelfish reserviert. Das Erstellen einer eigenen „babelfish_db“-Datenbank auf einem Babelfish für Aurora PostgreSQL verhindert, dass Aurora Babelfish erfolgreich bereitstellt.

Konsole

So erstellen Sie einen Cluster mit Babelfish, der mit AWS-Managementkonsole läuft

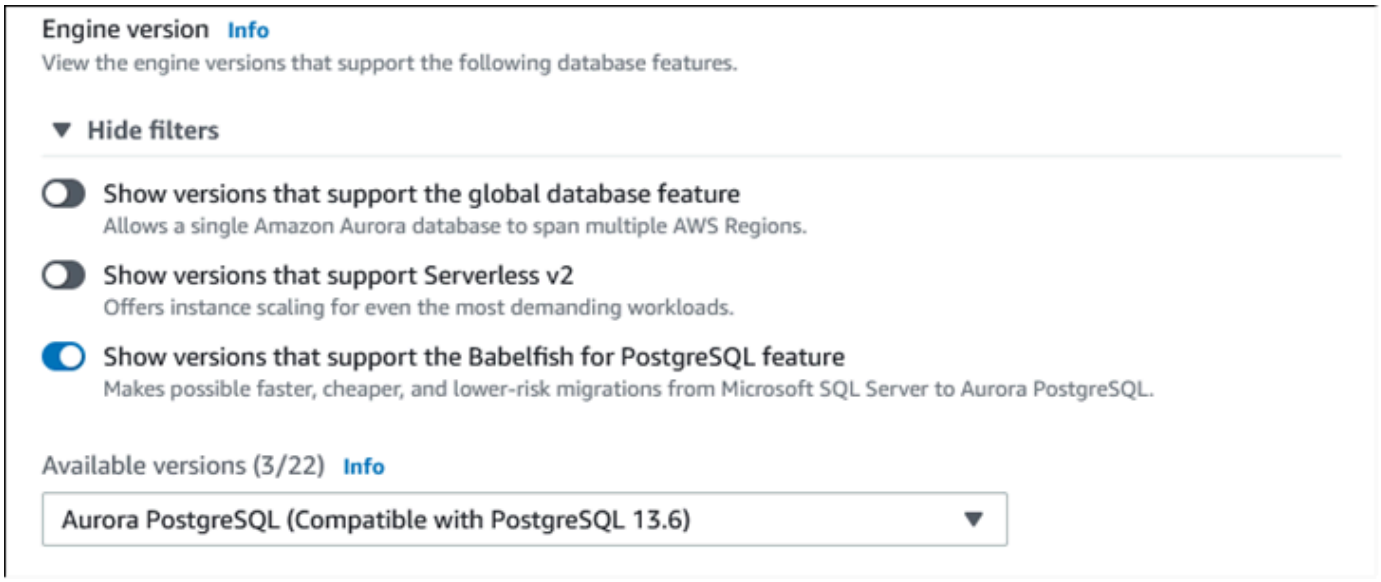
1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/ und wählen Sie Erstellen einer Datenbank aus.



2. Für Wählen Sie eine Datenbankerstellungsmethode aus führen Sie einen der folgenden Schritte aus:

- Um detaillierte Engine-Optionen festzulegen, wählen Sie Standard erstellen aus.
- Um vorkonfigurierte Optionen zu verwenden, die Best Practices für einen Aurora-Cluster unterstützen, wählen Sie Einfache Erstellung aus.

3. Wählen Sie als Engine-Typ Aurora (PostgreSQL-kompatibel).
4. Wählen Sie Filter anzeigen und danach Versionen anzeigen, welche die Funktion Babelfish für PostgreSQL-Funktion unterstützen, um die Engine-Typen aufzulisten, die Babelfish unterstützen. Babelfish wird derzeit von Aurora PostgreSQL Version 13.4 und höher unterstützt.
5. Für Verfügbare Versionen, wählen Sie eine Aurora-PostgreSQL-Version. Die neuesten Babelfish-Funktionen erhalten Sie, wenn Sie die höchste Aurora-PostgreSQL-Hauptversion auswählen.



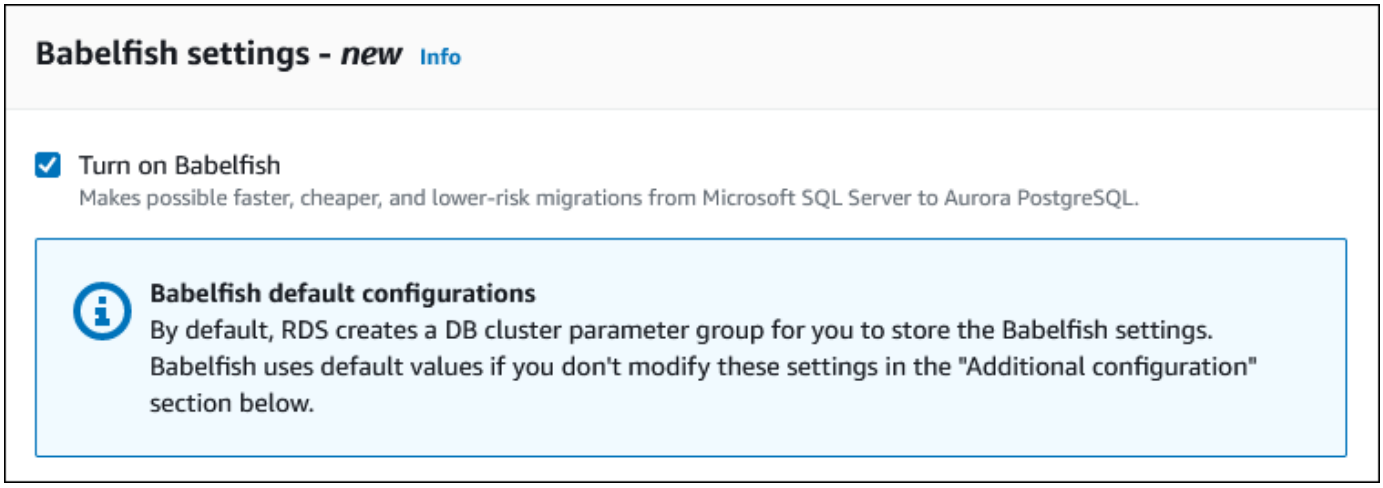
6. Wählen Sie unter Templates (Vorlagen) die Vorlage für Ihr Anwendungsszenario aus.
7. Für DB-Cluster-ID, geben Sie einen Namen ein, den Sie später in der DB-Clusterliste leicht finden können.
8. Für Master username (Masterbenutzername), geben Sie einen Administratorbenutzernamen ein. Der Standardwert für Aurora PostgreSQL ist `postgres`. Sie können den Standardwert akzeptieren oder einen anderen Namen auswählen. Um beispielsweise der Benennungskonvention zu folgen, die in Ihren SQL-Server-Datenbanken verwendet wird, können Sie `sa` (Systemadministrator) als Hauptbenutzername eingeben.

   Wenn Sie keinen Benutzer mit dem Namen `sa` zu diesem Zeitpunkt erstellen, können Sie später einen mit Ihrer Wahl des Kunden erstellen. Verwenden Sie nach dem Erstellen des Benutzers

den Befehl `ALTER SERVER ROLE`, um ihn der `sysadmin`-Gruppe (Rolle) für den Cluster hinzuzufügen.

> **Warning**
>
> Der Master-Benutzername muss immer Kleinbuchstaben verwenden, andernfalls kann der DB-Cluster keine Verbindung zu Babelfish über den TDS-Port herstellen.

9. Erstellen Sie für Master password (Hauptpasswort) ein sicheres Passwort und bestätigen Sie es.
10. Für die folgenden Optionen bis zum Abschnitt Babelfish Einstellungen, geben Sie Ihre DB-Cluster-Einstellungen an. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.
11. Um die Babelfish-Funktionalität verfügbar zu machen, wählen Sie Aktivieren Sie Babelfish aus.



12. Für DB-Cluster-Parametergruppe führen Sie einen der folgenden Schritte aus:

    - Klicken Sie auf Neue erstellen, um eine neue Parametergruppe zu erstellen, bei der Babelfish aktiviert ist.
    - Klicken Sie auf Wählen Sie existierende, um eine vorhandene Parametergruppe zu verwenden. Wenn Sie eine vorhandene Gruppe verwenden, müssen Sie die Gruppe ändern, bevor Sie den Cluster erstellen, und fügen Sie Werte für Babelfish-Parameter hinzu. Informationen zum Festlegen von Parametern finden Sie unter Einstellungen der DB-Cluster-Parametergruppe für Babelfish.

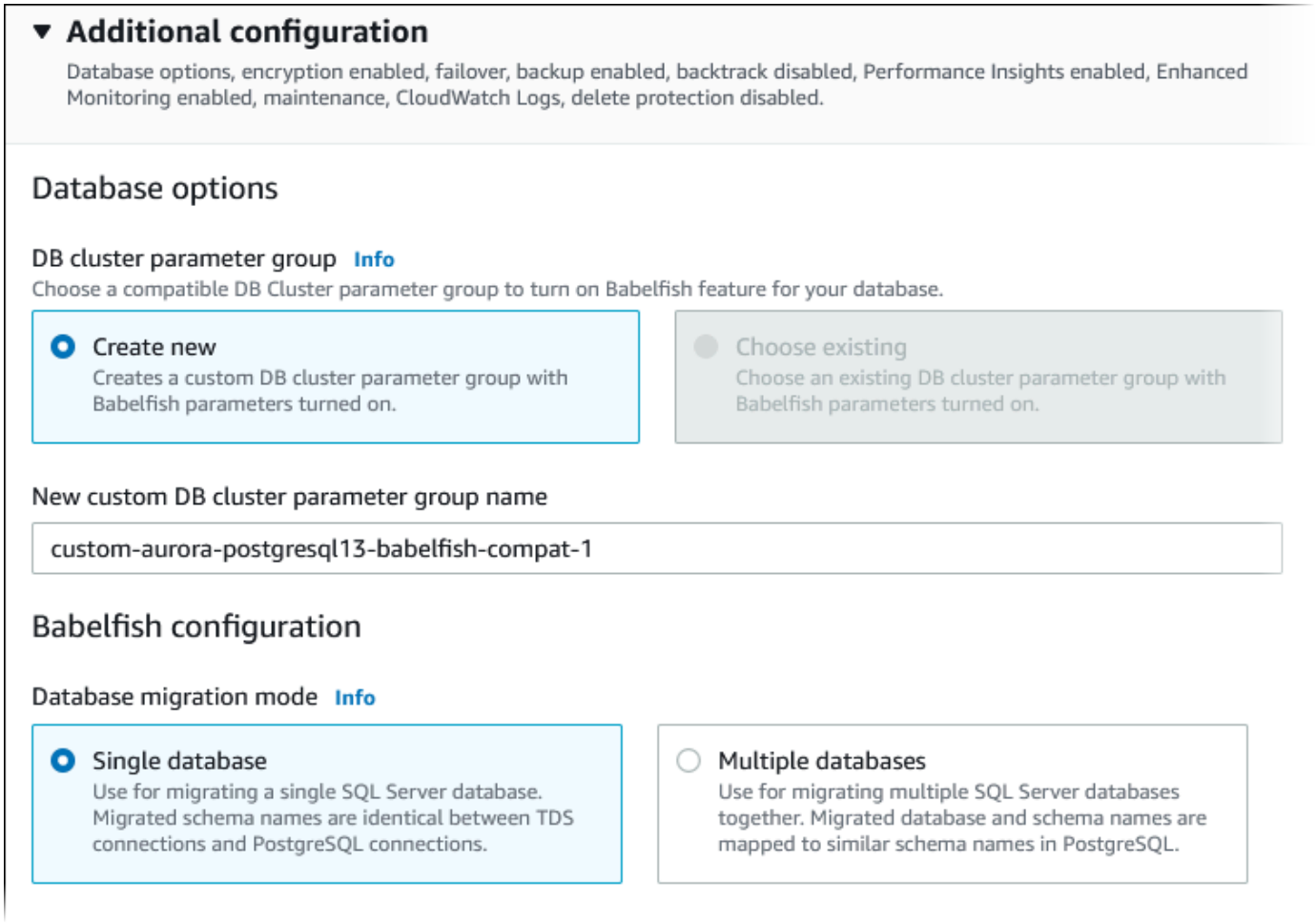
      Wenn Sie eine vorhandene Gruppe verwenden, geben Sie den Gruppennamen in das folgende Feld ein.

13. Wählen Sie für Migration Mode eine der nachstehenden Optionen aus:

- Einzelne Datenbank, um eine einzelne SQL Server-Datenbank zu migrieren.

  In einigen Fällen können Sie mehrere Benutzerdatenbanken zusammen migrieren, wobei Ihr Endziel eine vollständige Migration zur nativen Aurora PostgreSQL ohne Babelfish ist. Wenn die endgültigen Anträge konsolidierte Schemas erfordern (ein einziges dbo-Schema), stellen Sie sicher, dass Sie Ihre SQL Server-Datenbanken zuerst in einer einzigen SQL-Server-Datenbank konsolidieren. Migrieren Sie dann zu Babelfish mit dem Einzelne Datenbank-Modus.
- Mehrere Datenbanken, um mehrere SQL Server-Datenbanken zu migrieren (die aus einer einzigen SQL Server-Installation stammen). Der Mehrfachdatenbankmodus konsolidiert nicht mehrere Datenbanken, die nicht von einer einzigen SQL Server-Installation stammen. Weitere Informationen zum Migrieren mehrerer Datenbanken finden Sie unter Verwenden von Babelfish mit einer einzigen Datenbank oder mehreren Datenbanken.

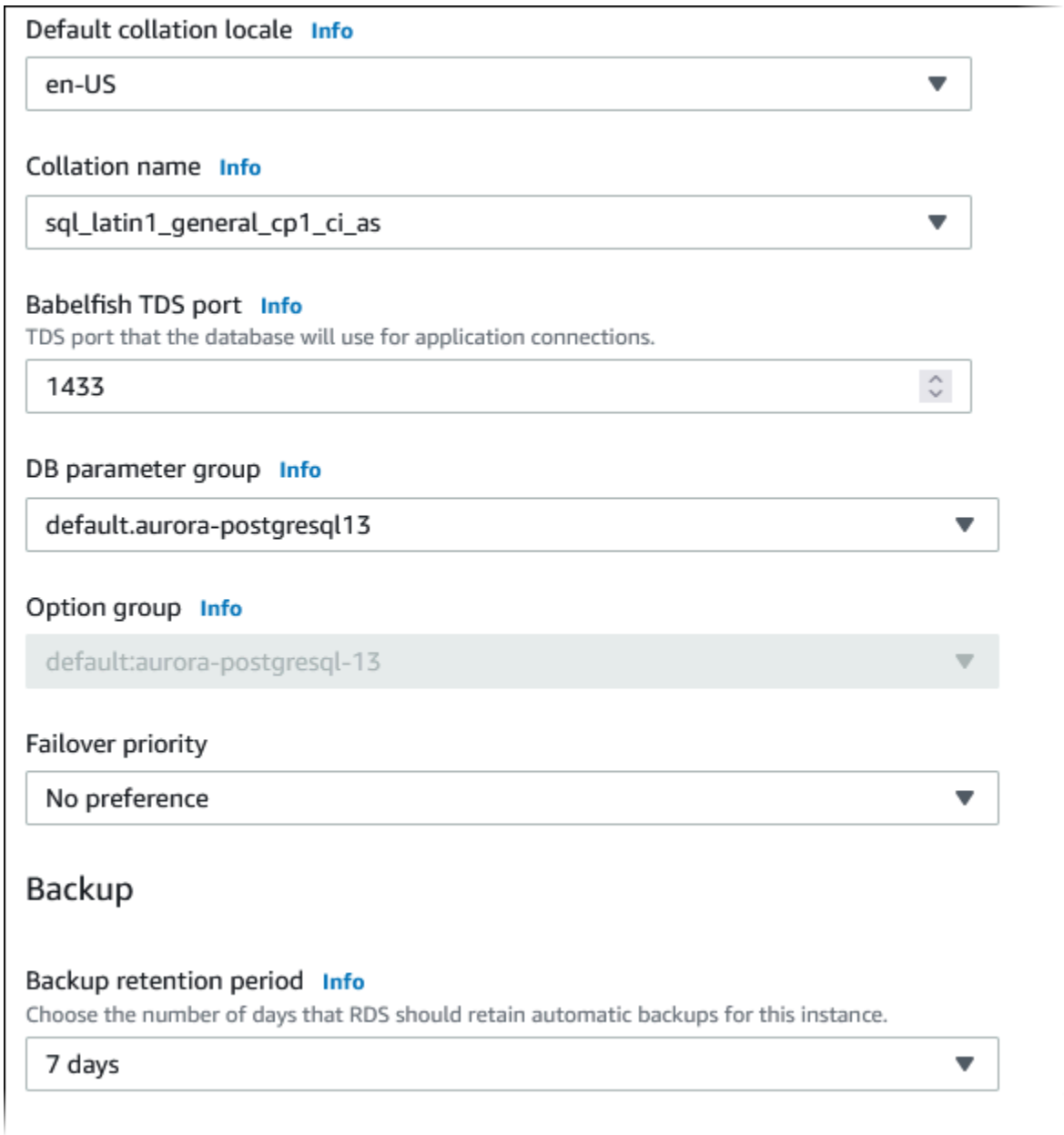
> **Note**
>
> Ab Aurora-PostgreSQL-Version 16 ist die Option Mehrere Datenbanken standardmäßig als Datenbankmigrationsmodus ausgewählt.



14. Für Standardmäßige Sortierreihenfolgen geben Sie das Gebietsschema des Servers ein. Der Standardwert ist `en-US`. Ausführliche Informationen zu Aliassen finden Sie unter Verstehen der Kollationen in Babelfish für Aurora PostgreSQL.
15. Für Name der Sortierung, geben Sie Ihre Standardsortierung ein. Der Standardwert ist `sql_latin1_general_cp1_ci_as`. Weitere Informationen hierzu finden Sie unter Verstehen der Kollationen in Babelfish für Aurora PostgreSQL.
16. Geben Sie für den Babelfish TDS-Port den Standardport `1433` ein. Derzeit unterstützt Babelfish nur den Port `1433` für Ihren DB-Cluster.
17. Für DB-Parametergruppe wählen Sie eine Parametergruppe aus oder lassen Sie Aurora eine neue Gruppe mit Standardeinstellungen erstellen.
18. Für Failover priority (Failover-Priorität) wählen Sie eine Failover-Priorität für die Instance aus. Wenn Sie keinen Wert auswählen, wird als Standard `tier-1` gesetzt. Diese Priorität bestimmt die Reihenfolge, in der -Replicas bei der Wiederherstellung nach einem Ausfall der primären

Instance hochgestuft werden. Weitere Informationen finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.

19. Wählen Sie für Aufbewahrungszeitraum für Backups die Dauer (1–35 Tage) aus, die Aurora Backup-Kopien der Datenbank aufbewahrt. Sie können Sicherungskopien für sekundengenaue Point-in-Time-Wiederherstellungen (PITR) der Datenbank verwenden. Der Standardaufbewahrungszeitraum beträgt sieben Tage.



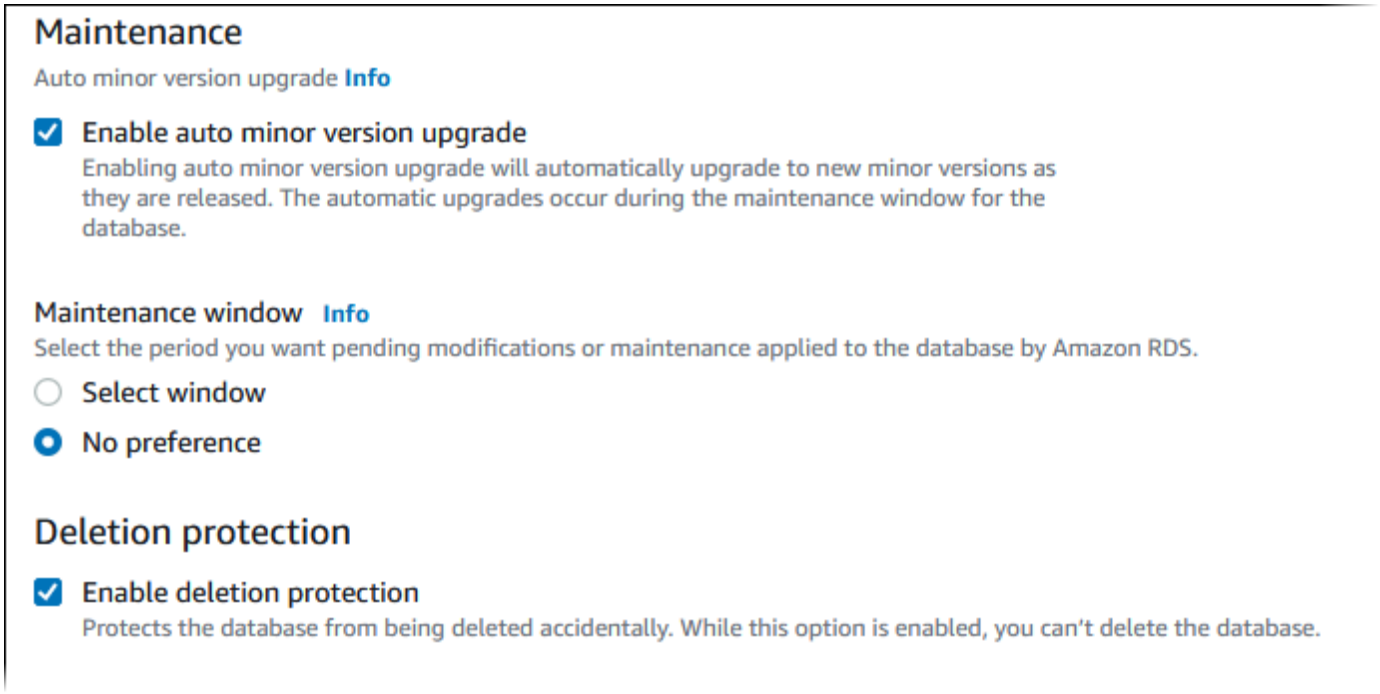
20. Wählen Sie Tags zu Snapshots kopieren, wenn beim Erstellen eines Snapshots DB-Instance-Tags in den DB-Snapshot kopiert werden sollen.

> **Note**
>
> Wenn Sie einen DB-Cluster aus einem Snapshot wiederherstellen, wird er nicht als DB-Cluster von Babelfish für Aurora PostgreSQL wiederhergestellt. Sie müssen die Parameter, die die Babelfish-Einstellungen steuern, in der DB-Cluster-Parametergruppe aktivieren, um Babelfish wieder zu aktivieren. Weitere Informationen zu diesen Babelfish-Parametern finden Sie unter Einstellungen der DB-Cluster-Parametergruppe für Babelfish.

21. Klicken Sie auf Enable encryption (Verschlüsselung aktivieren), um die Verschlüsselung im Ruhezustand (Aurora-Speicherverschlüsselung) für diesen DB-Cluster einzuschalten.
22. Klicken Sie auf Enable Performance Insights (Performance Insights aktivieren), um Amazon RDS Performance Insights zu aktivieren.
23. Wählen Sie Enable enhanced monitoring (Erweiterte Überwachung aktivieren) aus, um die Erfassung von Metriken in Echtzeit für das Betriebssystem zu aktivieren, in dem Ihr DB-Cluster ausgeführt wird.
24. Klicken Sie auf PostgreSQL-Protokoll, um die Protokolldateien in Amazon CloudWatch Logs zu veröffentlichen.
25. Klicken Sie auf Automatischer Unterversion-Upgrade aktivieren, um Ihren Aurora-DB-Cluster automatisch zu aktualisieren, wenn ein Minor-Versions-Upgrade verfügbar ist.
26. Für Maintenance window (Wartungsfenster) gehen Sie wie folgt vor:

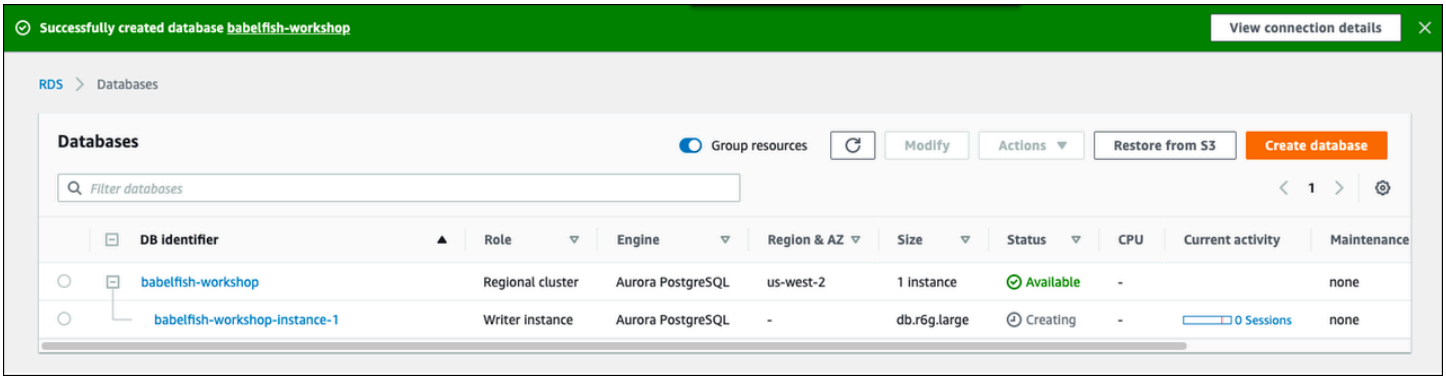
    - Um einen Zeitpunkt auszuwählen, an dem Amazon RDS Änderungen vornehmen oder Wartungsarbeiten durchführen kann, wählen Sie Fenster wählen aus.
    - Um die Amazon-RDS-Wartung zu einem ungeplanten Zeitpunkt durchzuführen, wählen Sie Keine Präferenz aus.
27. Wählen Sie Enable deletion protection (Löschschutz aktivieren), um Ihre Datenbank vor versehentlichem Löschen zu schützen.

    Wenn Sie diese Funktion aktivieren, können Sie die Datenbank nicht direkt löschen. Stattdessen müssen Sie den Datenbankcluster ändern und diese Funktion deaktivieren, bevor Sie die Datenbank löschen.



28. Wählen Sie Create database (Datenbank erstellen) aus.

Ihre neue Datenbank für Babelfish finden Sie in der Auflistung Datenbanken. Die Spalte Status zeigt Verfügbar an, wenn die Bereitstellung abgeschlossen ist.



AWS CLI

Wenn Sie einen Babelfish for Aurora PostgreSQL erstellen: Mit der AWS CLI müssen Sie dem Befehl den Namen der DB-Cluster-Parametergruppe übergeben, die für den Cluster verwendet werden soll. Weitere Informationen finden Sie unter Voraussetzungen für DB-Cluster.

Bevor Sie die AWS CLI verwenden können um einen Aurora-PostgreSQL-Cluster mit Babelfish zu erstellen, gehen Sie wie folgt vor:

- Wählen Sie Ihre Endpunkt-URL aus der Liste der Services unter Amazon Aurora Aurora-Endpunkte und -Kontingente aus.
- Erstellen einer Parametergruppe für das Cluster. Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.
- Ändern Sie die Parametergruppe und fügen Sie den Parameter hinzu, der Babelfish aktiviert.

So erstellen Sie ein Aurora-PostgreSQL-DB-Cluster über die AWS CLI

Die folgenden Beispiele verwenden den standardmäßigen Hauptbenutzernamen, `postgres`. Ersetzen Sie den Namen nach Bedarf durch den Benutzernamen, den Sie für Ihren DB-Cluster erstellt haben, z. B. `sa` bzw. den Benutzernamen, den Sie anstelle des Standardwerts ausgewählt haben.

1. Erstellen Sie eine Parametergruppe.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-cluster-parameter-group \
   --endpoint-url endpoint-url \
   --db-cluster-parameter-group-name parameter-group \
   --db-parameter-group-family aurora-postgresql14 \
   --description "description"
   ```

   Für Windows:

   ```
   aws rds create-db-cluster-parameter-group ^
   --endpoint-url endpoint-URL ^
   --db-cluster-parameter-group-name parameter-group ^
   --db-parameter-group-family aurora-postgresql14 ^
   --description "description"
   ```

2. Ändern Sie Ihre Parametergruppe, um Babelfish zu aktivieren.

   Für Linux, macOS oder Unix:

   ```
   aws rds modify-db-cluster-parameter-group \
   --endpoint-url endpoint-url \
   --db-cluster-parameter-group-name parameter-group \
   ```

```
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-group ^
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. Geben Sie die DB-Subnetzgruppen- und VPC-Sicherheitsgruppen-ID für Ihr neues DB-Cluster an und rufen Sie dann den -Befehl create-db-cluster auf.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-cluster \
   --db-cluster-identifier  cluster-name\
   --master-username postgres \
   --manage-master-user-password \
   --engine aurora-postgresql \
   --engine-version 14.3            \
   --vpc-security-group-ids security-group \
   --db-subnet-group-name subnet-group-name \
   --db-cluster-parameter-group-name parameter-group
   ```

   Für Windows:

   ```
   aws rds create-db-cluster ^
   --db-cluster-identifier cluster-name ^
   --master-username postgres ^
   --manage-master-user-password ^
   --engine aurora-postgresql ^
   --engine-version 14.3 ^
   --vpc-security-group-ids security-group ^
   --db-subnet-group-name subnet-group ^
   --db-cluster-parameter-group-name parameter-group
   ```

   In diesem Beispiel wird die Option `--manage-master-user-password` zum Generieren des Hauptbenutzerpassworts und zum Verwalten dieses Passworts in Secrets Manager angegeben. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS

Secrets Manager. Alternativ können Sie die Option `--master-password` verwenden, um das Passwort selbst festzulegen und zu verwalten.

4. Erstellen Sie die primäre Instance für Ihren DB-Cluster explizit. Verwenden Sie den Namen des Clusters, den Sie in Schritt 3 erstellt haben, für das Argument `--db-cluster-identifier` beim Aufrufen des Befehls create-db-instance, wie nachfolgend dargestellt.

   Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
--db-instance-identifier instance-name \
--db-instance-class db.r6g \
--db-subnet-group-name subnet-group \
--db-cluster-identifier cluster-name \
--engine aurora-postgresql
```

   Für Windows:

```
aws rds create-db-instance ^
--db-instance-identifier instance-name ^
--db-instance-class db.r6g ^
--db-subnet-group-name subnet-group ^
--db-cluster-identifier cluster-name ^
--engine aurora-postgresql
```

Migrieren einer SQL-Server-Datenbank zu Babelfish for Aurora PostgreSQL

Sie können Babelfish for Aurora PostgreSQL verwenden, um die Migration einer SQL-Server-Datenbank zu einem DB-Cluster von Amazon Aurora PostgreSQL zu erleichtern. Vor der Migration überprüfen Sie Verwenden von Babelfish mit einer einzigen Datenbank oder mehreren Datenbanken.

Themen

- Übersicht über den Migrationsprozess
- Auswertung und Handhabung von Unterschieden zwischen SQL Server und Babelfish
- Importieren/Exportieren von Tools für die Migration von SQL Server zu Babelfish

Übersicht über den Migrationsprozess

In der folgenden Zusammenfassung sind die Schritte aufgeführt, die erforderlich sind, damit Ihre SQL-Server-Anwendung mit Babelfish arbeiten kann. Informationen zu den Tools, die Sie für die Export- und Importprozesse verwenden können, sowie weitere Informationen finden Sie unter Importieren/Exportieren von Tools für die Migration von SQL Server zu Babelfish. Zum Laden der Daten empfehlen wir, AWS DMS mit einem DB-Cluster von Aurora PostgreSQL als Zielendpunkt zu verwenden.

1. Erstellen Sie einen neuen DB-Cluster von Aurora PostgreSQL und lassen Sie Babelfish aktiviert. Um zu erfahren wie dies geht, vgl. Erstellen eines DB-Clusters von Babelfish for Aurora PostgreSQL.

   Um die verschiedenen SQL-Artefakte zu importieren, die aus Ihrer SQL-Server-Datenbank exportiert werden, verbinden Sie sich mithilfe eines nativen SQL-Server-Tools wie sqlcmd mit dem Babelfish-Cluster. Weitere Informationen finden Sie unter Verbinden mit Ihrem DB-Cluster mithilfe eines SQL Server-Clients.
2. Exportieren Sie die Datendefinitionssprache (DDL) in Ihrer SQL-Server-Datenbank, die Sie migrieren möchten. Die DDL ist SQL-Code, der Datenbankobjekte beschreibt, die Benutzerdaten (wie Tabellen, Indizes und Ansichten) und vom Benutzer geschriebenen Datenbankcode (wie gespeicherte Prozeduren, benutzerdefinierte Funktionen und Trigger) enthalten.

   Weitere Informationen finden Sie unter Migrieren zu Babelfish mit SQL Server Management Studio (SSMS).
3. Führen Sie ein Bewertungstool aus, um den Umfang aller Änderungen zu bewerten, die Sie möglicherweise vornehmen müssen, damit Babelfish die auf SQL Server ausgeführte Anwendung

effektiv unterstützen kann. Weitere Informationen finden Sie unter Auswertung und Handhabung von Unterschieden zwischen SQL Server und Babelfish.

4. Überprüfen Sie die Einschränkungen des AWS DMS-Zielendpunkts und aktualisieren Sie das DDL-Skript nach Bedarf. Weitere Informationen finden Sie unter Einschränkungen bei der Verwendung eines PostgreSQL-Zielendpunkts mit Babelfish-Tabellen unter Verwenden von Babelfish für Aurora PostgreSQL als Ziel.
5. Führen Sie die DDL auf Ihrem neuen Babelfish-DB-Cluster innerhalb Ihrer angegebenen T-SQL-Datenbank aus, um nur die Schemas, benutzerdefinierten Datentypen und Tabellen mit ihren Primärschlüsseleinschränkungen zu erstellen.
6. Verwenden Sie AWS DMS, um Ihre Daten von SQL Server zu Babelfish-Tabellen zu migrieren. Verwenden Sie für die kontinuierliche Replikation mit Change Data Capture bei SQL Server oder SQL Replication Aurora PostgreSQL anstelle von Babelfish als Endpunkt. Informationen dazu finden Sie im Artikel Verwendung von Babelfish für Aurora PostgreSQL als Ziel für AWS Database Migration Service.
7. Wenn das Laden der Daten abgeschlossen ist, erstellen Sie alle verbleibenden T-SQL-Objekte, die die Anwendung auf Ihrem Babelfish-Cluster unterstützen.
8. Konfigurieren Sie Ihre Client-Anwendung neu, um eine Verbindung mit dem Babelfish-Endpunkt statt mit der SQL-Server-Datenbank herzustellen. Weitere Informationen finden Sie unter Verbinden mit einem Babelfish-DB-Cluster.
9. Ändern Sie Ihre Anwendung nach Bedarf und testen Sie sie erneut. Weitere Informationen finden Sie unter Unterschiede zwischen Babelfish für Aurora PostgreSQL und SQL Server.

Sie müssen nach wie vor Ihre clientseitigen SQL-Abfragen bewerten. Die aus Ihrer SQL-Server-Instance generierten Schemas konvertieren nur den serverseitigen SQL-Code. Wir empfehlen, die folgenden Schritte auszuführen:

- Erfassen Sie clientseitige Abfragen mit dem SQL Server Profiler anhand der vordefinierten Vorlage TSQL_Replay. Diese Vorlage erfasst Informationen zu T-SQL-Anweisungen, die Sie dann für iterative Optimierung und Tests wiedergeben können. Sie können den Profiler über das Menü Tools in SQL Server Management Studio starten. Klicken Sie auf SQL Server Profiler, um den Profiler zu öffnen und die Vorlage TSQL_Replay auszuwählen.

  Zur Verwendung für Ihre Babelfish-Migration starten Sie eine Nachverfolgung und führen Sie Ihre Anwendung dann mit Ihren Funktionstests aus. Der Profiler erfasst die T-SQL-Anweisungen. Stoppen Sie nach dem Abschluss des Tests die Nachverfolgung. Speichern Sie das Ergebnis mit

Ihren clientseitigen Abfragen in einer XML-Datei (File (Datei) > Save as (Speichern unter) > Trace XML File for Replay (XML-Datei für Wiederholung nachverfolgen)).

Weitere Informationen finden Sie unter SQL Server Profiler in der Microsoft-Dokumentation. Weitere Informationen zur TSQL_Replay-Vorlage finden Sie unter Vorlagen für SQL Server Profiler.

- Für Anwendungen mit komplexen clientseitigen SQL-Abfragen empfehlen wir Ihnen, diese mit Babelfish Compass auf Babelfish-Kompatibilität zu analysieren. Wenn die Analyse ergibt, dass die clientseitigen SQL-Anweisungen nicht unterstützte SQL-Funktionen enthalten, überprüfen Sie die SQL-Aspekte in der Clientanwendung und nehmen Sie ggf. Änderungen vor.
- Sie können die SQL-Abfragen auch als erweiterte Ereignisse (XEL-Format) erfassen. Verwenden Sie dazu den SSMS XEvent Profiler. Nachdem Sie die XEL-Datei generiert haben, extrahieren Sie die SQL-Anweisungen in XML-Dateien, die Compass dann verarbeiten kann. Weitere Informationen finden Sie unter Verwenden des SSMS XEvent Profilers in der Microsoft-Dokumentation.

Wenn Sie mit allen Tests, Analysen und Änderungen an Ihrer migrierten Anwendung zufrieden sind, können Sie Ihre Babelfish-Datenbank für die Produktion verwenden. Beenden Sie dazu die ursprüngliche Datenbank und leiten Sie Live-Clientanwendungen auf den Babelfish-TDS-Port um.

> **Note**
>
> AWS DMS unterstützt jetzt die Replikation von Daten aus Babelfish. Weitere Informationen finden Sie unter AWS DMS unterstützt jetzt Babelfish für Aurora PostgreSQL als Ziel.

Auswertung und Handhabung von Unterschieden zwischen SQL Server und Babelfish

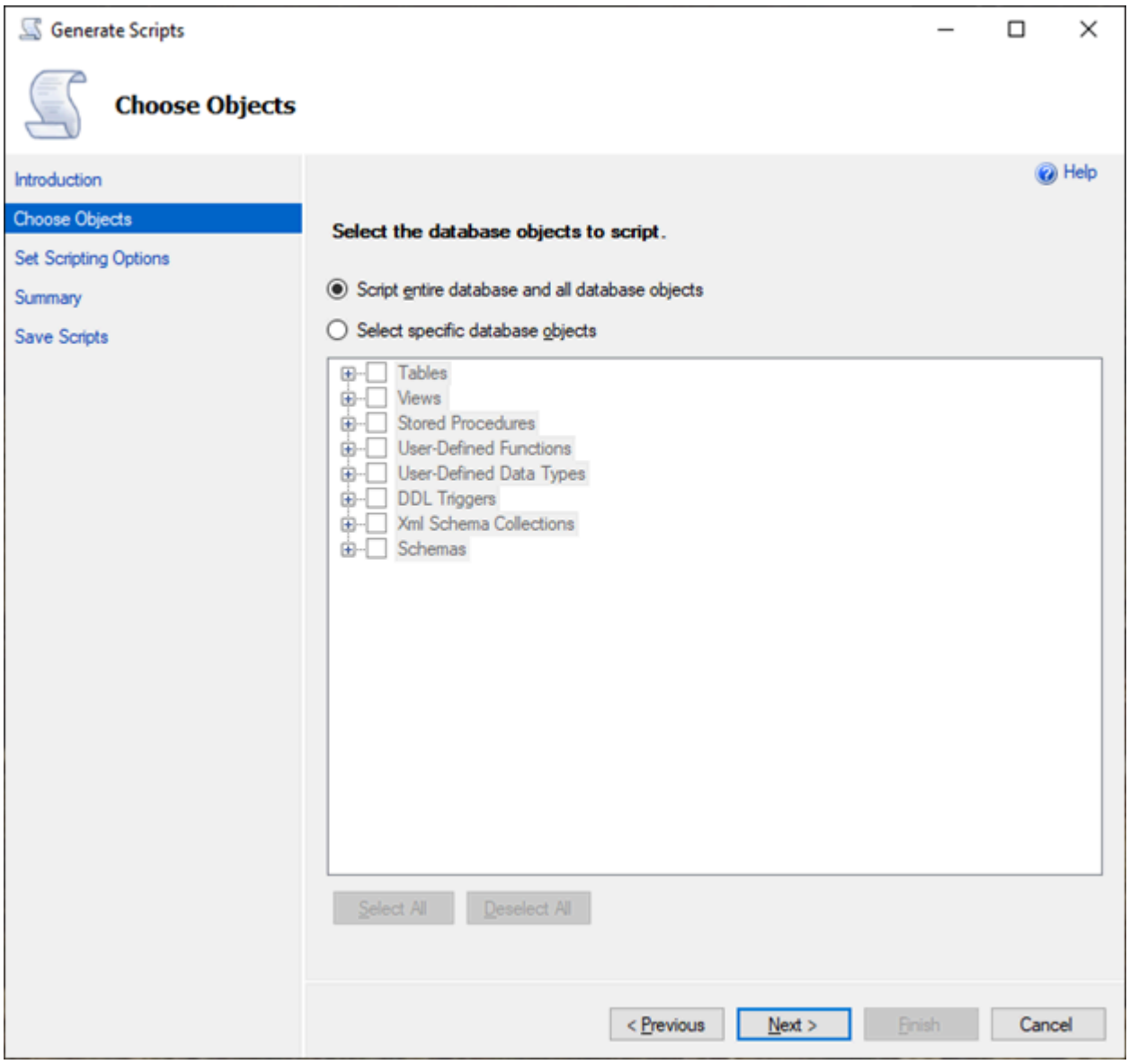
Für beste Ergebnisse empfehlen wir Ihnen, den generierten DDL/DML und den Client-Abfragecode auszuwerten, bevor Sie Ihre SQL-Server-Datenbankanwendung tatsächlich zu Babelfish migrieren. Abhängig von der Babelfish-Version und den spezifischen Funktionen von SQL Server, die Ihre Anwendung implementiert, müssen Sie möglicherweise Ihre Anwendung umgestalten oder Alternativen für Funktionen verwenden, die in Babelfish nicht vollständig unterstützt werden.

- Verwenden Sie zur Bewertung Ihres SQL-Server-Anwendungscodes Babelfish Compass für die generierte DDL, um festzustellen, wie viel T-SQL-Code von Babelfish unterstützt wird. Identifizieren Sie T-SQL-Code, für den vor der Ausführung in Babelfish möglicherweise Änderungen erforderlich sind. Weitere Informationen zu diesem Tool finden Sie unter Babelfish-Compass-Tool auf GitHub.

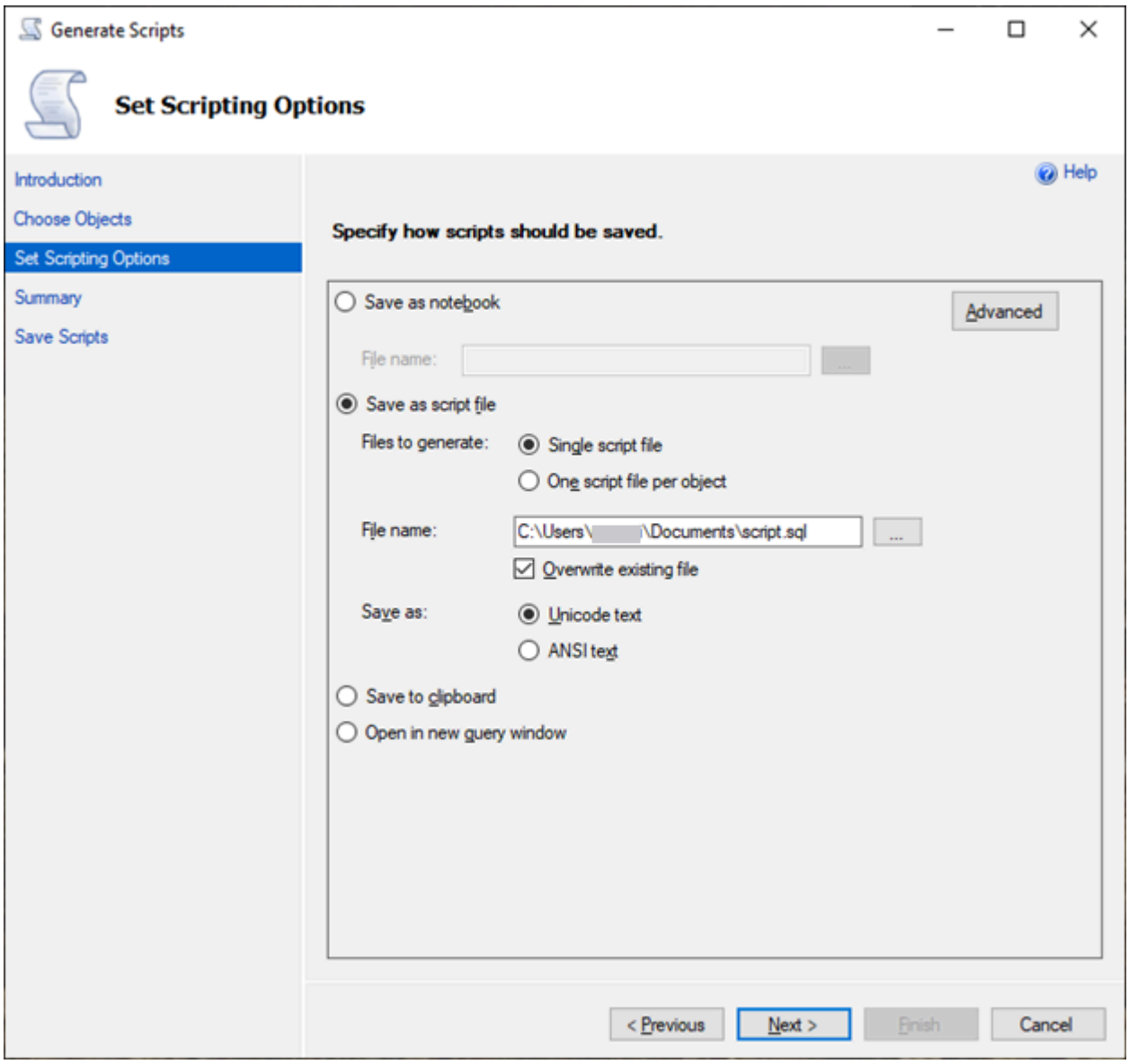
> **Note**
>
> Babelfish Compass ist ein Open-Source-Tool. Melden Sie Probleme mit Babelfish Compass über GitHub statt an den AWS-Support.

Sie können den Assistenten zum Generieren von Skripts mit SQL Server Management Studio (SSMS) verwenden, um die SQL-Datei zu generieren, die von Babelfish Compass oder die AWS Schema Conversion Tool-CLI bewertet wird. Wir empfehlen die folgenden Schritte, um die Bewertung zu optimieren.

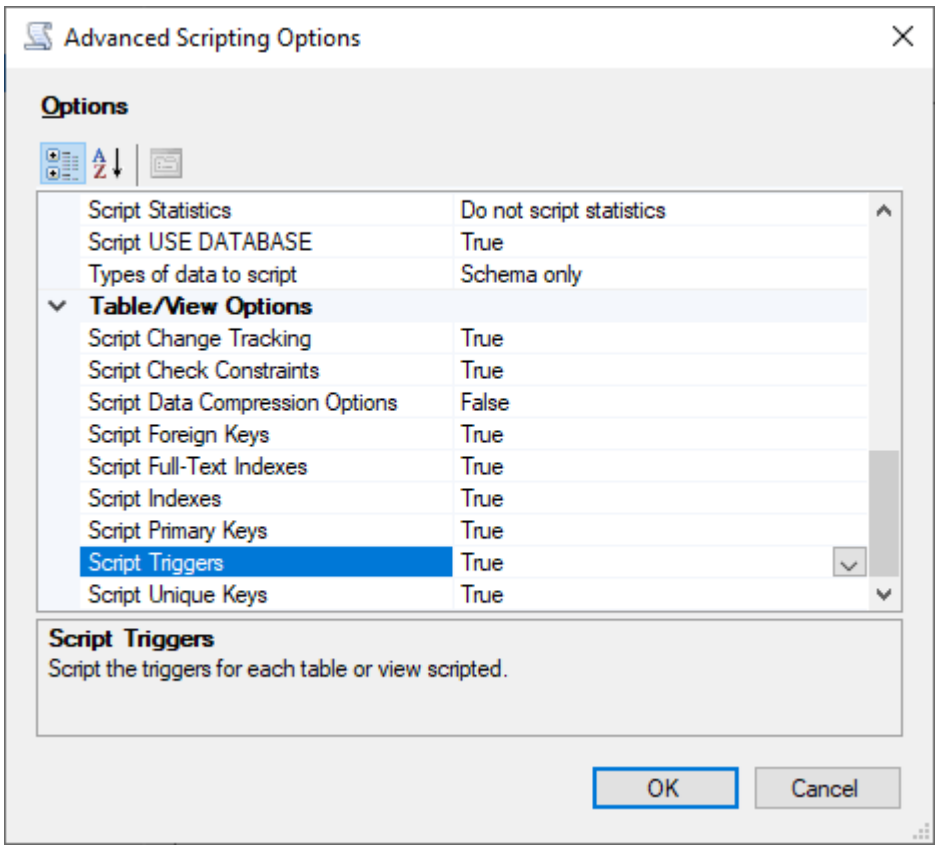
1. Wählen Sie auf der Seite Choose Objects (Objekte auswählen) die Option Script entire database and all database objects (Gesamte Datenbank und alle Datenbankobjekte in das Skript aufnehmen) aus.



2. Wählen Sie für Set Scripting Options (Scripting-Optionen festlegen) die Option Save as script file (Als Skriptdatei speichern) und Single script file (Einzelne Skriptdatei) aus.



3. Wählen Sie Advanced (Erweitert) aus, um die standardmäßigen Scripting-Optionen zu ändern und Funktionen zu identifizieren, die für eine vollständige Bewertung normalerweise auf „False“ (Falsch) festgelegt sind:
   - Skriptänderungsverfolgung auf „True“ (Wahr)
   - Skriptvolltextindizes auf „True“ (Wahr)
   - Skriptauslöser auf „True“ (Wahr)
   - Skriptanmeldungen auf „True“ (Wahr)
   - Skripteigentümer auf „True“ (Wahr)
   - Skriptberechtigungen auf Objektebene auf „True“ (Wahr)
   - Skriptsortierungen auf „True“ (Wahr)



4. Führen Sie die restlichen Schritte im Assistenten aus, um die Datei zu generieren.

Importieren/Exportieren von Tools für die Migration von SQL Server zu Babelfish

Wir empfehlen, AWS DMS als primäres Tool für die Migration von SQL Server zu Babelfish zu verwenden. Babelfish unterstützt jedoch verschiedene andere Möglichkeiten, Daten mithilfe von SQL-Server-Tools zu migrieren, darunter die folgenden.

- SQL Server Integration Services (SSIS) für alle Versionen von Babelfish. Weitere Informationen finden Sie unter Migration von SQL Server zu Aurora PostgreSQL mit SSIS und Babelfish.
- Verwenden Sie den Import/Export-Assistenten von SSMS für Babelfish-Versionen 2.1.0 und höher. Dieses Tool ist nicht nur über das SSMS, sondern auch als eigenständiges Tool verfügbar. Weitere Informationen finden Sie unter Willkommen beim SQL-Server-Assistenten für Import und Export in der Microsoft-Dokumentation.
- Mit dem Microsoft-Dienstprogramm zum Kopieren von Massendaten (bcp) können Sie Daten einer Instance von Microsoft SQL Server in eine Datendatei im von Ihnen angegebenen Format kopieren. Weitere Informationen finden Sie unter bcp-Hilfsprogramm in der Microsoft-Dokumentation. Babelfish unterstützt jetzt die Datenmigration mit dem BCP-Client und das bcp-

Dienstprogramm unterstützt jetzt `-E`-Flags (für Identitätsspalten) und -b-Flags (für stapelweise Einfügungen). Bestimmte bcp-Optionen werden nicht unterstützt, einschließlich `-C`, `-T`, `-G`, `-K`, `-R`, `-V` und `-h`.

Migrieren zu Babelfish mit SQL Server Management Studio (SSMS)

Wir empfehlen, separate Dateien für jeden der spezifischen Objekttypen zu generieren. Sie können den Assistenten zum Generieren von Skripts in SSMS zunächst für jeden Satz von DDL-Anweisungen verwenden und dann die Objekte als Gruppe ändern, um alle während der Bewertung festgestellten Probleme zu beheben.

Führen Sie diese Schritte aus, um die Daten mithilfe von AWS DMS oder anderen Datenmigrationsmethoden zu migrieren. Führen Sie zuerst diese Methoden zum Erstellen von Skripts aus, um die Daten schneller und besser in die Babelfish-Tabellen in Aurora PostgreSQL zu laden.

1. Führen Sie `CREATE SCHEMA`-Anweisungen aus.
2. Führen Sie `CREATE TYPE`-Anweisungen aus, um benutzerdefinierte Datentypen zu erstellen.
3. Führen Sie grundlegende `CREATE TABLE`-Anweisungen mit den Primärschlüsseln oder eindeutigen Einschränkungen aus.

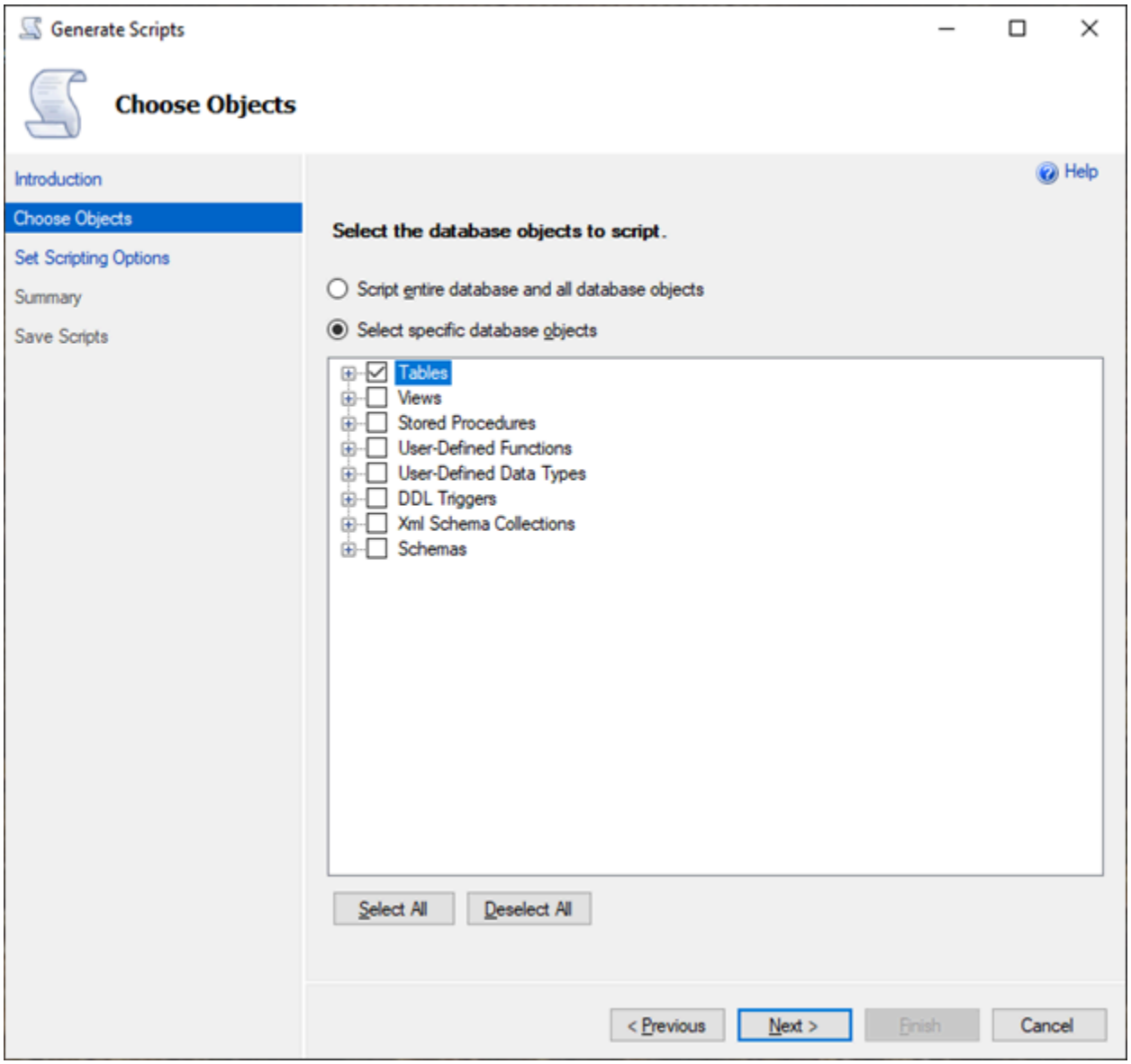
Führen Sie das Laden der Daten mit dem empfohlenen Import-/Export-Tool durch. Führen Sie die geänderten Skripts für die folgenden Schritte aus, um die verbleibenden Datenbankobjekte hinzuzufügen. Sie benötigen die Create-Table-Anweisungen, um diese Skripts für die Einschränkungen, Auslöser und Indizes auszuführen. Nachdem die Skripts generiert wurden, löschen Sie die Create-Table-Anweisungen.

1. Führen Sie `ALTER TABLE`-Anweisungen für die Prüfbeschränkungen, Fremdschlüsseleinschränkungen und Standardeinschränkungen aus.
2. Führen Sie `CREATE TRIGGER`-Anweisungen aus.
3. Führen Sie `CREATE INDEX`-Anweisungen aus.
4. Führen Sie `CREATE VIEW`-Anweisungen aus.
5. Führen Sie `CREATE STORED PROCEDURE`-Anweisungen aus.

So generieren Sie Skripts für jeden Objekttyp

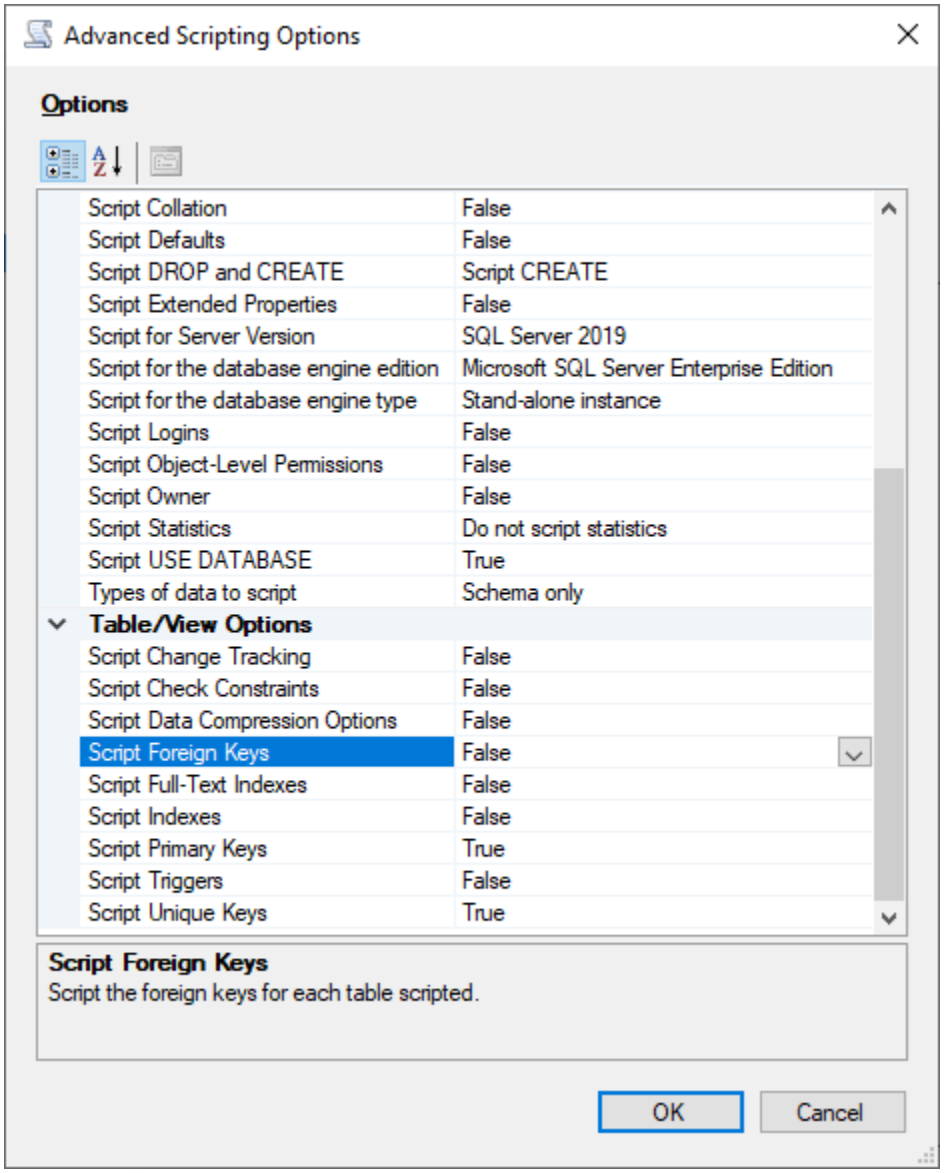
Führen Sie die folgenden Schritte aus, um die grundlegenden Create-Table-Anweisungen mithilfe des Assistenten zum Erstellen von Skripts in SSMS zu erstellen. Führen Sie die gleichen Schritte aus, um Skripts für die verschiedenen Objekttypen zu generieren.

1. Stellen Sie eine Verbindung zu Ihrer vorhandenen SQL-Server-Instance her.
2. Öffnen Sie das Kontextmenü (rechte Maustaste) für einen Datenbanknamen.
3. Wählen Sie Tasks (Aufgaben) und dann Generate Scripts... (Skripts generieren).
4. Klicken Sie im Bereich Choose Objects (Objekte auswählen) auf Select specific database objects (Bestimmte Datenbankobjekte auswählen). Klicken Sie auf Tables (Tabellen) und wählen Sie alle Tabellen aus. Wählen Sie Next (Weiter), um fortzufahren.

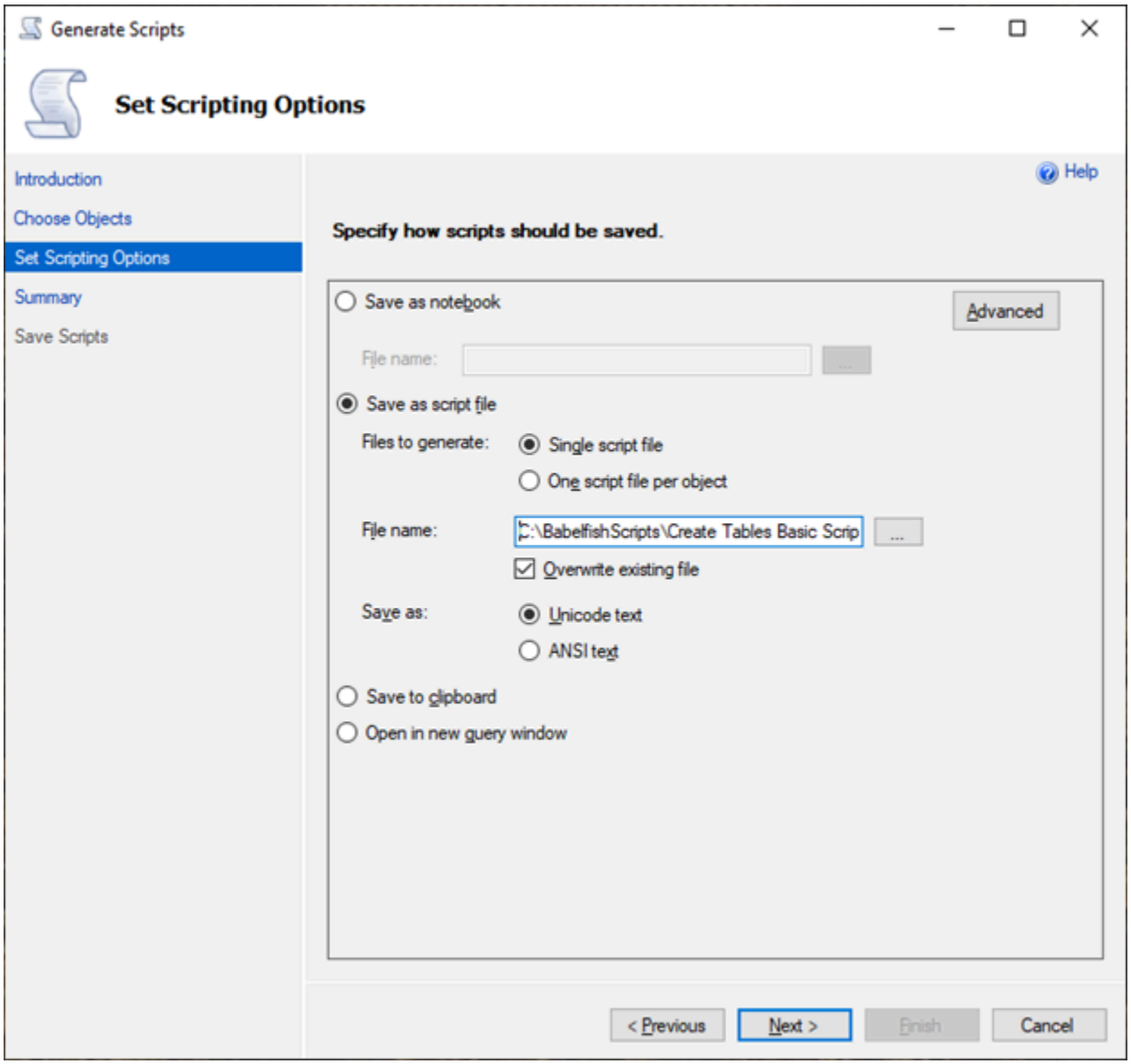


5. Wählen Sie auf der Seite Set Scripting Options (Scripting-Optionen festlegen) die Option Advanced (Erweitert) aus, um die Einstellungen Options (Optionen) zu öffnen. Um die grundlegenden Create-Table-Anweisungen zu generieren, ändern Sie die folgenden Standardwerte:

   - „Script Defaults“ (Skriptstandardwerte) auf „False“ (Falsch).
   - „Script Extended Properties“ (Erweiterte Skripteigenschaften) auf „False“ (Falsch). Babelfish unterstützt keine erweiterten Eigenschaften.
   - „Script Check Constraints“ (Skriptprüfeinschränkungen) auf „False“ (Falsch). „Script Foreign Keys“ (Skriptfremdschlüssel) auf „False“ (Falsch).



6. Wählen Sie OK aus.
7. Wählen Sie auf der Seite Set Scripting Options (Scripting-Optionen festlegen) die Option Save as script file (Als Skriptdatei speichern) und dann die Option Single script file (Einzelne Skriptdatei) aus. Geben Sie unter File name (Dateiname) einen Namen ein.



8. Wählen Sie Next (Weiter) aus, um die Seite Summary wizard (Übersichtsassistent) aufzurufen.
9. Wählen Sie Next (Weiter) aus, um die Skriptgenerierung zu starten.

   Sie können weiterhin Skripts für die anderen Objekttypen im Assistenten generieren. Anstatt nach dem Speichern der Datei Finish (Beenden) auszuwählen, klicken Sie dreimal auf die Schaltfläche Previous (Zurück), um zur Seite Choose Objects (Objekte auswählen)

zurückzukehren. Wiederholen Sie dann die Schritte im Assistenten, um Skripts für die anderen Objekttypen zu generieren.

Datenbankauthentifizierung mit Babelfish für Aurora PostgreSQL

Babelfish für Aurora PostgreSQL unterstützt zwei Möglichkeiten zum Authentifizieren von Datenbankbenutzern. Die Passwort-Authentifizierung ist standardmäßig für alle Babelfish-DB-Cluster verfügbar. Sie können auch Kerberos-Authentifizierung für denselben DB-Cluster hinzufügen.

Themen

- Passwort-Authentifizierung mit Babelfish
- Kerberos-Authentifizierung mit Babelfish
- Einrichten der Kerberos-Authentifizierung mithilfe von Active-Directory-Sicherheitsgruppen für Babelfish

Passwort-Authentifizierung mit Babelfish

Babelfish for Aurora PostgreSQL unterstützt die Passwortauthentifizierung. Passwörter werden verschlüsselt auf der Festplatte gespeichert. Weitere Informationen über die Authentifizierung in einem Aurora-PostgreSQL-Cluster finden Sie unter Sicherheit in Amazon Aurora PostgreSQL.

Sie werden möglicherweise jedes Mal aufgefordert, wenn Sie eine Verbindung zu Babelfish herstellen. Jeder Benutzer, der zu Aurora PostgreSQL migriert oder auf Aurora erstellt wurde, kann dieselben Anmeldeinformationen sowohl am SQL Server-Port als auch am PostgreSQL-Port verwenden. Babelfish erzwingt keine Passwortrichtlinien, aber wir empfehlen Ihnen, Folgendes zu tun:

- Es muss sich um ein komplexes Passwort mit mindestens acht (8) Zeichen handeln.
- Durchsetzen einer Richtlinie zum Ablauf des Kennworts.

Verwenden Sie den Befehl `SELECT * FROM pg_user;`, um eine vollständige Liste der Datenbankbenutzer zu überprüfen.

Kerberos-Authentifizierung mit Babelfish

Die Version 15.2 von Babelfish für Aurora PostgreSQL unterstützt die Authentifizierung bei Ihrem DB-Cluster mithilfe von Kerberos. Diese Methode ermöglicht es Ihnen, die Microsoft Windows-Authentifizierung zum Authentifizieren von Benutzern zu verwenden, wenn diese eine Verbindung mit Ihrer Babelfish-Datenbank herstellen. Dazu müssen Sie Ihren DB-Cluster so konfigurieren, dass AWS Directory Service for Microsoft Active Directory für die Kerberos-Authentifizierung verwendet wird. Weitere Informationen finden Sie unter Was ist Directory Service im AWS Directory Service-Administratorhandbuch.

Einrichten der Kerberos-Authenifizierung

Der DB-Cluster von Babelfish für Aurora PostgreSQL kann eine Verbindung über zwei verschiedene Ports herstellen, die Einrichtung der Kerberos-Authentifizierung ist jedoch ein einmaliger Vorgang. Daher müssen Sie zuerst die Kerberos-Authentifizierung für Ihren DB-Cluster einrichten. Weitere Informationen finden Sie unter Einrichten der Kerberos-Authentifizierung. Stellen Sie nach Abschluss der Einrichtung sicher, dass Sie mithilfe von Kerberos eine Verbindung mit einem PostgreSQL-Client herstellen können. Weitere Informationen finden Sie unter Herstellen einer Verbindung mithilfe der Kerberos-Authentifizierung.

Anmeldung und Benutzerbereitstellung in Babelfish

Windows-Anmeldungen, die über den Tabular Data Stream (TDS)-Port erstellt wurden, können entweder mit dem TDS-Port oder dem PostgreSQL-Port verwendet werden. Zunächst muss die Anmeldung, die Kerberos für die Authentifizierung verwenden kann, vom TDS-Port aus bereitgestellt werden, bevor sie von den T-SQL-Benutzern und -Anwendungen genutzt werden, um eine Verbindung mit einer Babelfish-Datenbank herzustellen. Beim Erstellen von Windows-Anmeldungen können Administratoren die Anmeldung entweder mit dem DNS-Domain-Namen oder dem NetBIOS-Domain-Namen angeben. In der Regel ist die NetBIOS-Domain die Subdomain des DNS-Domain-Namens. Wenn der DNS-Domain-Name beispielsweise `CORP.EXAMPLE.COM` lautet, kann die NetBIOS-Domain `CORP` sein. Wenn das NetBIOS-Domain-Namenformat für eine Anmeldung bereitgestellt wird, muss eine Zuordnung zum DNS-Domain-Namen vorhanden sein.

Verwalten der Zuordnung von NetBIOS-Domain-Namen zum DNS-Domain-Namen

Um Zuordnungen zwischen dem NetBIOS-Domain-Namen und dem DNS-Domain-Namen zu verwalten, bietet Babelfish gespeicherte Systemprozeduren zum Hinzufügen, Entfernen und Kürzen von Zuordnungen. Nur ein Benutzer mit einer `sysadmin`-Rolle kann diese Verfahren ausführen.

Verwenden Sie die von Babelfish bereitgestellte gespeicherte Systemprozedur `babelfish_add_domain_mapping_entry`, um eine Zuordnung zwischen NetBIOS und DNS-Domain-Namen zu erstellen. Beide Argumente müssen einen gültigen Wert haben und dürfen nicht NULL sein.

Example

```
EXEC babelfish_add_domain_mapping_entry 'netbios_domain_name',
 'fully_qualified_domain_name'
```

Das folgende Beispiel zeigt, wie die Zuordnung zwischen dem NetBIOS-Namen CORP und dem DNS-Domain-Namen CORP.EXAMPLE.COM erstellt wird.

Example

```
EXEC babelfish_add_domain_mapping_entry 'corp', 'corp.example.com'
```

Verwenden Sie die gespeicherte Systemprozedur „babelfish_remove_domain_mapping_entry“, um einen vorhandenen Zuordnungseintrag zu löschen.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'netbios_domain_name'
```

Das folgende Beispiel zeigt, wie Sie die Zuordnung für den NetBIOS-Namen CORP entfernen.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'corp'
```

Verwenden Sie die gespeicherte Systemprozedur babelfish_truncate_domain_mapping_table, um alle vorhandenen Zuordnungseinträge zu löschen:

Example

```
EXEC babelfish_truncate_domain_mapping_table
```

Verwenden Sie die folgende Abfrage, um alle Zuordnungen zwischen NetBIOS und dem DNS-Domain-Namen anzuzeigen.

Example

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

Verwalten von Anmeldungen

Erstellen von Anmeldungen

Stellen Sie über den TDS-Endpunkt eine Verbindung mit der DB her, indem Sie eine Anmeldung verwenden, die über die entsprechenden Berechtigungen verfügt. Wenn kein Datenbankbenutzer für die Anmeldung angelegt wurde, wird die Anmeldung dem Gastbenutzer zugeordnet. Wenn der Gastbenutzer nicht aktiviert ist, schlägt der Anmeldeversuch fehl.

Erstellen Sie eine Windows-Anmeldung mit der folgenden Abfrage. Die Option `FROM WINDOWS` ermöglicht die Authentifizierung mit Active Directory.

```
CREATE LOGIN login_name FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Example

Das folgende Beispiel zeigt das Erstellen einer Anmeldung für den Active Directory-Benutzer [corp \test1] mit der Standarddatenbank db1.

```
CREATE LOGIN [corp\test1] FROM WINDOWS WITH DEFAULT_DATABASE=db1
```

In diesem Beispiel wird davon ausgegangen, dass eine Zuordnung zwischen der NetBIOS-Domain CORP und dem DNS-Domain-Namen CORP.EXAMPLE.COM besteht. Wenn es keine Zuordnung gibt, müssen Sie den DNS-Domain-Namen [CORP.EXAMPLE.COM\test1] angeben.

> **Note**
>
> Anmeldungen, die auf Active-Directory-Benutzern basieren, sind auf Namen mit weniger als 21 Zeichen beschränkt.

Löschen von Anmeldung

Um eine Anmeldung zu löschen, verwenden Sie dieselbe Syntax wie für jede beliebige Anmeldung, wie im folgenden Beispiel gezeigt:

```
DROP LOGIN [DNS domain name\login]
```

Ändern von Anmeldungen

Wenn Sie eine Anmeldung ändern möchten, verwenden Sie dieselbe Syntax wie für jede beliebige Anmeldung, wie im folgenden Beispiel gezeigt:

```
ALTER LOGIN [DNS domain name\login] { ENABLE|DISABLE|WITH DEFAULT_DATABASE=[master] }
```

Der Befehl ALTER LOGIN unterstützt eingeschränkte Optionen für Windows-Anmeldungen, darunter folgende:

- DISABLE – zum Deaktivieren einer Anmeldung. Sie können eine deaktivierte Anmeldung nicht für die Authentifizierung verwenden.
- ENABLE – zum Aktivieren einer deaktivierten Anmeldung.
- DEFAULT_DATABASE – zum Ändern der Standarddatenbank einer Anmeldung.

> **Note**
>
> Die gesamte Passwortverwaltung erfolgt über Directory Service, sodass der Befehl ALTER LOGIN Datenbankadministratoren nicht erlaubt, Passwörter für Windows-Logins zu ändern oder festzulegen.

Herstellen einer Verbindung mit Babelfish für Aurora PostgreSQL mit Kerberos-Authentifizierung

Normalerweise authentifizieren sich die Datenbankbenutzer, die die Kerberos-Authentifizierung nutzen, über ihren Client-Computer. Diese Computer sind Mitglieder der Active-Directory-Domain. Sie verwenden die Windows-Authentifizierung von ihren Client-Anwendungen aus, um auf den Server von Babelfish für Aurora PostgreSQL am TDS-Port zuzugreifen.

Herstellen einer Verbindung mit Babelfish für Aurora PostgreSQL am PostgreSQL-Port mit Kerberos-Authentifizierung

Sie können Anmeldungen, die über den TDS-Port erstellt wurden, entweder mit dem TDS-Port oder dem PostgreSQL-Port verwenden. PostgreSQL verwendet für Benutzernamen jedoch standardmäßig Vergleiche unter Berücksichtigung der Groß- und Kleinschreibung. Damit Aurora PostgreSQL

Kerberos-Benutzernamen ohne Berücksichtigung von Groß- und Kleinschreibung interpretiert, müssen Sie den Parameter `krb_caseins_users` in der benutzerdefinierten Babelfish-Cluster-Parametergruppe auf `true` festlegen. Dieser Parameter ist standardmäßig auf `false` festgelegt. Weitere Informationen finden Sie unter Konfigurieren von Benutzernamen, bei denen die Groß-/Kleinschreibung unterschieden wird. Darüber hinaus müssen Sie den Anmeldebenutzernamen im Format <login@DNS domain name> über die PostgreSQL-Client-Anwendungen angeben. Das Format <DNS domain name\login> kann nicht verwendet werden.

Häufig auftretende Fehler

Sie können Gesamtstruktur-Vertrauensstellungen zwischen Ihrem On-Premises Microsoft Active Directory und dem AWS Managed Microsoft AD konfigurieren. Weitere Informationen finden Sie unter Erstellen einer Vertrauensstellung. Anschließend müssen Sie eine Verbindung über einen speziellen Domain-spezifischen Endpunkt herstellen, anstatt die Amazon-Domain `rds.amazonaws.com` im Host-Endpunkt zu verwenden. Wenn Sie nicht den richtigen Domain-spezifischen Endpunkt verwenden, wird möglicherweise folgende Fehlermeldung angezeigt:

```
Error: “Authentication method "NTLMSSP" not supported (Microsoft SQL Server, Error:
 514)"
```

Dieser Fehler tritt auf, wenn der TDS-Client das Serviceticket für die angegebene Endpunkt-URL nicht zwischenspeichern kann. Weitere Informationen finden Sie unter Herstellen einer Verbindung mithilfe der Kerberos-Authentifizierung.

Einrichten der Kerberos-Authentifizierung mithilfe von Active-Directory-Sicherheitsgruppen für Babelfish

Ab Babelfish-Version 4.2.0 können Sie die Kerberos-Authentifizierung für Babelfish mit Active-Directory-Sicherheitsgruppen einrichten. Die folgenden Voraussetzungen müssen erfüllt sein, um die Kerberos-Authentifizierung mithilfe von Active Directory einzurichten:

- Sie müssen alle unter Kerberos-Authentifizierung mit Babelfish genannten Schritte ausführen.
- Stellen Sie sicher, dass die DB-Instance mit Active Directory verknüpft ist. Zur Bestätigung können Sie den Status der Domainmitgliedschaft in der Konsole oder durch Ausführen des AWS-CLI-Befehls describe-db-instances anzeigen.

  Der Status der DB-Instance sollte kerberos-enabled lauten. Weitere Informationen zum Verständnis der Domainmitgliedschaft finden Sie unter Grundlegendes zur Domänenmitgliedschaft.

- Überprüfen Sie die Zuordnungen zwischen dem NetBIOS-Domainnamen und dem DNS-Domainnamen mithilfe der folgenden Abfrage:

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

- Bevor Sie fortfahren, überprüfen Sie, ob die Kerberos-Authentifizierung mit individueller Anmeldung erwartungsgemäß funktioniert. Die Verbindung mit der Kerberos-Authentifizierung als Active-Directory-Benutzer sollte erfolgreich sein. Falls Sie auf Probleme stoßen, finden Sie weitere Informationen unter Häufig auftretende Fehler.

Einrichten der pg_ad_mapping-Erweiterung

Sie müssen alle unter Einrichten der pg_ad_mapping-Erweiterung genannten Schritte ausführen. Führen Sie die folgende Abfrage über den TDS-Endpunkt aus, um zu überprüfen, ob die Erweiterung installiert wurde:

```
1> SELECT extname, extversion FROM pg_extension where extname like 'pg_ad_mapping';
2> GO
extname       extversion
------------- ----------
pg_ad_mapping 0.1

(1 rows affected)
```

Verwalten von Gruppenanmeldungen

Erstellen Sie Gruppenanmeldungen, indem Sie die unter Verwalten von Anmeldungen genannten Schritte ausführen. Wir empfehlen, dass der Anmeldename mit dem Namen der Active Directory (AD)-Sicherheitsgruppe identisch ist, um die Wartung zu vereinfachen, obwohl dies nicht verpflichtend ist. Zum Beispiel:

```
CREATE LOGIN [corp\accounts-group] FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Zuordnen von T-SQL-Gruppenanmeldungen zur AD-Sicherheitsgruppe

Sie müssen explizit eine T-SQL-Windows-Gruppenanmeldung für jede AD-Sicherheitsgruppe bereitstellen, die Zugriff auf den Datenbankserver benötigt. Ein AD-Benutzer, der Teil mindestens einer bereitgestellten AD-Sicherheitsgruppe ist, erhält Zugriff auf den Datenbankserver.

> **Note**
>
> Diese T-SQL-Anmeldung kann nicht mehr über eine passwortbasierte Authentifizierung erfolgen.

accounts-group ist beispielsweise eine Sicherheitsgruppe in AD. Wenn Sie diese Sicherheitsgruppe in Babelfish bereitstellen möchten, müssen Sie das Format [corp\accounts-group] verwenden.

- AD-Sicherheitsgruppe: accounts-group
- TSQL-Anmeldung: [corp\accounts-group]
- Entsprechende PG-Rolle für die angegebene TSQL-Anmeldung: accounts-group@CORP.EXAMPLE.COM

Der Administrator kann nun mit dem Erstellen der Zuordnung zwischen der AD-Sicherheitsgruppe und der T-SQL-Anmeldung vom PostgreSQL-Endpunkt aus über den folgenden psql-Befehl fortfahren. Weitere Informationen zur Funktionsverwendung finden Sie unter Verwenden von Funktionen der `pg_ad_mapping`-Erweiterung.

> **Note**
>
> Die T-SQL-Anmeldung sollte beim Hinzufügen der Zuordnung im Format login_name@FQDN angegeben werden. Gewichtungen werden ignoriert, wenn eine Verbindung über den TDS-Endpunkt hergestellt wird. Weitere Informationen zur Verwendung von Gewichtungen finden Sie unter Herstellen einer Verbindung zu Babelfish über den PostgreSQL-Endpunkt auf dem PostgreSQL-Port.

```
postgres=>select pgadmap_set_mapping('accounts-group', 'accounts-
group@CORP.EXAMPLE.COM', <SID>, <Weight>);
```

Informationen zum Abrufen der SID der AD-Sicherheitsgruppe finden Sie unter Abrufen der SID der Active-Directory-Gruppe in PowerShell.

Die folgende Tabelle zeigt ein Beispiel einer Zuordnung von AD-Sicherheitsgruppen zu T-SQL-Anmeldungen:

AD-Sicherheitsgruppe	TSQL-Anmeldungen	Entsprechende PG-Rolle für die angegebene TSQL-Anmeldung	Gewicht
accounts-group	[corp\accounts-group]	accounts-group@CORP.EXAMPLE.COM	7
sales-group	[corp\sales-group]	sales-group@CORP.EXAMPLE.COM	10
dev-group	[corp\dev-group]	dev-group@CORP.EXAMPLE.COM	7

```
postgres=> select admap.ad_sid, admap.ad_grp, lgn.orig_loginname, lgn.rolname,
 admap.weight from pgadmap_read_mapping() as admap, sys.babelfish_authid_login_ext as
 lgn where admap.pg_role = lgn.rolname;
    ad_sid    |     ad_grp     |    orig_loginname   |            rolname
 | weight
--------------+----------------+---------------------+--------------------------------
+--------
 S-1-5-67-890 | accounts-group | corp\accounts-group  | accounts-group@CORP.EXAMPLE.COM
 |     7
 S-1-2-34-560 | sales-group    | corp\sales-group     | sales-group@CORP.EXAMPLE.COM
 |     10
 S-1-8-43-612 | dev-group      | corp\dev-group       | dev-group@CORP.EXAMPLE.COM
 |     7
 (7 rows)
```

Herstellen einer Verbindung zu Babelfish über den TDS-Endpunkt

Im folgenden Beispiel ist user1 Mitglied von accounts-group und sales-group, user2 ist Mitglied von accounts-group und dev-group.

Username	AD-Sicherheitsgruppenmitgliedschaft
Benutzer 1	accounts-group, sales-group
user2	accounts-group, dev-group

Stellen Sie mit dem sqlcmd-Dienstprogramm eine Verbindung zum Babelfish-Datenbankserver her. Sie können überprüfen, ob ein Benutzer (in diesem Beispiel user1) mit Kerberos authentifiziert wurde, indem Sie dem folgenden Beispiel folgen:

```
1> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();
2>  GO
principal               gss_authenticated
----------------------  -----------------
user1@CORP.EXAMPLE.COM  1

((1 rows affected))
1> select suser_name();
2>  GO
suser_name
----------
corp\user1

(1 rows affected)
```

In diesem Beispiel erbt user1 die Berechtigungen von accounts-group und sales-group. Sie können die Gruppenmitgliedschaft mithilfe der Systemansicht `sys.login_token` überprüfen.

```
1> SELECT name, type FROM sys.login_token;
2>  GO
name                type
------------------- ----
corp\accounts-group WINDOWS GROUP
corp\sales-group    WINDOWS GROUP

(2 rows affected)
```

Prüfen und Protokollieren

Verwenden Sie die folgenden Befehle, um die Identität des AD-Sicherheitsprinzipals zu ermitteln:

```
1> select suser_name();
2> GO
suser_name
----------
corp\user1

(1 rows affected)
```

Derzeit ist die AD-Benutzeridentität in den Protokollen nicht sichtbar. Sie können den Parameter `log_connections` aktivieren, um die Einrichtung einer DB-Sitzung zu protokollieren. Weitere Informationen finden Sie unter log_connections. Die Ausgabe umfasst die AD-Benutzeridentität als Prinzipal, wie im folgenden Beispiel gezeigt. Die dieser Ausgabe zugeordnete Backend-PID kann dann dabei helfen, Aktionen dem tatsächlichen AD-Benutzer zuzuordnen.

```
bbf_group_ad_login@babelfish_db:[615]:LOG: connection authorized:
 user=bbf_group_ad_login database=babelfish_db application_name=sqlcmd GSS
 (authenticated=yes, encrypted=yes, principal=user1@CORP.EXAMPLE.COM)
```

Verwenden der Berechtigungen aus der AD-Sicherheitsgruppenmitgliedschaft

Übernehmen von Berechtigungen auf Serverebene

AD-Benutzer, die Mitglieder der angegebenen AD-Sicherheitsgruppe sind, übernehmen die Berechtigungen auf Serverebene, die der zugeordneten Windows-Gruppenanmeldung erteilt wurden. Denken Sie zum Beispiel an die `accounts-group`-AD-Sicherheitsgruppe, der die Mitgliedschaft in der `sysadmin`-Serverrolle in Babelfish erteilt wird. Sie können die Berechtigungen auf Serverebene mit dem folgenden Befehl übernehmen:

```
1> ALTER SERVER ROLE sysadmin ADD MEMBER [corp\accounts-group];
```

Folglich übernimmt jeder Active-Directory-Benutzer, der Mitglied der `accounts-group`-AD-Sicherheitsgruppe ist, die mit der `sysadmin`-Rolle verbundenen Berechtigungen auf Serverebene.

Das bedeutet, dass ein Benutzer wie `corp\user1`, der Mitglied von `accounts-group` ist, nun in Babelfish Operationen auf Serverebene ausführen kann.

> **Note**
>
> Zum Ausführen von DDLs auf Serverebene muss eine Windows-Anmeldung für den einzelnen AD-Benutzer vorhanden sein. Weitere Informationen finden Sie unter Einschränkungen.

Übernehmen von Berechtigungen auf Datenbankebene

Zum Erteilen von Berechtigungen auf Datenbankebene muss ein Datenbankbenutzer erstellt und ihm eine Windows-Gruppenanmeldung zugeordnet werden. AD-Benutzer, die Mitglieder einer angegebenen AD-Sicherheitsgruppe sind, übernehmen die Berechtigungen auf Datenbankebene, die diesem Datenbankbenutzer erteilt wurden. Im folgenden Beispiel können Sie sehen, wie Berechtigungen auf Datenbankebene für die Windows-Gruppe [corp\accounts-group] zugewiesen werden.

```
1> CREATE DATABASE db1;
2> GO
1> USE db1;
2> GO
Changed database context to 'db1'.
1> CREATE TABLE dbo.t1(a int);
2> GO
```

Erstellen Sie einen Datenbankbenutzer [corp\sales-group] für die Windows-Gruppenanmeldung [corp\accounts-group]. Zum Ausführen dieses Schrittes stellen Sie eine Verbindung über den TDS-Endpunkt her. Verwenden Sie dazu die Anmeldung, die Mitglied von sysadmin ist.

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group];
2> GO
```

Stellen Sie nun eine Verbindung als AD-Benutzer user1 her, um den Zugriff auf die Tabelle t1 zu überprüfen. Da wir die Berechtigungen auf Datenbankebene noch nicht erteilt haben, wird dies zu dem Fehler „Zugriff verweigert“ führen.

```
1> SELECT * FROM dbo.t1;
2> GO
Msg 33557097, Level 16, State 1, Server db-inst, Line 1
permission denied for table t1
```

Erteilen Sie dem Datenbankbenutzer [corp\accounts-group] SELECT für die Tabelle t1. Zum Ausführen dieses Schrittes stellen Sie eine Verbindung über den TDS-Endpunkt her. Verwenden Sie dazu die Anmeldung, die Mitglied von sysadmin ist.

```
1> GRANT SELECT ON dbo.t1 TO [corp\accounts-group];
2> GO
```

Stellen Sie eine Verbindung als AD-Benutzer user1 her, um den Zugriff zu validieren.

```
1> SELECT * FROM dbo.t1;
2> GO
a
-----------

(0 rows affected)
```

Umgang mit dem Verhalten von DDL-Anweisungen auf der Grundlage eines standardmäßigen oder expliziten Schemas

Bei Verwendung einer AD-authentifizierten Sitzung wird das Standardschema für die aktuelle Sitzung durch die folgenden Bedingungen bestimmt:

- Wenn ein einzelner Datenbankbenutzer existiert, wird das Standardschema des Benutzers als Standardschema der aktuellen Sitzung betrachtet.

- Wenn das Standardschema für einen Gruppendatenbankbenutzer existiert, wird das Standardschema des Gruppendatenbankbenutzers als das Standardschema der aktuellen Sitzung mit der kleinsten Prinzipal-ID betrachtet.

Verstehen des Verhaltens der CREATE DDL-Anweisung

Wenn in der CREATE DDL-Anweisung kein explizites Schema angegeben ist, erfolgt die Objekterstellung im Standardschema der aktuellen Sitzung. Wenn nicht bestimmt werden kann, ob das Schema standardmäßig oder explizit ist, gibt die DDL-Anweisung den folgenden Fehler aus:

```
"Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command."
```

Example : Ein Standardschema für Windows-Gruppenbenutzer ist nicht vorhanden.

Der Windows-Gruppenbenutzer [corp\accounts-group] hat ein NULL-Standardschema und der AD-Benutzer user1 versucht, die DDL auszuführen, ohne das Schema explizit anzugeben. Da für user1 keine individuelle Windows-Anmeldung und kein Benutzer existiert, erhält dieser Benutzer nur die Datenbankberechtigungen des Windows-Gruppenbenutzers [corp\accounts-group].

```
1> create TABLE t2(a int);
2> GO

Msg 33557097, Level 16, State 1, Server db-inst, Line 1
Babelfish Unsupported Command : Schema required for CREATE DDLs when connecting with
 Active Directory Group authentication. Assign default schema to group user or specify
 schema in command.
```

> **Note**
>
> Für den AD-Benutzer user1 existiert keine individuelle Windows-Anmeldung und kein Benutzer.

Example : Ein Standardschema für Windows-Gruppenbenutzer ist vorhanden.

Erstellen Sie mithilfe von sysadmin einen Windows-Gruppenbenutzer für die [corp\accounts-group]-Anmeldung mit dem Standardschema.

```
1> CREATE USER [corp\accounts-group] FOR LOGIN [corp\accounts-group] WITH
 DEFAULT_SCHEMA = sch_acc;
2> GO
1> CREATE SCHEMA sch_acc AUTHORIZATION [gad\accounts-group];
2> GO
1> SELECT name, principal_id, default_schema_name FROM sys.database_principals WHERE
 name = 'corp\accounts-group';
2> GO

name               principal_id default_schema_name
------------------ ------------ -------------------
corp\accounts-group 24162        sch_acc

(1 rows affected)
```

Versuchen Sie, mithilfe des AD-Benutzers user1 ein Objekt zu erstellen, ohne das Schema explizit anzugeben. Die Tabelle t2 wird im Standardschema des Windows-Gruppenbenutzers [corp\accounts-group] erstellt. Der Eigentümer dieses Objekts ist derselbe wie der Eigentümer des Schemas sch_acc.

```
1> CREATE TABLE t_group(a int);
2> GO
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't_group';
2> GO

name    schema_name
------- -----------
t_group sch_acc

(1 rows affected)
```

> **Note**
>
> Für den AD-Benutzer user1 existiert keine individuelle Windows-Anmeldung und kein Benutzer.

Example : Ein einzelner Datenbankbenutzer ist auch für einen AD-Benutzer vorhanden.

Wenn für einen AD-Benutzer auch ein einzelner Datenbankbenutzer existiert, werden die Objekte immer in dem Schema erstellt, das dem einzelnen Datenbankbenutzer zugeordnet ist. Wenn das Schema für den Datenbankbenutzer nicht existiert, wird das dbo-Schema verwendet. Erstellen Sie eine individuelle Windows-Anmeldung und einen Datenbankbenutzer für den AD-Benutzer user1. Stellen Sie mithilfe einer sysadmin-Anmeldung eine Verbindung über den TDS-Endpunkt her.

```
1> CREATE LOGIN [corp\user1] FROM WINDOWS;
2> GO
1> CREATE USER [corp\user1] FOR LOGIN [corp\user1] WITH DEFAULT_SCHEMA = sch1;
2> GO
1> CREATE SCHEMA sch1 AUTHORIZATION [corp\user1];
2> GO
1> SELECT name, default_schema_name FROM sys.database_principals WHERE name = 'corp
\user1';
2> GO

name      default_schema_name
--------- -------------------
corp\user1 sch1

(1 rows affected)
```

Stellen Sie mit dem AD-Benutzer user1 eine Verbindung her und versuchen Sie, ein Objekt zu erstellen, ohne das Schema explizit anzugeben. Die Tabelle t2 wird im Schema sch1 erstellt. Beachten Sie auch, dass der Eigentümer dieses Objekts mit dem Eigentümer des Schemas sch1 identisch sein wird.

```
1> CREATE TABLE t2(a int);
2> GO
```

```
1> SELECT name, schema_name(schema_id) FROM sys.objects WHERE name like 't2';
2> GO

name schema_name
---- -----------
t2   sch1

(1 rows affected)
```

Einschränkungen

- Das Dienstprogramm für Dump/Wiederherstellung unterstützt das Dumping der pg_ad_mapping-Erweiterungszuordnungen nicht. Sie müssen diese Zuordnungen nach der Wiederherstellung neu erstellen.
- Die Blau/Grün-Bereitstellung wird für Babelfish- und Aurora-PostgreSQL-Instances mit `pg_ad_mapping` nicht unterstützt.
- Die implizite Schemaerstellung wird nicht unterstützt. DDL-Anweisungen, die eine implizite Schemaerstellung erfordern, werden nicht unterstützt.
- Die DDLs auf Serverebene ALTER AUTHORIZATION ON DATABASE, CREATE DATABASE, CREATE LOGIN, ALTER LOGIN, ALTER SERVER ROLE, ALTER DATABASE werden in einer Gruppen-AD-authentifizierten Sitzung nicht unterstützt, wenn keine individuelle Windows-Anmeldung, sondern nur eine Windows-Gruppenanmeldung vorhanden ist. Zur Umgehung dieser Einschränkung wird empfohlen, diese Operationen in einer passwortauthentifizierten Sitzung durchzuführen oder eine individuelle Windows-Anmeldung zu erstellen.
- Die implizite Benutzererstellung wird nicht unterstützt. Ideales T-SQL-Verhalten [in Babelfish noch nicht unterstützt]; In einigen Fällen wie DDL- und Zugriffskontrollanweisungen wie GRANT/REVOKE, bei denen der AD-Benutzername im Befehl angegeben ist, aber nicht in der Datenbank existiert, werden Datenbankbenutzer, die als AD-Benutzer benannt sind, implizit erstellt.
- Für DDLs in PL/pgSQL-Prozeduren oder -Funktionen, die über den PSQL-Endpunkt erstellt und über den TDS-Endpunkt in einer authentifizierten Gruppen-AD-Sitzung ausgeführt werden:
  - ALTER/DROP-Anweisungen werden unterstützt.
  - CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE FUNCTION/PROC, CREATE TYPE, CREATE SEQUENCE, CREATE TRIGGER, SELECT INTO, CREATE FULLTEXT INDEX, CREATE UNIQUE INDEX geben einen Fehler aus, wenn das Schema nicht explizit angegeben wird und das Standardschema für die aktuelle Sitzung Null ist.

  - CREATE DATABASE, CREATE EXTENSION und alle anderen CREATE-Anweisungen für PG-spezifische Objekte (nicht in T-SQL), CREATE subscription, CREATE tablespace, CREATE policy und CREATE conversion werden nicht unterstützt.
- DDLs über den PostgreSQL-Endpunkt werden in einer authentifizierten Gruppen-AD-Sitzung nicht unterstützt. Zur Umgehung dieses Problems können Sie jederzeit eine Verbindung über den Master-Benutzer oder einen anderen Benutzer mithilfe eines passwortbasierten Authentifizierungsmechanismus herstellen.
- Für Systemobjekte wie SUSER_SID(), IS_SRVROLEMEMBER(), IS_MEMBER(), sys.dm_exec_sessions gelten die folgenden Einschränkungen.
  - SUSER_SID() gibt die SID nicht zurück, wenn ein AD-Benutzer oder eine AD-Sicherheitsgruppe angegeben wird.
  - IS_SRVROLEMEMBER() berücksichtigt die Rollenmitgliedschaft nicht, wenn der aktuelle AD-Benutzer die Serverrollenmitgliedschaft von einer Windows-Gruppenanmeldung übernimmt.
  - IS_MEMBER() gibt für alle Abfragen im Zusammenhang mit Windows-Gruppen den Wert false zurück.
  - sys.dm_exec_sessions zeigt keine erwarteten Werte für die Spalten login_name und nt_user_name an.

Herstellen einer Verbindung zu Babelfish über den PostgreSQL-Endpunkt auf dem PostgreSQL-Port

Sie können Gruppenanmeldungen, die über den TDS-Port erstellt wurden, auch verwenden, um eine Verbindung über den PostgreSQL-Port herzustellen. Zum Herstellen einer Verbindung über den PostgreSQL-Port müssen Sie den Namen des AD-Benutzers im Format `<ad_username@FQDN>` in den PostgreSQL-Clientanwendungen angeben. Sie können das Format `<DNS domain name \ad_username>` nicht verwenden.

PostgreSQL verwendet für Benutzernamen standardmäßig Vergleiche unter Berücksichtigung der Groß- und Kleinschreibung. Damit Aurora PostgreSQL Kerberos-Benutzernamen ohne Berücksichtigung von Groß- und Kleinschreibung interpretiert, müssen Sie den Parameter krb_caseins_users in der benutzerdefinierten Babelfish-Cluster-Parametergruppe auf true festlegen. Dieser Parameter ist standardmäßig auf false festgelegt. Weitere Informationen finden Sie unter Konfigurieren Ihres DB-Clusters von Aurora PostgreSQL für Benutzernamen, bei denen die Groß-/Kleinschreibung unterschieden wird.

Verhaltensunterschiede zwischen T-SQL- und PostgreSQL-Endpunkten, wenn ein AD-Benutzer Teil mehrerer Gruppen ist

Bedenken Sie, dass der AD-Benutzer user1 Teil von zwei AD-Sicherheitsgruppen [corp\accounts-group] und [corp\sales-group] ist und dass der Datenbankadministrator die Benutzerzuordnung wie folgt festgelegt hat.

```
postgres=> select * from pgadmap_read_mapping();

ad_sid       | pg_role                          | weight | ad_grp
-------------+----------------------------------+--------+---------------
S-1-5-67-980 | accounts-group@CORP.EXAMPLE.COM | 7      | accounts-group
S-1-2-34-560 | sales-group@CORP.EXAMPLE.COM    | 10     | sales-group
(2 rows)
```

Wenn der Benutzer vom T-SQL-Endpunkt aus eine Verbindung herstellt, erbt er während der Autorisierung die Berechtigungen aller zugehörigen T-SQL-Anmeldungen. In diesem Beispiel erbt user1 die Vereinigung der Berechtigungen aus beiden T-SQL-Gruppenanmeldungen, wobei Gewichtungen ignoriert werden. Dies entspricht dem Standardverhalten von T-SQL.

Wenn derselbe Benutzer jedoch vom PostgreSQL-Endpunkt aus eine Verbindung herstellt, kann er Berechtigungen von nur einer zugehörigen T-SQL-Anmeldung mit der höchsten Gewichtung erben. Wenn den beiden T-SQL-Gruppenanmeldungen dieselbe Gewichtung zugewiesen wurde, erbt der AD-Benutzer die Berechtigungen der T-SQL-Anmeldung, die der zuletzt hinzugefügten Zuordnung entspricht. Für PostgreSQL wird empfohlen, Gewichtungen anzugeben, die den relativen Berechtigungen der einzelnen DB-Rollen entsprechen, um eine Mehrdeutigkeit zu vermeiden. Im folgenden Beispiel hat user1 über den PSQL-Endpunkt eine Verbindung hergestellt und nur die Berechtigungen für sales-groups geerbt.

```
babelfish_db=> select session_user, current_user;

   session_user               |   current_user
------------------------------+--------------------------
 sales-group@CORP.EXAMPLE.COM | sales-group@CORP.EXAMPLE.COM
(1 row)
```

```
babelfish_db=> select principal, gss_authenticated from pg_stat_gssapi where pid =
 pg_backend_pid();

     principal          | gss_authenticated
------------------------+-------------------
 user1@CORP.EXAMPLE.COM | t
(1 row)
```

Verbinden mit einem Babelfish-DB-Cluster

Zum Herstellen einer Verbindung mit Babelfish verbinden Sie sich mit dem Endpunkt des Aurora-PostgreSQL-Clusters, auf dem Babelfish ausgeführt wird. Ihr Client kann einen der folgenden Clienttreiber verwenden, die mit TDS Version 7.1 bis 7.4 kompatibel sind:

- Upgrades für Open Database Connectivity (ODBC)-Treiber
- OLE DB Treiber/msoledBSQL
- Java Database Connectivity (JDBC) Version 8.2.2 (mssql-jdbc-8.2.2) und höher
- Microsoft SqlClient-Datenprovider für SQL Server
- .NET-Datenprovider für SQL Server
- SQL Server Native Client 11.0 (veraltet)
- OLE DB Provider/SQLOLEDB (veraltet)

Mit Babelfish führen Sie Folgendes aus:

- SQL Server-Tools, Anwendungen und Syntax auf dem TDS-Port, standardmäßig Port 1433.
- PostgreSQL-Tools, Anwendungen und Syntax auf dem PostgreSQL-Port standardmäßig Port 5432.

Weitere Informationen zum Herstellen einer Verbindung mit Aurora PostgreSQL im Allgemeinen finden Sie unter Herstellen einer Verbindung mit einem Amazon-Aurora-PostgreSQL-DB-Cluster.

> **Note**
>
> Entwicklertools von Drittanbietern, die den OLEDB-Anbieter des SQL Servers für den Zugriff auf Metadaten verwenden, werden nicht unterstützt. Wir empfehlen Ihnen, für diese Tools JDBC-, ODBC- oder SQL-Native-Clientverbindungen des SQL Servers zu verwenden.

Ab Babelfish-Version 5.1.0 wird die durchgängige Verbindungsverschlüsselung standardmäßig erzwungen. So gewährleisten Sie eine kontinuierliche Konnektivität:

- Konfigurieren Sie die SSL/TLS-Verschlüsselung für Ihre Verbindungen. Weitere Informationen finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

- Importieren Sie die erforderlichen Zertifikate auf Ihre Client-Computer. Weitere Informationen finden Sie unter Verwenden von SSL mit einer DB-Instance von Microsoft SQL Server.

Wenn Sie weiterhin die Verschlüsselungseinstellungen einer früheren Version von Babelfish (vor Version 5.1.0) verwenden möchten, können Sie den Parameter `rds.force_ssl` in Ihrer DB-Cluster-Parametergruppe auf `0` setzen.

Themen

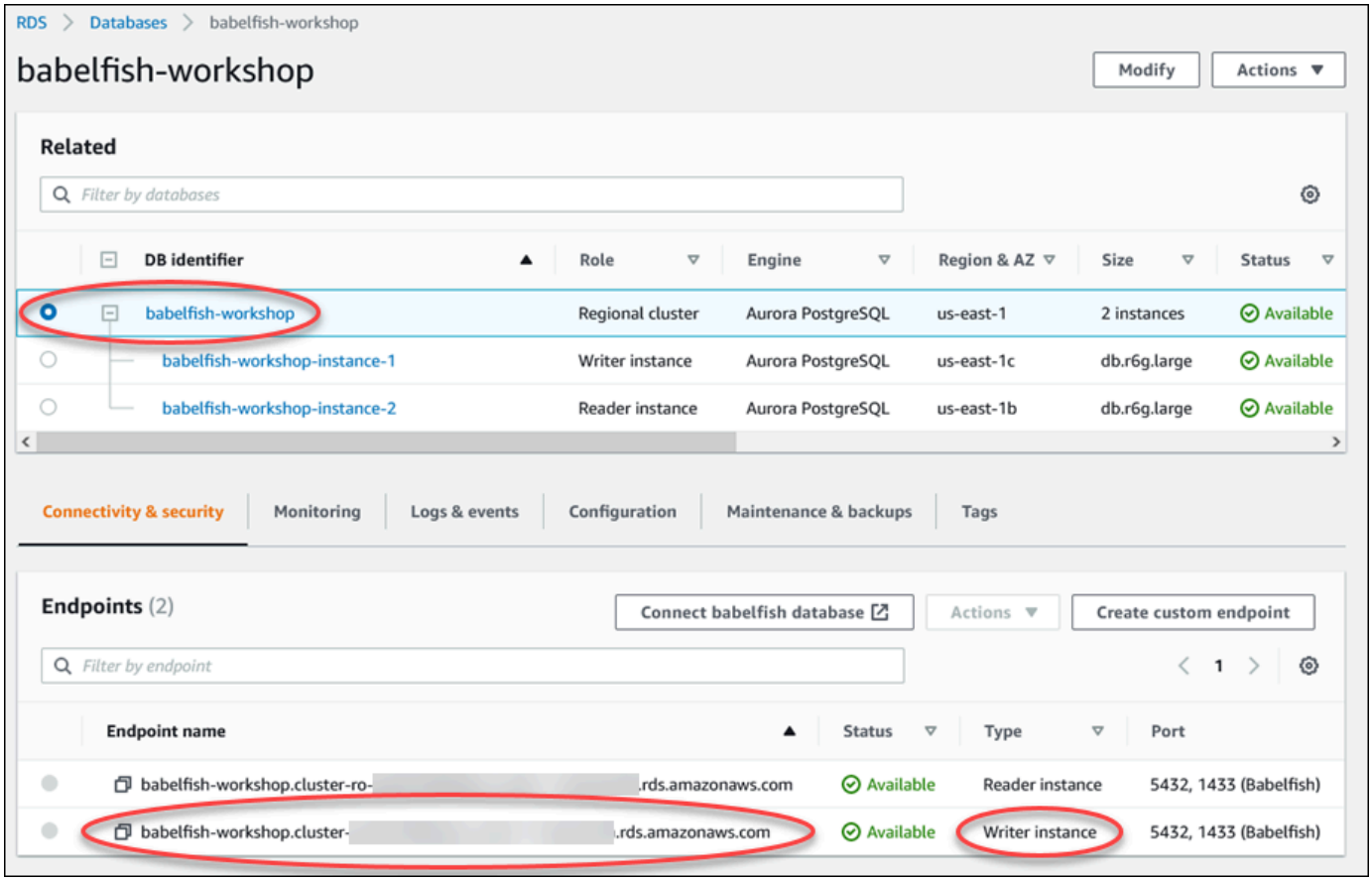
- Finden des Schreiber-Endpunkts und der Portnummer
- Herstellen von C#- oder JDBC-Clientverbindungen mit Babelfish
- Verbinden mit Ihrem DB-Cluster mithilfe eines SQL Server-Clients
- Verbinden mit Ihrem DB-Cluster mit einem PostgreSQL-Client

Finden des Schreiber-Endpunkts und der Portnummer

Zum Herstellen einer Verbindung mit Ihrem Babelfish-DB-Cluster verwenden Sie den Endpunkt, der mit der (primären) Writer-Instance des DB-Clusters verknüpft ist. Die Instance muss den Status Available (Verfügbar) haben. Es kann bis zu 20 Minuten dauern, bis die Instances verfügbar sind, nachdem Sie den DB-Cluster von Babelfish for Aurora PostgreSQL erstellt haben.

So finden Sie Ihren Datenbank-Endpunkt

1. Öffnen Sie die -Konsole für Babelfish.
2. Wählen Sie aus dem Navigationsbereich Datenbanken aus.
3. Wählen Sie Ihren DB-Cluster von Babelfish for Aurora PostgreSQL in der Liste aus, um die Details anzuzeigen.
4. Auf Konnektivität & Sicherheit, notieren Sie sich den verfügbaren Cluster Endpunkte-Werte. Verwenden Sie den Cluster-Endpunkt für die Schreib-Instance in Ihrer Verbindung für Anwendungen, die Lese- oder Schreibvorgänge ausführen.



Weitere Informationen über Aurora-DB-Cluster finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Herstellen von C#- oder JDBC-Clientverbindungen mit Babelfish

Im Folgenden finden Sie einige Beispiele für die Verwendung von C#- und JDBC-Klassen zum Verbinden mit Babelfish for Aurora PostgreSQL.

Example : Verwenden von C#-Code zum Herstellen einer Verbindung mit einem DB-Cluster

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionString = connectionString + ";ServerCertificate=/path/to/certificate.pem";


SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example : Verwenden generischer JDBC-API-Klassen und -Schnittstellen zum Herstellen einer Verbindung mit einem DB-Cluster

```
String dbServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// [optional] To validate server certificate during TLS/SSL connection
connectionUrl = connectionUrl + ";serverCertificate=/path/to/certificate.pem";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example : Verwenden von SQL-Server-spezifischen JDBC-Klassen und -Schnittstellen zum Herstellung einer Verbindung mit einem DB-Cluster

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
```

```
ds.setPassword("password");
String babelfishServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";

ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");

// [optional] To validate server certificate during TLS/SSL connection
ds.setServerCertificate("/path/to/certificate.pem");

Connection con = ds.getConnection();
```

Important

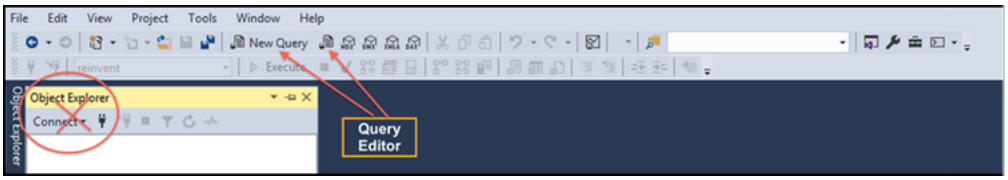
Stellen Sie sicher, dass das Zertifikat mit der Zertifizierungsstelle übereinstimmt, die in Ihrer DB-Cluster-Konfiguration in der AWS-Managementkonsole angezeigt wird.

Verbinden mit Ihrem DB-Cluster mithilfe eines SQL Server-Clients

Sie können einen SQL Server-Client verwenden, um sich mit Babelfish am TDS-Port zu verbinden. Ab Babelfish 2.1.0 und in höheren Versionen können Sie den SSMS Object Explorer oder den SSMS-Abfrage-Editor verwenden, um eine Verbindung mit Ihrem Babelfish-Cluster herzustellen.

Einschränkungen

- In Babelfish 2.1.0 und älteren Versionen, funktioniert `PARSE` zum Prüfen der SQL-Syntax nicht so, wie es sollte. Anstatt die Syntax zu überprüfen, ohne die Abfrage auszuführen, führt der Befehl `PARSE` die Abfrage aus, zeigt aber keine Ergebnisse an. Bei der Verwendung der SMSS-Tastenkombination <Strg><F5> zur Überprüfung der Syntax tritt das gleiche anomale Verhalten auf, d. h., Babelfish führt die Abfrage unerwartet aus, ohne eine Ausgabe zu liefern.
- Babelfish unterstützt MARS (Multiple Active Result Sets) nicht. Stellen Sie sicher, dass alle Clientanwendungen, die Sie für die Verbindung mit Babelfish verwenden, nicht auf MARS eingestellt sind.
- Mit Babelfish 1.3.0 und älteren Versionen wird nur der Abfrage-Editor für SSMS unterstützt. Um SSMS mit Babelfish zu verwenden, öffnen Sie unbedingt das Verbindungsdialogfeld des Abfrage-Editors in SSMS und nicht im Objekt-Explorer. Wenn das Dialogfeld des Objekt-Explorers tatsächlich geöffnet wird, brechen Sie das Dialogfeld ab und öffnen den Abfrage-Editor erneut. In der folgenden Abbildung finden Sie die Menüoptionen, die Sie beim Herstellen einer Verbindung mit Babelfish 1.3.0 oder älteren Versionen auswählen können.



Weitere Informationen über Interoperabilität und Verhaltensunterschiede zwischen SQL Server und Babelfish finden Sie unter Unterschiede zwischen Babelfish für Aurora PostgreSQL und SQL Server.

Herstellen einer Verbindung mit dem DB-Cluster mit sqlcmd

Sie können sich mit einem DB-Cluster von Aurora PostgreSQL verbinden und mit diesem interagieren, der Babelfish unterstützt, indem Sie ausschließlich die Version 19.1 und niedriger des Befehlszeilen-Clients `sqlcmd` des SQL Servers verwenden. SSMS-Version 19.2 wird nicht unterstützt, um eine Verbindung zu einem Babelfish-Cluster herzustellen. Verwenden Sie den nachfolgenden Befehl, um eine Verbindung herzustellen.

```
sqlcmd -S endpoint,port -U login-id -P password -d your-DB-name
```

Es handelt sich um folgende Optionen:

- `-S` ist der Endpunkt und (optional) TDS-Port des DB-Clusters.
- `-U` ist der Anmeldename des Benutzers.
- `-P` ist das Passwort, das dem Benutzer zugeordnet ist.
- `-d` ist der Name deiner Babelfish-Datenbank.

Nach der Verbindung können Sie viele der gleichen Befehle verwenden, die Sie mit SQL Server verwenden. Einige Beispiele finden Sie unter Abrufen von Informationen aus dem Babelfish-Systemkatalog.

Herstellen einer Verbindung mit dem DB-Cluster mit SSMS

Mit Microsoft SQL Server Management Studio (SSMS) können Sie eine Verbindung mit einem DB-Cluster von Aurora PostgreSQL herstellen, der Babelfish ausführt. SSMS enthält eine Vielzahl von Tools, darunter den SQL-Server-Assistenten für Import und Export, der in Migrieren einer SQL-Server-Datenbank zu Babelfish for Aurora PostgreSQL behandelt wird. Weitere Informationen zu SSMS finden Sie unter Download von SQL Server Management Studio (SSMS) in der Microsoft-Dokumentation. Informationen zur Konfiguration von SSL/TLS finden Sie unter Verwenden von SSL mit einer DB-Instance von Microsoft SQL Server.

Stellen Sie eine Verbindung mit Ihrer Babelfish-Datenbank mit SSMS wie folgt her:

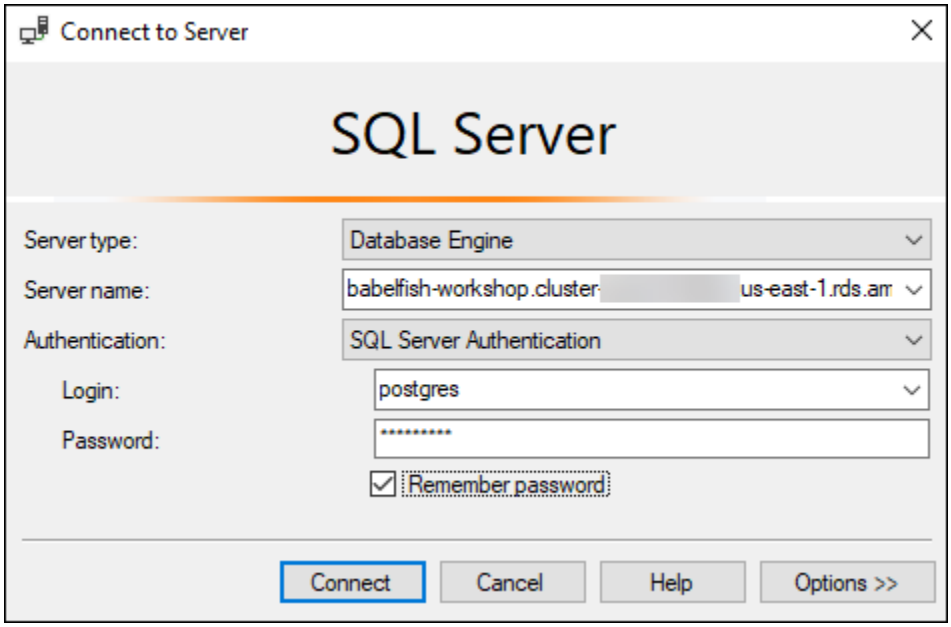
1. Starten Sie SSMS.
2. Öffnen Sie das Dialogfeld Mit Server verbinden. Führen Sie einen der folgenden Schritte aus, um mit der Verbindung fortzufahren:
   - Klicken Sie auf Neue Abfrage.
   - Wenn der Abfrage-Editor geöffnet ist, wählen Sie Abfragen, Connection (Verbindung), Verbinden aus.
3. Stellen Sie folgende Informationen für Ihre Datenbank bereit:
   a. Wählen Sie für Servertyp die Option Datenbank-Engine aus.
   b. Geben Sie für Server name (Servername) den DNS-Namen ein. Ihr Servername sollte beispielsweise wie folgt aussehen.

```
cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
```

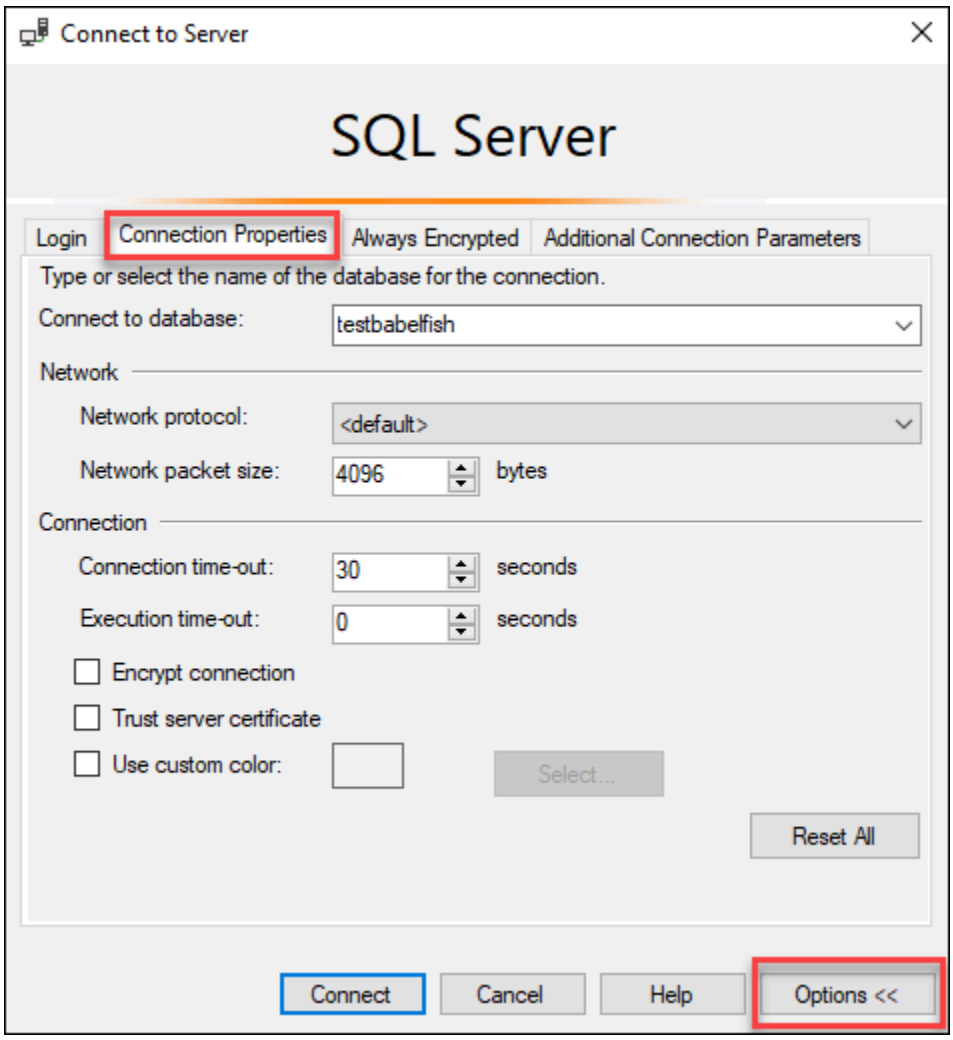
c. Wählen Sie für Authentifizierung die Option SQL Server-Authentifizierung aus.

d. Für Anmeldung geben Sie den Benutzernamen ein, den Sie beim Erstellen Ihrer Datenbank ausgewählt haben.

e. Für Passwort geben Sie das Passwort ein, das Sie beim Erstellen Ihrer Datenbank ausgewählt haben.



4. (Optional) Wählen Sie Optionen und wählen Sie dann die Registerkarte Verbindungseigenschaften.



5. (Optional) Für Verbindung zur Datenbank herstellen, geben Sie den Namen der migrierten SQL Server-Datenbank an, mit der eine Verbindung hergestellt werden soll, und wählen Sie Verbinden aus.

   Wenn eine Meldung angezeigt wird, die angibt, dass SSMS keine Verbindungszeichenfolgen anwenden kann, wählen Sie OK aus.

   Wenn Sie Probleme haben, sich mit Babelfish zu verbinden, finden Sie weitere Informationen unter Verbindungsfehler.

   Weitere Informationen zu Verbindungsproblemen mit SQL Server finden Sie unter Fehlerbehebung bei Verbindungen mit Ihrer SQL-Server-DB-Instance im Amazon-RDS-Benutzerhandbuch.

Verbinden mit Ihrem DB-Cluster mit einem PostgreSQL-Client

Sie können einen PostgreSQL-Client verwenden, um eine Verbindung mit Babelfish auf dem PostgreSQL-Port herzustellen. Ab Version 5.1.0 erzwingt der Babelfish-Server standardmäßig eine durchgängige Verbindungsverschlüsselung. Aktualisieren Sie Ihre Anwendung für die Verwendung mit SSL/TLS-Zertifikaten. Weitere Informationen zur Konfiguration von SSL/TLS-Zertifikaten finden Sie unter the section called “Sicherung von Aurora-PostgreSQL-Daten mit SSL/TLS”.

Herstellen einer Verbindung mit Ihrem psql

Sie können den PostgreSQL-Client von der PostgreSQL-Website herunterladen. Folgen Sie den Anweisungen für Ihr Betriebssystem, um diese Version zu installieren.

Sie können einen Aurora-PostgreSQL-DB-Cluster abfragen, der Babelfish mit dem Befehlszeilen-Client `psql` unterstützt. Verwenden Sie beim Herstellen der Verbindung den PostgreSQL-Port (standardmäßig Port 5432). In der Regel müssen Sie die Portnummer nicht angeben, es sei denn, Sie haben die Standardeinstellung geändert. Verwenden Sie den folgenden Befehl, um eine Verbindung mit Babelfish über den `psql`-Client herzustellen:

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
-p 5432 -U postgres -d babelfish_db
```

Dabei werden die folgenden Parameter verwendet:

- `-h` – Der Hostname des DB-Clusters (Cluster-Endpunkt), auf den Sie zugreifen möchten.
- `-p` – Die Nummer des PostgreSQL-Ports, der für die Verbindung mit der DB-Instance verwendet wird.
- `-d` – Die Datenbank, mit der Sie eine Verbindung herstellen möchten. Der Standardwert ist `babelfish_db`.
- `-U` – Das Datenbank-Benutzerkonto, auf das Sie zugreifen möchten. (Das Beispiel zeigt den Standard-Hauptbenutzernamen.)

Wenn Sie einen SQL-Befehl auf dem PSQL-Client ausführen, beenden Sie den Befehl mit einem Semikolon. Der folgende SQL-Befehl fragt beispielsweise die pg_tables Systemansicht ab, um Informationen über jede Tabelle in der Datenbank zurückzugeben.

```
SELECT * FROM pg_tables;
```

Der PSQL-Client verfügt auch über eine Reihe von integrierten Metakommanden. Eine metacommand ist eine Verknüpfung, die die Formatierung anpasst oder eine Verknüpfung bereitstellt, die Metadaten in einem einfach zu bedienenden Format zurückgibt. Beispielsweise gibt der Meta-Befehl in etwa die ähnlichen Informationen wie der vorige SQL_Befehl zurück:

```
\d
```

Metacommands müssen nicht mit einem Semikolon (;) beendet werden.

Um den PSQL-Client zu beenden, geben Sie \q ein.

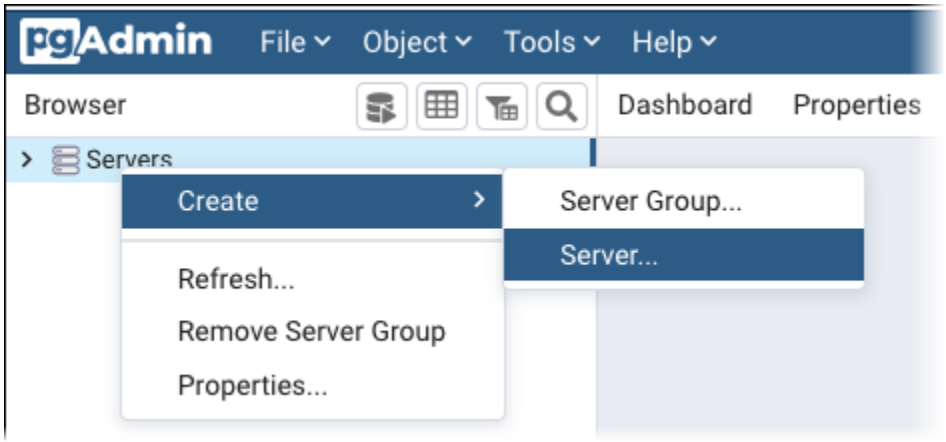
Weitere Informationen über die Verwendung des PSQL-Clients zum Abfragen eines Aurora-PostgreSQL-Clusters finden Sie unter Die PostgreSQL-Dokumentation.

Herstellen einer Verbindung mit dem DB-Cluster mit pgAdmin

Sie können den pgAdmin-Client verwenden, um im nativen PostgreSQL-Dialekt auf Ihre Daten zuzugreifen.

Stellen Sie eine Verbindung zum Cluster mit dem pgAdmin-Client wie folgt her:

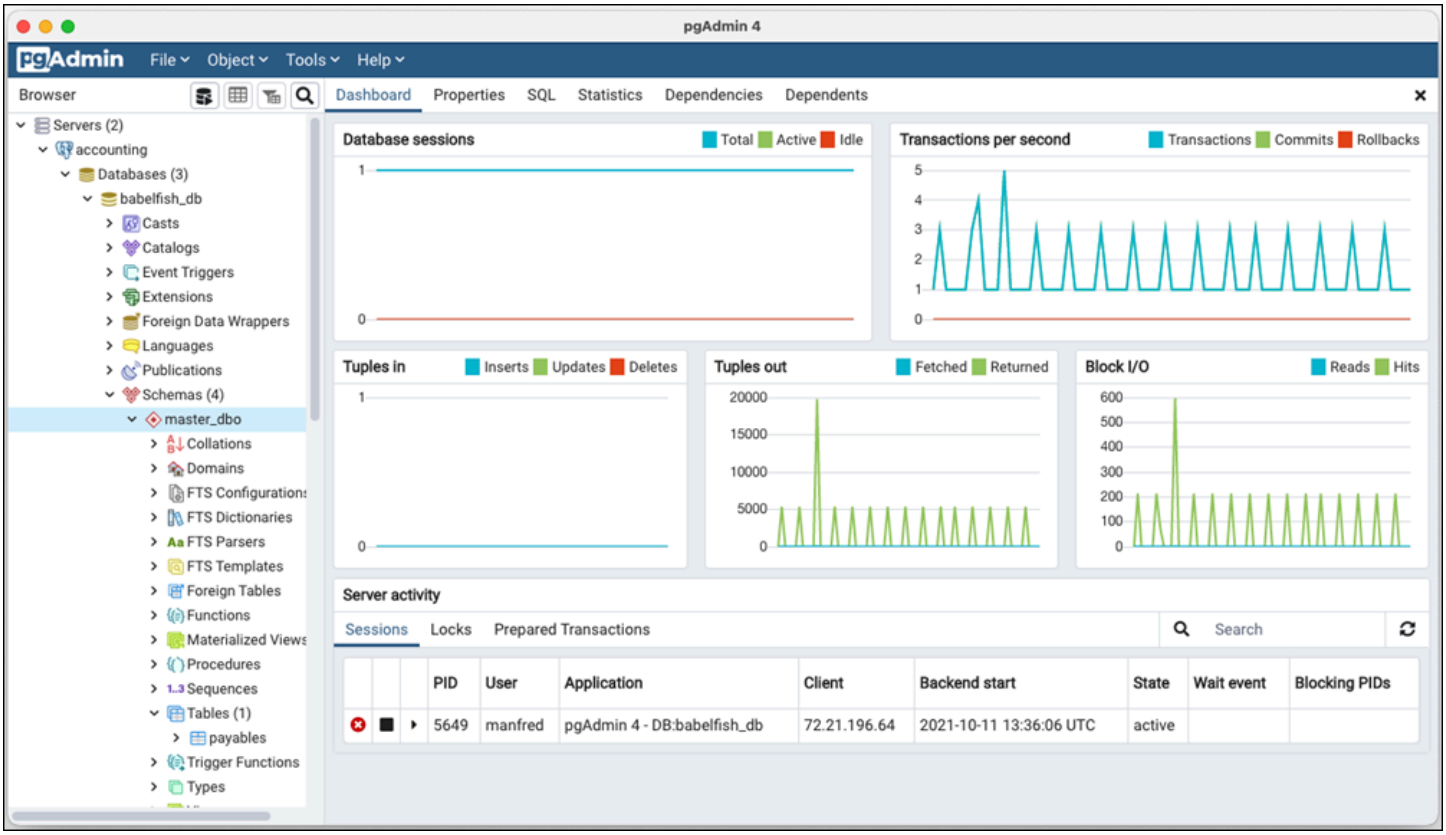
1. Laden Sie den pgAdmin-Client von pgAdmin Webseite herunter.
2. Öffnen Sie den Client und authentifizieren Sie sich bei pgAdmin.
3. Öffnen Sie das Kontextmenü (rechte Maustaste) für Server und klicken Sie auf und danach auf Erstellen, Server.



4. Geben Sie die Informationen im Dialogfeld Erstellen - Server ein.

   Auf erConnection (Verbindung) geben Sie die Aurora-PostgreSQL-Clusteradresse für Host und die PostgreSQL-Portnummer (standardmäßig 5432) für Port ein. Geben Sie Authentifizierungsdetails an und wählen Sie Save aus.

Nach der Verbindung können Sie die pgAdmin-Funktionalität verwenden, um Ihren Aurora-PostgreSQL-Cluster am PostgreSQL-Port zu überwachen und zu verwalten.



Weitere Informationen hierzu finden Sie auf der Webseite pgAdmin.

Arbeiten mit Babelfish

Im Folgenden finden Sie Nutzungsinformationen für Babelfish, darunter einige Unterschiede zwischen Babelfish und SQL Server sowie zwischen Babelfish- und PostgreSQL-Datenbanken.

Themen

- Abrufen von Informationen aus dem Babelfish-Systemkatalog
- Verwalten von Berechtigungen und Zugriffskontrolle in Babelfish für Aurora PostgreSQL
- Unterschiede zwischen Babelfish für Aurora PostgreSQL und SQL Server
- Verwenden von Babelfish-Funktionen mit eingeschränkter Implementierung
- Verbessern der Abfrageleistung in Babelfish
- Verwenden von Aurora PostgreSQL-Erweiterungen mit Babelfish
- Babelfish unterstützt verknüpfte Server
- Verwenden der Volltextsuche in Babelfish
- Babelfish unterstützt raumbezogene Datentypen
- Verstehen der Partitionierung in Babelfish

Abrufen von Informationen aus dem Babelfish-Systemkatalog

Sie können Informationen über die Datenbankobjekte abrufen, die in Ihrem Babelfish-Cluster gespeichert sind, indem Sie viele der Systemansichten abfragen, die auch in SQL Server verwendet werden. Jede neue Version von Babelfish bietet Unterstützung für weitere Systemansichten. Eine Liste der derzeit verfügbaren Ansichten finden Sie in der Tabelle SQL Server system catalog views.

Diese Systemansichten enthalten Informationen aus dem Systemkatalog (`sys.schemas`). Bei Babelfish enthalten diese Ansichten sowohl SQL-Server- als auch PostgreSQL-Systemschemas. Zum Abfragen von Informationen zum Systemkatalog in Babelfish können Sie den TDS-Port oder den PostgreSQL-Port verwenden, wie in den folgenden Beispielen gezeigt.

- Fragen Sie den T-SQL-Port mit **`sqlcmd`** oder einem anderen SQL-Server-Client ab.

  ```
  1> SELECT * FROM sys.schemas
  2> GO
  ```

  Diese Abfrage gibt SQL-Server- und Aurora-PostgreSQL-Systemschemas zurück, wie im Folgenden dargestellt.

```
name
------------------------------------------------------------
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- Fragen Sie den PostgreSQL-Port mit **`psql`** oder **`pgAdmin`** ab. In diesem Beispiel verwenden wir den Metabefehl `psql` zum Auflisten von Schemas (`\dn`):

  ```
  babelfish_db=> \dn
  ```

  Die Abfrage gibt dieselbe Ergebnismenge zurück wie die von `sqlcmd` am T-SQL-Port.

  ```
          List of schemas
               Name
  ------------------------------

   demographic_dbo

   public
   sys
   master_dbo
   tempdb_dbo
  ...
  ```

SQL-Server-Systemkataloge in Babelfish verfügbar

In der folgenden Tabelle finden Sie die derzeit in Babelfish implementierten SQL-Server-Ansichten. Weitere Informationen über die Systemkataloge in SQL Server finden Sie unter Systemkatalogansichten (Transact-SQL) in der Microsoft-Dokumentation.

Name der Ansicht	Beschreibung oder Babelfish-Beschränkung (falls vorhanden)
`sys.all_columns`	Alle Spalten in allen Tabellen und Ansichten

Name der Ansicht	Beschreibung oder Babelfish-Beschränkung (falls vorhanden)
`sys.all_objects`	Alle Objekte in allen Schemas
`sys.all_sql_modules`	Die Vereinigung von `sys.sql_modules` und `sys.system_sql_modules`
`sys.all_views`	Alle Ansichten in allen Schemas
`sys.columns`	Alle Spalten in benutzerdefinierten Tabellen und Ansichten
`sys.configurations`	Die Unterstützung von Babelfish ist auf eine einzige schreibgeschützte Konfiguration beschränkt.
`sys.data_spaces`	Enthält eine Zeile für jeden Datenbereich. Dies kann eine Dateigruppe, ein Partitionsschema oder eine FILESTREAM-Datendateigruppe sein.
`sys.database_files`	Eine Ansicht pro Datenbank, die eine Zeile für jede Datei einer Datenbank enthält, wie sie in der Datenbank selbst gespeichert ist.
`sys.database_mirroring`	Weitere Informationen finden Sie unter sys.database_mirroring in der Microsoft-Transact-SQL-Dokumentation.
`sys.database_principals`	Weitere Informationen finden Sie unter sys.database_principals in der Microsoft-Transact-SQL-Dokumentation.
`sys.database_role_members`	Weitere Informationen finden Sie unter sys.database_role_members in der Microsoft-Transact-SQL-Dokumentation.
`sys.databases`	Alle Datenbanken in allen Schemas

Name der Ansicht	Beschreibung oder Babelfish-Beschränkung (falls vorhanden)
`sys.dm_exec_connections`	Weitere Informationen finden Sie unter sys.dm_exec_connections in der Microsoft-Transact-SQL-Dokumentation.
`sys.dm_exec_sessions`	Weitere Informationen finden Sie unter sys.dm_exec_sessions in der Microsoft-Transact-SQL-Dokumentation.
`sys.dm_hadr_database_replica_states`	Weitere Informationen finden Sie unter sys.dm_hadr_database_replica_states in der Microsoft-Transact-SQL-Dokumentation.
`sys.dm_os_host_info`	Weitere Informationen finden Sie unter sys.dm_os_host_info in der Microsoft-Transact-SQL-Dokumentation.
`sys.endpoints`	Weitere Informationen finden Sie unter sys.endpoints in der Microsoft-Transact-SQL-Dokumentation.
`sys.indexes`	Weitere Informationen finden Sie unter sys.indexes in der Microsoft-Transact-SQL-Dokumentation.
`sys.languages`	Weitere Informationen finden Sie unter sys.languages in der Microsoft-Transact-SQL-Dokumentation.
`sys.schemas`	Alle Schemas
`sys.server_principals`	Alle Logins und Rollen
`sys.sql_modules`	Weitere Informationen finden Sie unter sys.sql_modules in der Microsoft-Transact-SQL-Dokumentation.

Name der Ansicht	Beschreibung oder Babelfish-Beschränkung (falls vorhanden)
`sys.sysconfigures`	Die Unterstützung von Babelfish ist auf eine einzige schreibgeschützte Konfiguration beschränkt.
`sys.syscurconfigs`	Die Unterstützung von Babelfish ist auf eine einzige schreibgeschützte Konfiguration beschränkt.
`sys.sysprocesses`	Weitere Informationen finden Sie unter sys.sysprocesses in der Microsoft-Transact-SQL-Dokumentation.
`sys.system_sql_modules`	Weitere Informationen finden Sie unter sys.system_sql_modules in der Microsoft-Transact-SQL-Dokumentation.
`sys.table_types`	Weitere Informationen finden Sie unter sys.table_types in der Microsoft-Transact-SQL-Dokumentation.
`sys.tables`	Alle Tabellen in einem Schema
`sys.xml_schema_collections`	Weitere Informationen finden Sie unter sys.xml_schema_collections in der Microsoft-Transact-SQL-Dokumentation.

PostgreSQL implementiert Systemkataloge, die den SQL Server-Objektkatalogansichten ähneln. Eine vollständige Liste der Systemkataloge finden Sie unter Systemkataloge in der PostgreSQL-Dokumentation.

Von Babelfish unterstützte DDL-Exporte

Ab den Versionen Babelfish 2.4.0 und 3.1.0 unterstützt Babelfish DDL-Exporte mit verschiedenen Tools. Sie können diese Funktion beispielsweise von SQL Server Management Studio (SSMS) aus verwenden, um die Datendefinitionsskripte für verschiedene Objekte in einer Datenbank von Babelfish für Aurora PostgreSQL zu generieren. Anschließend können Sie die generierten DDL-

Befehle in diesem Skript verwenden, um dieselben Objekte in einer anderen Datenbank von Babelfish für Aurora PostgreSQL oder SQL Server zu erstellen.

Babelfish unterstützt DDL-Exporte für die folgenden Objekte in den angegebenen Versionen.

Liste von Objekten	2.4.0	3.1.0
Tabelle USER	Ja	Ja
Primärschlüssel	Ja	Ja
Fremdschlüssel	Ja	Ja
Eindeutigkeitseinschränkungen	Ja	Ja
Indizes	Ja	Ja
Einschränkungen prüfen	Ja	Ja
Ansichten	Ja	Ja
Gespeicherte Prozeduren	Ja	Ja
Benutzerdefinierte Funktionen	Ja	Ja
Funktionen mit Tabellenwerten	Ja	Ja
Auslöser	Ja	Ja
Benutzerdefinierte Datentypen	Nein	Nein
Benutzerdefinierte Tabellentypen	Nein	Nein
Benutzer	Nein	Nein
Anmeldungen	Nein	Nein
Sequenzen	Nein	Nein
Rollen	Nein	Nein

Einschränkungen bei den exportierten DDLs

- Verwenden von Escape-Schraffuren, bevor die Objekte mit den exportierten DDLs neu erstellt werden – Babelfish unterstützt nicht alle Befehle im exportierten DDL-Skript. Verwenden Sie Escape-Schraffuren, um Fehler zu vermeiden, die beim Neuerstellen der Objekte aus den DDL-Befehlen in Babelfish entstehen. Weitere Informationen zu Escape-Schraffuren finden Sie unter Verwalten der Babelfish-Fehlerbehandlung mit Escape-Schraffuren.
- Objekte, die CHECK-Einschränkungen mit expliziten COLLATE-Klauseln enthalten – Die Skripts mit diesen Objekten, die aus einer SQL Server-Datenbank generiert wurden, haben unterschiedliche, aber gleichwertige Sortierungen wie in der Babelfish-Datenbank. Beispielsweise werden einige Sortierungen, wie sql_latin1_general_cp1_cs_as, sql_latin1_general_cp1251_cs_as und latin1_general_cs_as, als latin1_general_cs_as generiert, was der Windows-Sortierung am nächsten kommt.

Verwalten von Berechtigungen und Zugriffskontrolle in Babelfish für Aurora PostgreSQL

In Babelfish für Aurora PostgreSQL können Sie Berechtigungen sowie die Zugriffskontrolle für Datenbanken, Schemas und Objekte verwalten. In den folgenden Tabellen werden die spezifischen SQL-Befehle zur Erteilung von Berechtigungen in Babelfish beschrieben, um verschiedene Zugriffskontrollszenarien zu erreichen. Es werden unterstützte Anwendungsfälle behandelt, die implementiert werden können, sowie Problemumgehungen für derzeit nicht unterstützte Fälle. Auf diese Weise können Sie die entsprechenden Berechtigungen konfigurieren, um Ihre Sicherheits- und Compliance-Anforderungen bei der Arbeit mit Babelfish-Datenbanken zu erfüllen.

Unterstützte Anwendungsfälle

In der folgenden Tabelle werden die Anwendungsfälle erklärt, die in Babelfish unterstützt werden. Für jeden Anwendungsfall zeigt die Tabelle die Aktionen, die zur Erreichung dieses Ziels erforderlich sind, sowie Beispiele für SQL-Befehle.

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Babelfish -Versions kompatibilität
Der Anmeldung erlauben,	Die Anmeldung der sysadmin-	ALTER SERVER ROLE sysadmin	Keine	Alle Versionen

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Babelfish -Versions kompatibilität
SELECTs/DMLs/DDLs in einer beliebigen Datenbank auszuführen	Serverrolle hinzufügen	ADD MEMBER `login`		
Der Anmeldung erlauben, SELECTs/DMLs/DDLs in einer Datenbank auszuführen	Die Anmeldung zum Eigentümer der Datenbank machen	ALTER AUTHORIZATION ON DATABASE::`database` TO `login`	Eine Datenbank kann nur einen Eigentümer haben.	Version 3.4 und höher
Dem Datenbankbenutzer erlauben, SELECTs/DMLs in einem Schema auszuführen	Dem Datenbankbenutzer Berechtigungen für ein Schema erteilen	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON SCHEMA::`schema` TO `user`	Keine	Version 3.6 und höher, 4.2 und höher
Dem Datenbankbenutzer erlauben, SELECTs/DMLs in einem Schema auszuführen	Den Datenbankbenutzer bei der Schemaerstellung zum Eigentümer des Schemas machen	CREATE SCHEMA `schema` AUTHORIZATION `user`	Das Ändern der Schemaeigentümerschaft nach der Erstellung wird derzeit nicht unterstützt.	Version 1.2 und höher

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Babelfish -Versions kompatibilität
Dem Datenbankbenutzer erlauben, SELECTs/DMLs in einem Objekt auszuführen	Dem Datenbankbenutzer Berechtigungen auf ein Objekt erteilen	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON OBJECT::`object` TO `user`	Keine	Alle Versionen
Dem Datenbankbenutzer erlauben, SELECTs/DMLs/DDLs in einer Datenbank auszuführen, einschließlich des Löschens der Datenbank	Den Benutzer zur festen Datenbankrolle db_owner hinzufügen	ALTER ROLE db_owner ADD MEMBER `user`	Nur Benutzer können der festen Datenbankrolle db_owner hinzugefügt werden. Das Hinzufügen von Rollen zur Rolle db_owner wird noch nicht unterstützt.	Version 4.5 und höher, 5.1 und höher
Dem Benutzer oder den Mitgliedern einer benutzerdefinierten Datenbankrolle erlauben, nur SELECTs in einer Datenbank auszuführen	Den Benutzer oder die Rolle zur festen Datenbankrolle db_datareader hinzufügen	ALTER ROLE db_datareader ADD MEMBER `user` / `role`	Keine	Version 4.5 und höher, 5.1 und höher

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Babelfish -Versions kompatibilität
Dem Benutzer oder den Mitgliedern einer benutzerdefinierten Datenbankrolle erlauben, nur DMLs in einer Datenbank auszuführen	Den Benutzer oder die Rolle zur festen Datenbankrolle db_datawriter hinzufügen	ALTER ROLE db_datawriter ADD MEMBER `user` / `role`	Keine	Version 4.5 und höher, 5.1 und höher
Dem Benutzer oder den Mitgliedern einer benutzerdefinierten Datenbankrolle erlauben, nur DDLs in einer Datenbank auszuführen	Den Benutzer oder die Rolle zur festen Datenbankrolle db_accessadmin hinzufügen	ALTER ROLE db_accessadmin ADD MEMBER `user` / `role`	Keine	Version 4.5 und höher, 5.1 und höher

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Babelfish -Versions kompatibilität
Dem Benutzer oder den Mitgliedern einer benutzerdefinierten Datenbankrolle ausschließlich die Ausführung folgender Aktionen erlauben: CREATE/ALTER/DROP für benutzerdefinierte Rollen, GRANT/REVOKE für Berechtigungen für Objekte in einer Datenbank und/oder CREATE SCHEMA in einer Datenbank	Den Benutzer oder die Rolle zur festen Datenbankrolle db_securityadmin hinzufügen	ALTER ROLE db_securityadmin ADD MEMBER `user` / `role`	Keine	Version 4.5 und höher, 5.1 und höher

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Babelfish -Versions kompatibilität
Dem Benutzer oder den Mitgliedern einer benutzerdefinierten Datenbankrolle ausschließlich die Ausführung folgender Aktionen erlauben: CREATE/ALTER/DROP für jeden Benutzer, GRANT sowie REVOKE für den Datenbankzugriff, Zuordnungen der Benutzerkonten zu Anmeldungen und/oder CREATE SCHEMA in einer Datenbank	Den Benutzer oder die Rolle zur festen Datenbankrolle db_accessadmin hinzufügen	ALTER ROLE db_accessadmin ADD MEMBER `user` / `role`	Keine	Version 4.5 und höher, 5.1 und höher

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Babelfish -Versions kompatibilität
Der Anmeldung erlauben, nur CREATE/DROP/ALTER für jede Datenbank auszuführen	Die Anmeldung zur festen Serverrolle dbcreator hinzufügen	ALTER SERVER ROLE dbcreator ADD MEMBER `login`	Es können nur die Datenbanken geändert werden, auf die die dbcreator -Anmeldung Zugriff hat.	Version 4.5 und höher, 5.1 und höher
Der Anmeldung erlauben, nur CREATE/ALTER/DROP für jede Anmeldung auszuführen	Die Anmeldung zur festen Serverrolle securityadmin hinzufügen	ALTER SERVER ROLE securityadmin ADD MEMBER `login`	Keine	Version 4.5 und höher, 5.1 und höher

Nicht unterstützte Anwendungsfälle mit den Problemumgehungen

In der folgenden Tabelle werden die Anwendungsfälle erklärt, die in Babelfish nicht unterstützt werden, aber mithilfe einer Problemumgehung umgesetzt werden können.

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Kompatibilität der Babelfish -Version für Problemumgehungen
Dem Benutzer erlauben, SELECTs/DMLs für ein Objekt/Schema auszuführen, zusammen mit der Option,	GRANT ausführen, um die Berechtigungen allen anderen Benutzern direkt zu erteilen	GRANT SELECT/EXECUTE/INSERT/UPDATE/DELETE ON OBJECT/SC	GRANT ... WITH GRANT OPTION wird derzeit nicht unterstützt.	Version 3.6 und höher, 4.2 und höher

Anwendungsfall	Aktion	SQL-Befehle	Kommentare	Kompatibilität der Babelfish -Version für Problemum gehungen
diese Berechtigungen mit GRANT anderen Benutzern zu erteilen		HEMA::`object/sc` TO `user`		
Dem Datenbankbenutzer erlauben, SELECTs/DMLs/DDLs in einer Datenbank auszuführen, einschließlich des Löschens der Datenbank	Die Mitglieder der Rolle zur festen Datenbankrolle db_owner hinzufügen	ALTER ROLE db_owner ADD MEMBER `user`	Das Hinzufügen von Rollen zur Rolle db_owner wird derzeit nicht unterstützt.	Version 4.5 und höher, 5.1 und höher

Nicht unterstützte Anwendungsfälle

In der folgenden Tabelle werden die Anwendungsfälle erklärt, die in Babelfish nicht unterstützt werden.

Anwendungsfall	Kommentare
Dem Benutzer oder den Mitgliedern einer benutzerdefinierten Datenbankrolle verweigern, SELECTs in einer Datenbank auszuführen	Die feste Datenbankrolle db_denydatareader wird noch nicht unterstützt.
Dem Benutzer oder den Mitgliedern einer benutzerdefinierten Datenbankrolle verweigern, DMLs in einer Datenbank auszuführen	Die feste Datenbankrolle db_denydatawriter wird derzeit nicht unterstützt.

Anwendungsfall	Kommentare
Der Anmeldung erlauben, nur KILL für jede Datenbankverbindung auszuführen	Die feste Serverrolle processadmin wird derzeit nicht unterstützt.
Der Anmeldung erlauben, nur verknüpfte Server hinzuzufügen oder zu entfernen	Die feste Serverrolle setupadmin wird derzeit nicht unterstützt.

Beheben von Unterschieden in der Objekteigentümerschaft nach dem Upgrade

Die Babelfish-Versionen 4.6 und höher sowie 5.2 und höher beinhalten eine Änderung der Verwaltung der Objekteigentümerschaft über den TDS-Endpunkt. Wenn Sie neue Objekte über den TDS-Endpunkt erstellen, gehören diese Objekte nun dem Eigentümer des Schemas und nicht mehr dem aktuellen Benutzer. Diese Änderung der Eigentümerschaft kann sich auf das Verhalten der Berechtigungen für neue Objekte im Vergleich zu vorhandenen Objekten auswirken, wenn Sie ein Upgrade von Versionen vor 4.6 oder 5.2 durchführen.

Zum Beheben dieser Unterschiede in der Eigentümerschaft bietet Babelfish die Funktion `sys.generate_alter_ownership_statements()`. Diese Funktion generiert SQL-Anweisungen, die sicherstellen, dass Objekt- und Schemaeigentümerschaft übereinstimmen.

Beachten Sie folgende Einschränkungen beim Umgang mit der Objekteigentümerschaft:

- Benutzer mit CREATE-Berechtigungen, die über den PostgreSQL-Endpunkt erteilt wurden, können in diesen Schemas keine Objekte über den TDS-Endpunkt erstellen.
- Das Ändern von Berechtigungen für T-SQL-Objekte über den PostgreSQL-Endpunkt wird nicht empfohlen und kann zu falschem T-SQL-Verhalten führen.
- Die Zugriffsberechtigungen können zwischen alten und neuen Objekten variieren, wenn deren Eigentümerschaft nicht übereinstimmt. Beispielsweise kann ein Schema, das `sch_own` gehört, Objekte enthalten, die `dbo` gehören. In diesem Fall verfügen Objekte, die `dbo` gehören und vor dem Upgrade erstellt wurden, möglicherweise über andere Zugriffsberechtigungen als Objekte, die `sch_own` gehören und nach dem Upgrade erstellt wurden. Dies kann sich auf Operationen wie SELECT und INSERT auswirken.

Wenn Ihr DB-Cluster Objekte enthält, die in Babelfish-Versionen vor 4.6 oder 5.2 erstellt wurden, sollten Sie erwägen, deren Eigentümerschaft anzugleichen.

So beheben Sie Unterschiede in der Objekteigentümerschaft

1. Stellen Sie über den PostgreSQL-Endpunkt eine Verbindung zur `babelfish_db`-Datenbank in Ihrem DB-Cluster her.
2. Führen Sie den folgenden Befehl aus:

   ```
   SELECT * from sys.generate_alter_ownership_statements();
   ```

   Dadurch wird eine Liste von SQL-Anweisungen generiert, mit denen die Eigentümerschaft der Objekte standardisiert werden soll.
3. Führen Sie die generierten Anweisungen zunächst in einer Testumgebung aus, um ihre Wirkung zu überprüfen, bevor Sie sie auf Ihre Produktionsumgebung anwenden.

Wir empfehlen, dass Sie diese Anweisungen ausführen, um ein konsistentes Objekteigentümerschaftsmodell in Ihrer gesamten Datenbank zu erreichen.

Unterschiede zwischen Babelfish für Aurora PostgreSQL und SQL Server

Babelfish ist ein sich weiterentwickelnder Aurora-PostgreSQL-Service, der seit dem ersten Angebot in Aurora PostgreSQL 13.4 in jeder Version um weitere Funktionen ergänzt wurde. Babelfish wurde entwickelt, um T-SQL-Semantik zusätzlich zu PostgreSQL über den T-SQL-Dialekt unter Verwendung des TDS-Ports zur Verfügung zu stellen. Jede neue Version von Babelfish bietet weitere Funktionen, die besser auf die T-SQL-Funktionalität und das -Verhalten ausgerichtet sind, wie in der Tabelle Unterstützte Funktionen in Babelfish nach Version gezeigt. Für optimale Ergebnisse bei der Arbeit mit Babelfish sollten Sie die Unterschiede verstehen, die derzeit zwischen dem von SQL Server unterstützten T-SQL und Babelfish für die neueste Version bestehen. Weitere Informationen hierzu finden Sie unter T-SQL-Unterschiede bei Babelfish.

Zusätzlich zu den Unterschieden zwischen T-SQL, das von Babelfish unterstützt wird, und SQL Server, müssen Sie möglicherweise auch Interoperabilitätsprobleme zwischen Babelfish und PostgreSQL im Kontext des DB-Clusters von Aurora PostgreSQL berücksichtigen. Wie bereits erwähnt, unterstützt Babelfish T-SQL-Semantik zusätzlich zu PostgreSQL über den T-SQL-Dialekt unter Verwendung des TDS-Ports. Gleichzeitig kann auf die Babelfish-Datenbank auch über den Standard-PostgreSQL-Port mit SQL-Anweisungen von PostgreSQL zugegriffen werden. Wenn Sie erwägen, sowohl PostgreSQL- als auch Babelfish-Funktionen in einer Produktionsbereitstellung zu verwenden, müssen Sie sich der potenziellen Interoperabilitätsprobleme zwischen Schemanamen, Bezeichnern, Berechtigungen, Transaktionssemantik, verschiedenen Ergebnismengen,

Standardsortierungen usw. bewusst sein. Einfach ausgedrückt, wenn PostgreSQL-Anweisungen oder PostgreSQL-Zugriffe im Kontext von Babelfish auftreten, kann es zu Interferenzen zwischen PostgreSQL und Babelfish kommen, die möglicherweise Syntax, Semantik und Kompatibilität beeinträchtigen können, wenn neue Versionen von Babelfish veröffentlicht werden. Vollständige Informationen und Anleitungen zu allen Überlegungen finden Sie in der Anleitung zur Interoperabilität von Babelfish in der Dokumentation von Babelfish für PostgreSQL.

> **Note**
>
> Bevor Sie sowohl die native PostgreSQL-Funktionalität als auch die Babelfish-Funktionalität im selben Anwendungskontext verwenden, empfehlen wir Ihnen dringend, die Probleme, die in der Anleitung zur Interoperabilität von Babelfish der Dokumentation von Babelfish für PostgreSQL erörtert werden, zu berücksichtigen. Diese Interoperabilitätsprobleme (Aurora PostgreSQL und Babelfish) sind nur relevant, wenn Sie die PostgreSQL-Datenbank-Instance im selben Anwendungskontext wie Babelfish verwenden möchten.

Themen

- Babelfish – Dump und Wiederherstellung
- T-SQL-Unterschiede bei Babelfish
- Transaktionsisolationsstufen in Babelfish

Babelfish – Dump und Wiederherstellung

Ab den Versionen 4.0.0 und 3.4.0 können Babelfish-Benutzer jetzt die Dienstprogramme für Dump und Wiederherstellung verwenden, um ihre Datenbanken zu sichern und wiederherzustellen. Weitere Informationen finden Sie unter Babelfish – Dump und Wiederherstellung. Diese Funktion basiert auf den PostgreSQL-Dienstprogrammen für Dump und Wiederherstellung. Weitere Informationen finden Sie unter pg_dump und pg_restore. Zur effektiven Verwendung dieser Funktion in Babelfish müssen Sie PostgreSQL-basierte Tools verwenden, die speziell für Babelfish angepasst sind. Die Sicherungs- und Wiederherstellungsfunktion für Babelfish unterscheidet sich erheblich von der von SQL Server. Weitere Informationen zu diesen Unterschieden finden Sie unter Unterschiede in den Dump- und Wiederherstellungsfunktionen: Babelfish und SQL Server. Babelfish für Aurora PostgreSQL bietet zusätzliche Funktionen für die Sicherung und Wiederherstellung von DB-Clustern von Amazon Aurora PostgreSQL. Weitere Informationen finden Sie unter Sichern und Wiederherstellen eines Amazon-Aurora-DB-Clusters.

T-SQL-Unterschiede bei Babelfish

Im Folgenden finden Sie eine Tabelle der T-SQL-Funktionen, die in der aktuellen Version von Babelfish unterstützt werden, mit einigen Anmerkungen zu Unterschieden im Verhalten gegenüber dem von SQL Server.

Weitere Informationen über den Support für verschiedene Versionen finden Sie unter Unterstützte Funktionen in Babelfish nach Version. Weitere Informationen zu Funktionen, die derzeit nicht unterstützt werden, finden Sie unter Nicht unterstützte Funktionen in Babelfish.

Babelfish ist mit der PostgreSQL-kompatiblen Edition von Aurora verfügbar. Weitere Informationen zu Babelfish-Versionen finden Sie unter Versionshinweise für Aurora PostgreSQL.

Funktionalität oder Syntax	Beschreibung des Verhaltens oder Unterschieds
\ (Zeilenfortsetzungszeichen)	Das Zeilenfortsetzungszeichen (ein umgekehrter Schrägstrich vor einem Zeilenumbruch) für Zeichen- und Hexadezimalzeichen folgen wird derzeit nicht unterstützt. Für Zeichenfolgen wird der Backslash für neue Zeile als Zeichen in der Zeichenfolge interpretiert. Bei Hexadezimalzeichenfolgen führt der Backslash für neue Zeile zu einem Syntaxfehler.
@@Version	Das Format des von `@@version` zurückgegebenen Wertes unterscheidet sich geringfügig von dem von SQL Server zurückgegebenen Wert. Ihr Code funktioniert möglicherweise nicht richtig, wenn er von der Formatierung von `@@version` abhängt.
Aggregationsfunktionen	Aggregatfunktionen werden teilweise unterstützt (AVG, COUNT, COUNT_BIG, GROUPING, MAX, MIN, STRING_AGG und SUM werden unterstützt). Eine Liste der nicht unterstützten Aggregatf unktionen finden Sie unter Nicht unterstützte Funktionen.
ALTER TABLE	Unterstützt nur das Hinzufügen oder Löschen einer einzelnen Spalte oder Einschränkung.
ALTER TABLE..ALTER COLUMN	NULL und NOT NULL können derzeit nicht angegeben werden. Wenn Sie die Nullfähigkeit einer Spalte ändern möchten,

Funktionalität oder Syntax	Beschreibung des Verhaltens oder Unterschieds
	verwenden Sie die PostgreSQL-Anweisung ALTER TABLE..{SET|DROP} NOT NULL.

AT TIME ZONE	Während des Übergangs von der Sommerzeit zur Standardzeitabweichung wird der sich überschneidende Zeitraum mit der Standardzeitabweichung angezeigt. Zur Verdeutlichung betrachten Sie folgendes Beispiel:
	`SELECT CONVERT(DATETIME2(0), '2022-10-30T02:00:00', 126) AT TIME ZONE 'Central European Standard Time';`
	`GO;`
	`Result: 2022-10-30 02:00:00 +01:00`

Leere Spaltennamen ohne Spaltenalias	Die `sqlcmd`- und `psql`-Dienstprogramme behandeln Spalten mit leeren Namen unterschiedlich: • SQL Server `sqlcmd` gibt einen leeren Spaltennamen zurück. • PostgreSQL `psql` gibt einen generierten Spaltennamen zurück.
Die Funktion CHECKSUM	Babelfish und SQL Server verwenden unterschiedliche Hashing-Algorithmen für die CHECKSUM-Funktion. Infolgedessen können sich die von der CHECKSUM-Funktion in Babelfish generierten Hashwerte von denen unterscheiden, die von der CHECKSUM-Funktion in SQL Server generiert wurden.
Standardeinstellung der Spalte	Beim Erstellen einer Spalte wird der Name der Einschränkung ignoriert. Verwenden Sie die folgende Syntax, um eine Spalte zu löschen: `ALTER TABLE...ALTER COLUMN..DROP DEFAULT...`

Funktionalität oder Syntax	Beschreibung des Verhaltens oder Unterschieds
Constraint_name	In SQL Server müssen Einschränkungsnamen innerhalb des Schemas, zu dem die Tabelle gehört, eindeutig sein. In Babelfish gilt dies jedoch nur für PRIMARY KEY- und UNIQUE-Einschränkungen. Andere Arten von Einschränkungen unterlieg en dieser Einschränkung nicht.
Beschränkungen	PostgreSQL unterstützt das Ein- und Ausschalten einzelner Einschränkungen nicht. Die Aussage wird ignoriert und eine Warnung ausgegeben.
Einschränkungen mit IGNORE_DUP_KEY	Einschränkungen werden ohne diese Eigenschaft erstellt.
SERVERROLLE ERSTELLEN , ÄNDERN, LÖSCHEN	SERVERROLLE ÄNDERN wird nur für `sysadmin` unterstützt. Alle anderen Syntax wird nicht unterstützt. Der T-SQL-Benutzer in Babelfish hat eine Erfahrung, die SQL Server für die Konzepte eines Logins (Serverprinzipals), einer Datenbank und eines Datenbankbenutzers (Datenbankprinzipal) ähnelt.
CREATE, ALTER LOGIN-Kla useln werden mit eingeschr änkter Syntax unterstützt	Das CREATE LOGIN... PASSWORD-Klausel,... DEFAULT_D ATABASE-Klausel und... DEFAULT_LANGUAGE-Klausel werden unterstützt. Der ALTER LOGIN... PASSWORD-Klausel wird unterstützt, aber ALTER LOGIN... OLD_PASSWORD-Klaus el wird nicht unterstützt. Nur ein Login, der ein sysadmin-Mitglied ist, kann ein Passwort ändern.
Sortierung der Groß- und Kleinschreibung erstellen	Sortierungen unter Berücksichtigung der Groß-/Kleinschreibung werden mit der CREATE DATABASE-Anweisung nicht unterstüt zt.
CREATE DATABASE Schlüsselwörter und Klaus	Optionen außer COLLATE und CONTAINMENT=NONE werden nicht unterstützt. Die COLLATE-Klausel wird akzeptiert und wird immer auf den Wert von `babelfishpg_tsql.server_col lation_name` gesetzt.

Funktionalität oder Syntax	Beschreibung des Verhaltens oder Unterschieds
CREATE SCHEMA... unterstützende Klauseln	Sie können den Befehl CREATE SCHEMA verwenden, um ein leeres Schema zu erstellen. Verwenden Sie zusätzliche Befehle, um Schemaobjekte zu erstellen.
Die Werte der Datenbank-ID sind bei Babelfish unterschiedlich	Die Master- und tempdb-Datenbanken sind keine Datenbank-IDs 1 und 2.
Die FORMAT-Funktion für den Datumstyp wird mit den folgenden Einschränkungen unterstützt.	Der einstellige Meridian wird nicht unterstützt. Das Format „yyy“ in SQL Server gibt 4 Ziffern für Jahre über 1000 zurück, für andere jedoch nur 3 Ziffern. Die Formate „g“ und „R“ werden nicht unterstützt Die Übersetzung des Gebietsschemas „vi-VN“ unterscheidet sich geringfügig.
Bezeichner größer als 63 Zeichen	PostgreSQL unterstützt maximal 63 Zeichen für Bezeichner. Babelfish konvertiert Bezeichner, die länger als 63 Zeichen sind, in einen Namen, der einen Hash des ursprünglichen Namens enthält. Beispiel: Eine Tabelle, die als „AB(ABC1234567890123456789012345678901234567890123456789012345678901234567890“ erstellt wurde, kann in „ABC1234567890123456789012345678901234567890123456789012345678901234567890“ konvertiert werden.
Unterstützung für IDTITÄT	`IDENTITY` Spalten werden für Datentypen `tinyint`, `smallint`, `int`, `bigint`, `numeric` und `decimal` unterstützt. SQL Server unterstützt Genauigkeit auf 38 Stellen für Datentypen `numeric` und `decimal` in IDENTITY-Spalten. PostgreSQL unterstützt Genauigkeit auf 19 Stellen für Datentypen `numeric` und `decimal` in IDENTITY-Spalten.

Funktionalität oder Syntax	Beschreibung des Verhaltens oder Unterschieds
Indizes mit IGNORE_DUP_KEY	Die Syntax, die einen Index erstellt, der IGNORE_DUP_KEY enthält, erstellt einen Index, als würde diese Eigenschaft weggelassen.
Indizes mit mehr als 32 Spalten	Ein Index darf nicht mehr als 32 Spalten enthalten. Eingeschlossene Indexspalten zählen in PostgreSQL auf das Maximum, jedoch nicht in SQL Server.
Indizes (gruppiert)	Gruppierte Indizes werden erstellt, als ob NONCLUSTERED angegeben worden wäre.
Index-Klausel	Die folgenden Klauseln werden ignoriert: FILLFACTOR, ALLOW_PAGE_LOCKS, ALLOW_ROW_LOCKS, PAD_INDEX, STATISTICS_NORECOMPUTE, OPTIMIZE_FOR_SEQUENTIAL_KEY, SORT_IN_TEMPDB, DROP_EXISTING, ONLINE, COMPRESSION_DELAY, MAXDOP und DATA_COMPRESSION
JSON-Support	Die Reihenfolge der Name-Wert-Paare ist nicht garantiert. Der Array-Typ bleibt jedoch unberührt.
LOGIN-Objekte	Alle Optionen für LOGIN-Objekte werden nicht unterstützt, mit Ausnahme von PASSWORD, DEFAULT_DATABASE, DEFAULT_LANGUAGE, ENABLE, DISABLE.
NEWSEQUENTIALID-Funktion	Als NEWID implementiert; sequentielles Verhalten ist nicht garantiert. Beim Anrufen von `NEWSEQUENTIALID` generiert PostgreSQL einen neuen GUID-Wert.
Die OUTPUT-Klausel wird mit den folgenden Einschränkungen unterstützt	OUTPUT und OUTPUT INTO werden in derselben DML-Abfrage nicht unterstützt. Verweise auf Nicht-Zieltabelle von UPDATE- oder DELETE-Operationen in einer OUTPUT-Klausel werden nicht unterstützt. AUSGABE… GELÖSCHT *, EINGEFÜGT* werden in derselben Abfrage nicht unterstützt.

Funktionalität oder Syntax	Beschreibung des Verhaltens oder Unterschieds
Begrenzung von Prozedur- oder	Babelfish unterstützt maximal 100 Parameter für eine Prozedur oder Funktion.
ROWGUIDCOL	Diese Klausel wird derzeit ignoriert. Referenzfragen `$GUIDGOL` verursachen einen Syntaxfehler.
Unterstützung für SEQUENCE	SEQUENCE-Objekte werden für die Datentypen tinyint, smallint, int, bigint, numeric und decimal unterstützt. Aurora PostgreSQL unterstützt Genauigkeit auf 19 Stellen für Datentypen numerisch und dezimal in einer SEQUENCE.
Rollen auf Serverebene	Die Rolle `sysadmin` auf Serverebene wird unterstützt. Andere Rollen auf Serverebene (außer `sysadmin`) werden nicht unterstützt.
Andere Rollen auf Datenbank ebene als `db_owner`	Die Rollen `db_owner` auf Datenbankebene und die benutzerd efinierten Rollen auf Datenbankebene werden unterstützt. Andere Rollen auf Datenbankebene (als db_owner) werden nicht unterstützt.
SQL-Schlüsselwort `SPARSE`	Das Schlüsselwort SPARSE wird akzeptiert und ignoriert.
SQL Schlüsselwortklausel `ON filegroup`	Diese Klausel wird derzeit ignoriert.
SQL-Schlüsselwörter `CLUSTERED` und `NONCLUSTE RED` für Indizes und Beschränkungen	Babelfish akzeptiert und ignoriert `CLUSTERED` und `NONCLUSTE RED` -Schlüsselwörter.
`sysdatabases.cmptl evel`	`sysdatabases.cmptlevel` ist immer auf 120 eingestellt.
tempdb wird beim Neustart nicht neu initialisiert	Permanente Objekte (wie Tabellen und Prozeduren), die in tempdb erstellt wurden, werden beim Neustart der Datenbank nicht entfernt.

Funktionalität oder Syntax	Beschreibung des Verhaltens oder Unterschieds
TEXTIMAGE_ON Dateigruppe	Babelfish ignoriert die `TEXTIMAGE_ON` *`filegroup`* -Klausel.
Zeitgenauigkeit	Babelfish unterstützt eine 6-stellige Genauigkeit für Sekundenbruchteile. Bei diesem Verhalten sind keine negativen Auswirkungen zu erwarten.
Transaktionsisolierungsstufen	READUNCOMTED wird genauso behandelt wie READCOMTED.
Virtuell berechnete Spalten (nicht persistent)	Virtuell berechnete Spalten werden als persistent erstellt.
Ohne SCHEMABINDING-Klausel	Diese Klausel wird in Funktionen, Prozeduren, Triggern oder Ansichten nicht unterstützt. Das Objekt wird erstellt, aber als ob WITH SCHEMABINDING angegeben wurde.

Transaktionsisolationsstufen in Babelfish

Babelfish unterstützt die Transaktionsisolationsstufen `READ UNCOMMITTED`, `READ COMMITTED` und `SNAPSHOT`. Ab der Babelfish-Version 3.4 werden die zusätzlichen Isolationsstufen `REPEATABLE READ` und `SERIALIZABLE` unterstützt. Alle Isolationsstufen in Babelfish werden mit dem Verhalten der entsprechenden Isolationsstufen in PostgreSQL unterstützt. SQL Server und Babelfish verwenden unterschiedliche grundlegende Mechanismen zur Implementierung von Isolationsstufen für Transaktionen (Blockierung für gleichzeitigen Zugriff, Sperren aufgrund von Transaktionen, Fehlerbehandlung usw.). Und es gibt einige subtile Unterschiede darin, wie sich der gleichzeitige Zugriff für unterschiedliche Workloads auswirken kann. Weitere Informationen zu diesem PostgreSQL-Verhalten finden Sie unter Transaktionsisolation.

Themen

- Übersicht über die Transaktionsisolationsstufen
- Einrichten der Transaktionsisolationsstufen
- Aktivieren oder Deaktivieren von Transaktionsisolationsstufen
- Vergleich der Isolationsstufen in Babelfish und SQL Server

Übersicht über die Transaktionsisolationsstufen

Die ursprünglichen SQL-Server-Transaktionsisolationsstufen basieren auf pessimistischen Sperren, bei denen nur eine Kopie der Daten vorhanden ist und Abfragen Ressourcen wie Zeilen sperren müssen, bevor auf sie zugegriffen werden kann. Später wurde eine Variante der Isolationsstufe `READ COMMITTED` eingeführt. Dies ermöglicht die Verwendung von Zeilenversionen, um eine bessere Parallelität zwischen Lesern und Schreibern zu gewährleisten, indem der Zugriff nicht blockiert wird. Darüber hinaus ist eine neue Isolationsstufe namens `SNAPSHOT` verfügbar. Außerdem werden Zeilenversionen verwendet, um eine bessere Parallelität als die Isolationsstufe `REPEATABLE READ` zu gewährleisten, indem gemeinsame Sperren für gelesene Daten vermieden werden, die bis zum Ende der Transaktion aufbewahrt werden.

Im Gegensatz zu SQL Server basieren alle Transaktionsisolationsstufen in Babelfish auf optimistischen Sperren (MVCC). Jede Transaktion sieht einen Snapshot der Daten entweder zu Beginn der Anweisung (`READ COMMITTED`) oder zu Beginn der Transaktion (`REPEATABLE READ`, `SERIALIZABLE`), unabhängig vom aktuellen Status der zugrunde liegenden Daten. Daher kann sich das Ausführungsverhalten gleichzeitiger Transaktionen in Babelfish von dem in SQL Server unterscheiden.

Stellen Sie sich zum Beispiel eine Transaktion mit der Isolationsstufe `SERIALIZABLE` vor, die zunächst in SQL Server blockiert wird, aber später erfolgreich ist. Sie kann in Babelfish aufgrund eines Serialisierungskonflikts mit einer gleichzeitigen Transaktion, die dieselben Zeilen liest oder aktualisiert, fehlschlagen. Es kann auch Fälle geben, in denen die Ausführung mehrerer gleichzeitiger Transaktionen in Babelfish zu einem anderen Endergebnis führt als in SQL Server. Anwendungen, die Isolationsstufen verwenden, sollten gründlich auf Parallelitätsszenarien getestet werden.

Isolationsstufen in SQL Server	Babelfish-Isolationsstufe	PostgreSQL-Isolationsstufe	Kommentare
`READ UNCOMMITTED`	`READ UNCOMMITTED`	`READ UNCOMMITTED`	`READ UNCOMMITTED` ist in Babelfish oder PostgreSQL dasselbe wie `READ COMMITTED` .
`READ COMMITTED`	`READ COMMITTED`	`READ COMMITTED`	SQL Server `READ COMMITTED` basiert auf pessimistischen Sperren, Babelfish `READ COMMITTED` basiert auf Snapshots (MVCC).
`READ COMMITTED SNAPSHOT`	`READ COMMITTED`	`READ COMMITTED`	Beide basieren auf Snapshots (MVCC), sind aber nicht exakt identisch.
`SNAPSHOT`	`SNAPSHOT`	`REPEATABLE READ`	Genau dasselbe
`REPEATABLE READ`	`REPEATABLE READ`	`REPEATABLE READ`	SQL Server `REPEATABLE READ` basiert auf pessimistischen Sperren, Babelfish `REPEATABL`

Isolationsstufen in SQL Server	Babelfish-Isolationsstufe	PostgreSQL-Isolationsstufe	Kommentare
			E READ basiert auf Snapshots (MVCC).
SERIALIZABLE	SERIALIZABLE	SERIALIZABLE	SQL Server SERIALIZABLE basiert auf pessimistischen Isolationen, Babelfish SERIALIZABLE basiert auf Snapshots (MVCC).

> **Note**
>
> Die Tabellenhinweise werden derzeit nicht unterstützt und ihr Verhalten wird mithilfe der vordefinierten Notfalloption `escape_hatch_table_hints` von Babelfish gesteuert.

Einrichten der Transaktionsisolationsstufen

Verwenden Sie den folgenden Befehl, um eine Transaktionsisolationsstufe festzulegen:

Example

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
 SNAPSHOT | SERIALIZABLE }
```

Aktivieren oder Deaktivieren von Transaktionsisolationsstufen

Die Transaktionsisolationsstufen `REPEATABLE READ` und `SERIALIZABLE` sind in Babelfish standardmäßig deaktiviert und Sie müssen sie explizit aktivieren, indem Sie die Notfalloption `babelfishpg_tsql.isolation_level_serializable` oder `babelfishpg_tsql.isolation_level_repeatable_read` mithilfe von `sp_babelfish_configure` auf `pg_isolation` setzen. Weitere Informationen finden Sie unter Verwalten der Babelfish-Fehlerbehandlung mit Escape-Schraffuren.

Im Folgenden finden Sie Beispiele, wie Sie die Verwendung von `REPEATABLE READ` und `SERIALIZABLE` in der aktuellen Sitzung aktivieren oder deaktivieren können, indem Sie die entsprechenden Notfalloptionen setzen. Fügen Sie optional den Parameter `server` hinzu, um die Notfalloption für die aktuelle Sitzung sowie für alle nachfolgenden neuen Sitzungen festzulegen.

Zum Aktivieren der Verwendung von `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` nur in der aktuellen Sitzung

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation'
```

Zum Aktivieren der Verwendung von `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` in der aktuellen Sitzung und allen nachfolgenden neuen Sitzungen

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation',
 'server'
```

Zum Deaktivieren der Verwendung von `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ` in der aktuellen Sitzung und nachfolgenden neuen Sitzungen

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'off', 'server'
```

Zum Aktivieren der Verwendung von `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` nur in der aktuellen Sitzung

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation'
```

Zum Aktivieren der Verwendung von `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` in der aktuellen Sitzung und allen nachfolgenden neuen Sitzungen

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation', 'server'
```

Zum Deaktivieren der Verwendung von `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` in der aktuellen Sitzung und nachfolgenden neuen Sitzungen

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'off', 'server'
```

Vergleich der Isolationsstufen in Babelfish und SQL Server

Im Folgenden finden Sie einige Beispiele für die Nuancen der Implementierung der ANSI-Isolationsstufen durch SQL Server und Babelfish.

> **Note**
>
> - Die Isolationsstufen `REPEATABLE READ` und `SNAPSHOT` sind in Babelfish identisch.
> - Die Isolationsstufen `READ UNCOMMITTED` und `READ COMMITTED` sind in Babelfish identisch.

Im folgenden Beispiel wird gezeigt, wie die Basistabelle für alle unten genannten Beispiele erstellt wird:

```
CREATE TABLE employee (
    id sys.INT NOT NULL PRIMARY KEY,
    name sys.VARCHAR(255)NOT NULL,
    age sys.INT NOT NULL
);
INSERT INTO employee (id, name, age) VALUES (1, 'A', 10);
INSERT INTO employee (id, name, age) VALUES (2, 'B', 20);
INSERT INTO employee (id, name, age) VALUES (3, 'C', 30);
```

Themen

- Die Isolationsstufe READ UNCOMMITTED in Babelfish im Vergleich zur Isolationsstufe READ UNCOMMITTED in SQL Server
- Die Isolationsstufe READ COMMITTED in Babelfish im Vergleich zur Isolationsstufe READ COMMITTED in SQL Server
- Die Isolationsstufe READ COMMITTED in Babelfish im Vergleich zur Isolationsstufe READ COMMITTED SNAPSHOT in SQL Server
- Die Isolationsstufe REPEATABLE READ in Babelfish im Vergleich zur Isolationsstufe REPEATABLE READ in SQL Server
- Die Isolationsstufe SERIALIZABLE in Babelfish im Vergleich zur Isolationsstufe SERIALIZABLE in SQL Server

Die Isolationsstufe **`READ UNCOMMITTED`** in Babelfish im Vergleich zur Isolationsstufe **`READ UNCOMMITTED`** in SQL Server

Die folgende Tabelle enthält Details zu den „Dirty Reads“, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `READ UNCOMMITTED` in SQL Server im Vergleich zur Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**`READ UNCOMMITTED`** in SQL Server	**`READ UNCOMMITTED`** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`	Keine	Keine
Leerlauf in Transaktion	`UPDATE employee SET age=0;`	Update erfolgreich	Update erfolgreich
Leerlauf in Transaktion	`INSERT INTO employee VALUES (4, 'D', 40);`	Einfügen erfolgreich	Einfügen erfolgreich

Transaktion 1	Transaktion 2	**READ UNCOMMITTED** in SQL Server	**READ UNCOMMITTED** in Babelfish
`SELECT * FROM employee;`	Leerlauf in Transaktion	Transaktion 1 kann nicht festgeschriebene Änderungen aus Transaktion 2 sehen.	Wie bei `READ COMMITTED` in Babelfish. Nicht festgeschriebene Änderungen aus Transaktion 2 sind für Transaktion 1 nicht sichtbar.
Leerlauf in Transaktion	`COMMIT`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Sieht die von Transaktion 2 festgeschriebenen Änderungen	Sieht die von Transaktion 2 festgeschriebenen Änderungen

Die Isolationsstufe **`READ COMMITTED`** in Babelfish im Vergleich zur Isolationsstufe **`READ COMMITTED`** in SQL Server

Die folgende Tabelle enthält Details zum Verhalten beim Blockieren von Lese-/Schreibvorgängen, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `READ COMMITTED` in SQL Server im Vergleich zur Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**READ COMMITTED** in SQL Server	**READ COMMITTED** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	Keine	Keine

Transaktion 1	Transaktion 2	**READ COMMITTED** in SQL Server	**READ COMMITTED** in Babelfish
`SELECT * FROM employee;`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`UPDATE employee SET age=100 WHERE id = 1;`	Update erfolgreich	Update erfolgreich
`UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);`	Leerlauf in Transaktion	Der Schritt ist blockiert, bis Transaktion 2 festgeschrieben wird.	Die Änderungen an Transaktion 2 sind noch nicht sichtbar. Aktualisiert die Zeile mit id=3
Leerlauf in Transaktion	`COMMIT`	Transaktion 2 wird erfolgreich festgeschrieben. Transaktion 1 ist jetzt nicht mehr blockiert und sieht das Update von Transaktion 2.	Transaktion 2 wird erfolgreich festgeschrieben.
`SELECT * FROM employee;`	Leerlauf in Transaktion	Transaktion 1 aktualisiert die Zeile mit id = 1.	Transaktion 1 aktualisiert die Zeile mit id = 3.

Die Isolationsstufe **`READ COMMITTED`** in Babelfish im Vergleich zur Isolationsstufe **`READ COMMITTED SNAPSHOT`** in SQL Server

Die folgende Tabelle enthält Details zum Blockierverhalten der neu eingefügten Zeilen, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `READ COMMITTED SNAPSHOT` in SQL Server im Vergleich zur `READ COMMITTED`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**READ COMMITTED SNAPSHOT** in SQL Server	**READ COMMITTED** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`	Keine	Keine
`INSERT INTO employee VALUES (4, 'D', 40);`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`UPDATE employee SET age = 99;`	Der Schritt ist blockiert, bis Transaktion 1 festgeschrieben wird. Die eingefügte Zeile ist durch Transaktion 1 gesperrt.	Drei Zeilen wurden aktualisiert. Die neu eingefügte Zeile ist noch nicht sichtbar.
`COMMIT`	Leerlauf in Transaktion	Festschreiben erfolgreich. Transaktion 2 ist jetzt nicht mehr blockiert.	Festschreiben erfolgreich
Leerlauf in Transaktion	`SELECT * FROM employee;`	Alle 4 Zeilen haben age=99.	Die Zeile mit id = 4 hat den Age-Wert 40, da sie während der Update-Abfrage für Transaktion 2 nicht sichtbar war. Andere Zeilen werden auf age=99 aktualisiert.

Die Isolationsstufe **REPEATABLE READ** in Babelfish im Vergleich zur Isolationsstufe **REPEATABLE READ** in SQL Server

Die folgende Tabelle enthält Details zum Verhalten beim Blockieren von Lese-/Schreibvorgängen, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `REPEATABLE READ` in SQL Server im Vergleich zur `REPEATABLE READ`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Keine	Keine
`UPDATE employee SET name='A_TXN1' WHERE id=1;`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`SELECT * FROM employee WHERE id != 1;`	Keine	Keine
Leerlauf in Transaktion	`SELECT * FROM employee;`	Transaktion 2 ist blockiert, bis Transaktion 1 festgeschrieben wird.	Transaktion 2 wird normal fortgesetzt.
`COMMIT`	Leerlauf in Transaktion	Keine	Keine

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
Leerlauf in Transaktion	`SELECT * FROM employee;`	Das Update von Transaktion 1 ist sichtbar.	Das Update von Transaktion 1 ist nicht sichtbar.
`COMMIT`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`SELECT * FROM employee;`	Sieht das Update von Transaktion 1	Sieht das Update von Transaktion 1

Die folgende Tabelle enthält Details zum Verhalten beim Blockieren von Schreib-/Schreibvorgängen, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `REPEATABLE READ` in SQL Server im Vergleich zur `REPEATABLE READ`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	Keine	Keine
`UPDATE employee SET name='A_TXN1' WHERE id=1;`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`UPDATE employee SET name='A_T`	Transaktion 2 wurde blockiert.	Transaktion 2 wurde blockiert.

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
	`XN2' WHERE id=1;`		
`COMMIT`	Leerlauf in Transaktion	Das Festschreiben war erfolgreich und Transaktion 2 ist nicht mehr blockiert.	Das Festschreiben war erfolgreich und Transaktion 2 schlägt fehl mit dem Fehler: could not serialize access due to concurrent update.
Leerlauf in Transaktion	`COMMIT`	Festschreiben erfolgreich	Transaktion 2 wurde bereits abgebrochen.
Leerlauf in Transaktion	`SELECT * FROM employee;`	Die Zeile mit id=1 hat name='A_TX2'.	Die Zeile mit id=1 hat name='A_TX1'.

Die folgende Tabelle enthält Details zum Verhalten bei Phantomlesevorgängen, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `REPEATABLE READ` in SQL Server im Vergleich zur `REPEATABLE READ`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	Keine	Keine

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
`SELECT * FROM employee;`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`INSERT INTO employee VALUES (4, 'NewRowName', 20);`	Transaktion 2 wird ohne jegliche Blockierung fortgesetzt.	Transaktion 2 wird ohne jegliche Blockierung fortgesetzt.
Leerlauf in Transaktion	`SELECT * FROM employee;`	Die neu eingefügte Zeile ist sichtbar.	Die neu eingefügte Zeile ist sichtbar.
Leerlauf in Transaktion	`COMMIT`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Die von Transaktion 2 eingefügte neue Zeile ist sichtbar.	Die von Transaktion 2 eingefügte neue Zeile ist nicht sichtbar.
`COMMIT`	Leerlauf in Transaktion	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Die neu eingefügte Zeile ist sichtbar.	Die neu eingefügte Zeile ist sichtbar.

Die folgende Tabelle enthält Details zur Ausführung gleichzeitiger Transaktionen und zu den unterschiedlichen Endergebnissen bei Verwendung der Isolationsstufe `REPEATABLE READ` in SQL Server im Vergleich zur `REPEATABLE READ`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	Keine	Keine
`UPDATE employee SET age = 100 WHERE age IN (SELECT MIN(age) FROM employee);`	Leerlauf in Transaktion	Transaktion 1 aktualisiert die Zeile mit id 1.	Transaktion 1 aktualisiert die Zeile mit id 1.
Leerlauf in Transaktion	`UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);`	Transaktion 2 ist blockiert, da die SELECT-Anweisung versucht, Zeilen zu lesen, die durch die UPDATE-Abfrage in Transaktion 1 gesperrt wurden.	Transaktion 2 wird ohne Blockierung fortgesetzt, da der Lesevorgang nie blockiert wird, die SELECT-Anweisung ausgeführt wird und schließlich die Zeile mit id = 3 aktualisiert wird, da die Änderungen von Transaktion 1 noch nicht sichtbar sind.

Transaktion 1	Transaktion 2	**REPEATABLE READ** in SQL Server	**REPEATABLE READ** in Babelfish
Leerlauf in Transaktion	`SELECT * FROM employee;`	Dieser Schritt wird ausgeführt, nachdem Transaktion 1 festgeschrieben wurde. Die Zeile mit id = 1 wurde durch Transaktion 2 im vorherigen Schritt aktualisiert und ist hier sichtbar.	Die Zeile mit id = 3 wird durch Transaktion 2 aktualisiert.
`COMMIT`	Leerlauf in Transaktion	Transaktion 2 ist jetzt nicht mehr blockiert.	Festschreiben erfolgreich
Leerlauf in Transaktion	`COMMIT`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Beide Transaktionen führen das Update für die Zeile mit id = 1 aus.	Verschiedene Zeilen werden durch Transaktion 1 und 2 aktualisiert.

Die Isolationsstufe **SERIALIZABLE** in Babelfish im Vergleich zur Isolationsstufe **SERIALIZABLE** in SQL Server

Die folgende Tabelle enthält Details zu den Bereichssperren, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `SERIALIZABLE` in SQL Server im Vergleich zur `SERIALIZABLE`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**SERIALIZABLE** in SQL Server	**SERIALIZABLE** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine

Transaktion 1	Transaktion 2	**SERIALIZABLE** in SQL Server	**SERIALIZABLE** in Babelfish
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`INSERT INTO employee VALUES (4, 'D', 35);`	Transaktion 2 ist blockiert, bis Transaktion 1 festgeschrieben wird.	Transaktion 2 wird ohne jegliche Blockierung fortgesetzt.
Leerlauf in Transaktion	`SELECT * FROM employee;`	Keine	Keine
`COMMIT`	Leerlauf in Transaktion	Transaktion 1 wird erfolgreich festgeschrieben. Transaktion 2 ist jetzt nicht mehr blockiert.	Transaktion 1 wird erfolgreich festgeschrieben.
Leerlauf in Transaktion	`COMMIT`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Die neu eingefügte Zeile ist sichtbar.	Die neu eingefügte Zeile ist sichtbar.

Die folgende Tabelle enthält Details zur Ausführung gleichzeitiger Transaktionen und zu den unterschiedlichen Endergebnissen bei Verwendung der Isolationsstufe `SERIALIZABLE` in SQL Server im Vergleich zur `SERIALIZABLE`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**SERIALIZABLE** in SQL Server	**SERIALIZABLE** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	Keine	Keine
Leerlauf in Transaktion	`INSERT INTO employee VALUES (4, 'D', 40);`	Keine	Keine
`UPDATE employee SET age =99 WHERE id = 4;`	Leerlauf in Transaktion	Transaktion 1 ist blockiert, bis Transaktion 2 festgeschrieben wird.	Transaktion 1 wird ohne jegliche Blockierung fortgesetzt.
Leerlauf in Transaktion	`COMMIT`	Transaktion 2 wird erfolgreich festgeschrieben. Transaktion 1 ist jetzt nicht mehr blockiert.	Transaktion 2 wird erfolgreich festgeschrieben.
`COMMIT`	Leerlauf in Transaktion	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Die neu eingefügte Zeile ist mit age value = 99 sichtbar.	Die neu eingefügte Zeile ist mit age value = 40 sichtbar.

Die folgende Tabelle enthält Details zum Ausführen von `INSERT` für eine Tabelle mit eindeutiger Einschränkung. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `SERIALIZABLE` in SQL Server im Vergleich zur `SERIALIZABLE`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**SERIALIZABLE** in SQL Server	**SERIALIZABLE** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	Keine	Keine
Leerlauf in Transaktion	`INSERT INTO employee VALUES (4, 'D', 40);`	Keine	Keine
`INSERT INTO employee VALUES ((SELECT MAX(id)+1 FROM employee), 'E', 50);`	Leerlauf in Transaktion	Transaktion 1 ist blockiert, bis Transaktion 2 festgeschrieben wird.	Transaktion 1 ist blockiert, bis Transaktion 2 festgeschrieben wird.
Leerlauf in Transaktion	`COMMIT`	Transaktion 2 wird erfolgreich festgeschrieben. Transaktion 1 ist jetzt nicht mehr blockiert.	Transaktion 2 wird erfolgreich festgeschrieben. Transaktion 1 wurde abgebrochen mit dem Fehler: duplicate key value violates unique constraint.
`COMMIT`	Leerlauf in Transaktion	Transaktion 1 wird erfolgreich festgeschrieben.	Das Festschreiben von Transaktion 1 schlägt fehl mit: could not serialize access due to read or write dependencies among transactions.

Transaktion 1	Transaktion 2	**SERIALIZABLE** in SQL Server	**SERIALIZABLE** in Babelfish
`SELECT * FROM employee;`	Leerlauf in Transaktion	Die Zeile (5, 'E', 50) wird eingefügt.	Es sind nur 4 Zeilen vorhanden.

In Babelfish schlagen gleichzeitige Transaktionen, die mit der Isolationsstufe „serializable“ ausgeführt werden, mit einem Fehler aufgrund einer Serialisierungsanomalie fehl, wenn die Ausführung dieser Transaktionen nicht mit allen möglichen seriellen (einzeln ablaufenden) Ausführungen dieser Transaktionen übereinstimmt.

Die folgende Tabelle enthält Details zur Serialisierungsanomalie, wenn gleichzeitige Transaktionen ausgeführt werden. Sie zeigt die beobachteten Ergebnisse bei der Verwendung der Isolationsstufe `SERIALIZABLE` in SQL Server im Vergleich zur `SERIALIZABLE`-Babelfish-Implementierung.

Transaktion 1	Transaktion 2	**SERIALIZABLE** in SQL Server	**SERIALIZABLE** in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Keine	Keine
`UPDATE employee SET age=5 WHERE age=10;`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`SELECT * FROM employee;`	Transaktion 2 ist blockiert, bis Transaktion 1 festgeschrieben wird.	Transaktion 2 wird ohne jegliche Blockierung fortgesetzt.

Transaktion 1	Transaktion 2	**SERIALIZABLE** in SQL Server	**SERIALIZABLE** in Babelfish
Leerlauf in Transaktion	`UPDATE employee SET age=35 WHERE age=30;`	Keine	Keine
`COMMIT`	Leerlauf in Transaktion	Transaktion 1 wird erfolgreich festgeschrieben.	Transaktion 1 wird zuerst festgeschrieben und kann erfolgreich festgeschrieben werden.
Leerlauf in Transaktion	`COMMIT`	Transaktion 2 wird erfolgreich festgeschrieben.	Das Festschreiben von Transaktion 2 schlägt mit einem Serialisierungsfehler fehl. Die gesamte Transaktion wurde zurückgesetzt. Transaktion 2 wird wiederholt.
`SELECT * FROM employee;`	Leerlauf in Transaktion	Änderungen aus beiden Transaktionen sind sichtbar.	Transaktion 2 wurde zurückgesetzt. Nur Änderungen an Transaktion 1 werden gesehen.

In Babelfish ist eine Serialisierungsanomalie nur möglich, wenn alle gleichzeitigen Transaktionen auf der Isolationsstufe `SERIALIZABLE` ausgeführt werden. In der folgenden Tabelle nehmen wir das obige Beispiel, setzen aber stattdessen Transaktion 2 auf die Isolationsstufe `REPEATABLE READ`.

Transaktion 1	Transaktion 2	Isolationsstufen in SQL Server	Isolationsstufen in Babelfish
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`	Keine	Keine
`SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;`	`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;`	Keine	Keine
`SELECT * FROM employee;`	Leerlauf in Transaktion	Keine	Keine
`UPDATE employee SET age=5 WHERE age=10;`	Leerlauf in Transaktion	Keine	Keine
Leerlauf in Transaktion	`SELECT * FROM employee;`	Transaktion 2 ist blockiert, bis Transaktion 1 festgeschrieben wird.	Transaktion 2 wird ohne jegliche Blockierung fortgesetzt.
Leerlauf in Transaktion	`UPDATE employee SET age=35 WHERE age=30;`	Keine	Keine
`COMMIT`	Leerlauf in Transaktion	Transaktion 1 wird erfolgreich festgeschrieben.	Transaktion 1 wird erfolgreich festgeschrieben.
Leerlauf in Transaktion	`COMMIT`	Transaktion 2 wird erfolgreich festgeschrieben.	Transaktion 2 wird erfolgreich festgeschrieben.
`SELECT * FROM employee;`	Leerlauf in Transaktion	Änderungen aus beiden Transaktionen sind sichtbar.	Änderungen aus beiden Transaktionen sind sichtbar.

Verwenden von Babelfish-Funktionen mit eingeschränkter Implementierung

Jede neue Version von Babelfish bietet Unterstützung für weitere Funktionen, die besser auf die T-SQL-Funktionalität und das -Verhalten ausgerichtet sind. Dennoch gibt es einige nicht unterstützte Funktionen und Unterschiede in der aktuellen Implementierung. Im Folgenden finden Sie Informationen zu funktionalen Unterschieden zwischen Babelfish und T-SQL mit einigen Problemumgehungen oder Nutzungshinweisen.

Ab Version 1.2.0 von Babelfish haben die folgenden Funktionen derzeit eingeschränkte Implementierungen:

- SQL Server-Kataloge (Systemansichten) – Die Kataloge `sys.sysconfigures`, `sys.syscurconfigs` und `sys.configurations` unterstützen nur eine einzige schreibgeschützte Konfiguration. `sp_configure` wird zurzeit nicht unterstützt. Weitere Informationen zu den anderen von Babelfish implementierten SQL-Server-Ansichten finden Sie unter Abrufen von Informationen aus dem Babelfish-Systemkatalog.
- GRANT-Berechtigungen – GRANT...TO PUBLIC wird unterstützt, GRANT…TO PUBLIC WITH GRANT OPTION derzeit hingegen nicht.
- SQL-Server-Eigentumskette und Beschränkung des Berechtigungsmechanismus – In Babelfish funktioniert die SQL-Server-Eigentumskette für Ansichten, aber nicht für gespeicherte Prozeduren. Dies bedeutet, dass Prozeduren expliziten Zugriff auf andere Objekte gewährt werden muss, die demselben Eigentümer gehören wie die aufrufenden Prozeduren. In SQL Server reicht es aus, dem Aufrufer EXECUTE-Berechtigungen für die Prozedur zu erteilen, um andere Objekte aufzurufen, die demselben Eigentümer gehören. In Babelfish müssen dem Aufrufer auch Berechtigungen für die Objekte erteilt werden, auf die die Prozedur zugreift.
- Auflösung unqualifizierter (ohne Schemanamen) Objektreferenzen – Wenn ein SQL-Objekt (Prozedur, Ansicht, Funktion oder Trigger) auf ein Objekt verweist, ohne es mit einem Schemanamen zu qualifizieren, löst SQL Server den Schemanamen des Objekts unter Verwendung des Schemanamens des SQL-Objekts auf, in dem die Referenz auftritt. Derzeit weicht die Auflösung in Babelfish davon ab. Hier wird das Standardschema des Datenbankbenutzers verwendet, der die Prozedur ausführt.
- Standardschemaänderungen, Sitzungen und Verbindungen – Wenn Benutzer ihr Standardschema mit `ALTER USER...WITH DEFAULT SCHEMA` ändern, wird die Änderung in dieser Sitzung sofort wirksam. Für andere derzeit verbundenen Sitzungen, die demselben Benutzer gehören, unterscheidet sich das Timing jedoch wie folgt:
  - Für SQL Server: Die Änderung wird für diesen Benutzer sofort über alle anderen Verbindungen wirksam.

  - Für Babelfish: Die Änderung wird für diesen Benutzer nur für neue Verbindungen wirksam.
- ROWVERSION- und TIMESTAMP-Datentypimplementierung und Escape-Schraffureinstellung – Die ROWVERSION- und TIMESTAMP-Datentypen werden jetzt in Babelfish unterstützt. Um ROWVERSION oder TIMESTAMP in Babelfish zu verwenden, müssen Sie die Einstellung für die Escape-Schraffur `babelfishpg_tsql.escape_hatch_rowversion` von seinem Standardwert (strict) in `ignore` ändern. Die Babelfish-Implementierung der ROWVERSION- und TIMESTAMP-Datentypen ist meist semantisch identisch mit SQL Server, mit folgenden Ausnahmen:
  - Die integrierte @@DBTS-Funktion verhält sich ähnlich wie SQL Server, weist jedoch kleine Unterschiede auf. Anstatt den zuletzt verwendeten Wert für `SELECT @@DBTS` zurückzugeben, generiert Babelfish aufgrund der zugrunde liegenden PostgreSQL-Datenbank-Engine und seiner Multi-Version Concurrency Control (MVCC) einen neuen Zeitstempel.
  - In SQL Server erhält jede eingefügte oder aktualisierte Zeile einen eindeutigen ROWVERSION/TIMESTAMP-Wert. In Babelfish wird jeder eingefügten Zeile, die durch dieselbe Anweisung aktualisiert wurde, derselbe ROWVERSION/TIMESTAMP-Wert zugewiesen.

    Wenn sich beispielsweise eine UPDATE- oder INSERT-SELECT-Anweisung auf mehrere Zeilen auswirkt, haben die betroffenen Zeilen in SQL Server alle unterschiedliche Werte in ihrer ROWVERSION/TIMESTAMP-Spalte. In Babelfish (PostgreSQL) haben die Zeilen denselben Wert.
  - Wenn Sie in SQL Server eine neue Tabelle mit SELECT-INTO erstellen, können Sie einen expliziten Wert (wie NULL) in eine zu erstellende ROWVERSION/TIMESTAMP-Spalte umwandeln. Wenn Sie denselben Vorgang in Babelfish ausführen, wird jeder Zeile in der neuen Tabelle von Babelfish ein tatsächlicher ROWVERSION/TIMESTAMP-Wert zugewiesen.

  Diese geringfügigen Unterschiede in ROWVERSION/TIMESTAMP-Datentypen sollten keine nachteiligen Auswirkungen auf Anwendungen haben, die in Babelfish ausgeführt werden.
- Schemaerstellung, Eigentümerschaft und Berechtigungen – Berechtigungen zum Erstellen von Objekten in einem Schema, das einem Nicht-DBO-Benutzer gehört (mit `CREATE SCHEMA` *`schema name`* `AUTHORIZATION` *`user name`*), unterscheiden sich für Nicht-DBO-Benutzer von SQL Server und Babelfish, wie in der folgenden Tabelle dargestellt:

Der Datenbankbenutzer (Nicht-DBO), der Eigentümer des Schemas ist, kann folgende Vorgänge ausführen:	SQL Server	Babelfish

Der Datenbankbenutzer (Nicht-DBO), der Eigentümer des Schemas ist, kann folgende Vorgänge ausführen:	SQL Server	Babelfish
Objekte im Schema ohne zusätzliche Erteilungen durch den DBO erstellen?	Nein	Ja
Zugriff auf von DBO erstellte Objekte im Schema ohne zusätzliche Erteilungen?	Ja	Nein

- CREATE OR ALTER VIEW/ALTER VIEW-Syntax – Die Unterstützung dieser Syntax in Babelfish hat die folgenden Einschränkungen:
  - Diese Anweisungen können nicht für Ansichten verwendet werden, an die ein INSTEAD-OF-Trigger angehängt ist.
  - Diese Anweisungen können nicht für Ansichten verwendet werden, auf denen eine weitere Ansicht basiert.

Verbessern der Abfrageleistung in Babelfish

Mithilfe von Abfragehinweisen und dem PostgreSQL-Optimierer können Sie eine schnellere Abfrageverarbeitung in Babelfish erzielen.

Themen

- Verwenden eines Erläuterungsplans zur Verbesserung der Abfrageleistung von Babelfish
- Verwenden von T-SQL-Abfragehinweise zur Verbesserung der Abfrageleistung von Babelfish

Sie können die Abfrageleistung auch mithilfe der Prozedur `sp_babelfish_volatility` verbessern. Weitere Informationen finden Sie unter sp_babelfish_volatility.

Sie können die Abfrageleistung auch mithilfe der Unterabfragetransformation und des Unterabfrage-Caches verbessern. Weitere Informationen finden Sie unter Optimieren von korrelierten Unterabfragen in Aurora PostgreSQL.

Verwenden eines Erläuterungsplans zur Verbesserung der Abfrageleistung von Babelfish

Ab Version 2.1.0 enthält Babelfish zwei Funktionen, die den PostgreSQL-Optimierer transparent verwenden, um geschätzte und tatsächliche Abfragepläne für T-SQL-Abfragen am TDS-Port zu generieren. Diese Funktionen ähneln der Verwendung von SET STATISTICS PROFILE oder SET SHOWPLAN_ALL mit SQL-Server-Datenbanken, um langsam laufende Abfragen zu identifizieren und zu verbessern.

> **Note**
>
> Das Abrufen von Abfrageplänen von Funktionen, Kontrollflüssen und Cursor wird derzeit nicht unterstützt.

In der Tabelle finden Sie einen Vergleich der Funktionen zur Erläuterung des Abfrageplans in SQL Server, Babelfish und PostgreSQL.

SQL Server	Babelfish	PostgreSQL
SHOWPLAN_ALL	BABELFISH_SHOWPLAN_ALL	EXPLAIN
STATISTICS PROFILE	BABELFISH_STATISTICS PROFILE	EXPLAIN ANALYZE
Verwendet den SQL-Server-Optimierer	Verwendet den PostgreSQL-Optimierer	Verwendet den PostgreSQL-Optimierer
Ein- und Ausgabeformat von SQL Server	Eingabeformat von SQL Server und Ausgabeformat von PostgreSQL	Ein- und Ausgabeformat von PostgreSQL
Für die Sitzung einstellen	Für die Sitzung einstellen	Auf eine bestimmte Anweisung anwenden
Unterstützt Folgendes: • SELECT • INSERT	Unterstützt Folgendes: • SELECT • INSERT	Unterstützt Folgendes: • SELECT • INSERT

SQL Server	Babelfish	PostgreSQL
• AKTUALISIERUNG • DELETE • CURSOR • CREATE • EXECUTE • EXEC und Funktionen, einschließlich Kontrollfluss (CASE, WHILE-BREAK-CONTINUE, WAITFOR, BEGIN-END, IF-ELSE usw.)	• AKTUALISIERUNG • DELETE • CREATE • EXECUTE • EXEC • RAISEERROR • THROW • PRINT • USE	• AKTUALISIERUNG • DELETE • CURSOR • CREATE • EXECUTE

Verwenden Sie die Babelfish-Funktionen wie folgt:

- SET BABELFISH_SHOWPLAN_ALL [ON|OFF] – Legen Sie diese Funktion auf ON fest, um einen geschätzten Abfrageausführungsplan zu generieren. Diese Funktion implementiert das Verhalten des PostgreSQL-Befehls `EXPLAIN`. Verwenden Sie diesen Befehl, um den Erläuterungsplan für die angegebene Abfrage abzurufen.
- SET BABELFISH_STATISTICS PROFILE [ON|OFF] – Legen Sie diese Funktion auf ON fest, um tatsächliche Abfrageausführungspläne zu erhalten. Diese Funktion implementiert das Verhalten des PostgreSQL-Befehls `EXPLAIN ANALYZE`.

Weitere Informationen zu den PostgreSQL-Befehlen `EXPLAIN` und `EXPLAIN ANALYZE` finden Sie unter EXPLAIN in der PostgreSQL-Dokumentation.

> **Note**
>
> Ab Version 2.2.0 können Sie den Parameter `escape_hatch_showplan_all` auf Ignorieren einstellen, um die Verwendung des Präfixes BABELFISH_ in der SQL-Server-Syntax für die SET-Befehle `SHOWPLAN_ALL` und `STATISTICS PROFILE` zu vermeiden.

Die folgende Befehlssequenz aktiviert beispielsweise die Abfrageplanung und gibt dann einen geschätzten Abfrageausführungsplan für die SELECT-Anweisung zurück, ohne die Abfrage

auszuführen. In diesem Beispiel wird die SQL-Server-Beispieldatenbank `northwind` mit dem `sqlcmd`-Befehlszeilentool zum Abfragen des TDS-Ports verwendet:

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO

QUERY PLAN


------------------------------------------------------------------------------


Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeterritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=6231.74..6399.22 rows=66992 width=10)
  Sort Key: t.territoryid NULLS FIRST
  ->  Nested Loop  (cost=0.00..861.76 rows=66992 width=10)
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1264 width=4)
              Filter: ((territoryid)::"varchar" IS NOT NULL)
        ->  Materialize  (cost=0.00..1.79 rows=53 width=6)
              ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

Wenn Sie mit der Überprüfung und Anpassung Ihrer Abfrage fertig sind, deaktivieren Sie die Funktion wie nachfolgend gezeigt:

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

Wenn BABELFISH_STATISTICS PROFILE auf ON festgelegt ist, gibt jede ausgeführte Abfrage ihre reguläre Ergebnismenge zurück, gefolgt von einer zusätzlichen Ergebnismenge, die tatsächliche Abfrageausführungspläne anzeigt. Babelfish generiert den Abfrageplan, der die schnellste Ergebnismenge liefert, wenn die SELECT-Anweisung aufgerufen wird.

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
```

```
2> dbo.employeeterritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

Die Ergebnismenge und der Abfrageplan werden zurückgegeben (dieses Beispiel zeigt nur den Abfrageplan).

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT e.employeeid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
  Sort Key: t.territoryid NULLS FIRST


  ->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
       Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
             ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

Weitere Informationen darüber, wie Sie Ihre Abfragen und die vom PostgreSQL-Optimierer zurückgegebenen Ergebnisse analysieren können, finden Sie unter explain.depesz.com. Weitere Informationen zu den PostgreSQL-Befehlen EXPLAIN und EXPLAIN ANALYZE finden Sie unter EXPLAIN in der PostgreSQL-Dokumentation.

Parameter, die Erläuterungsoptionen von Babelfish steuern

Sie können die in der folgenden Tabelle aufgeführten Parameter verwenden, um die Art der Informationen zu steuern, die in Ihrem Abfrageplan angezeigt werden.

Parameter	Beschreibung
babelfishpg_tsql.explain_buffers	Ein boolescher Wert, der Informationen zur Puffernutzung für den Optimierer ein- und ausschaltet. (Standardwert: off) (Zulässig: off, on)

Parameter	Beschreibung
babelfishpg_tsql.explain_costs	Ein boolescher Wert, der geschätzte Start- und Gesamtkosteninformationen für den Optimiere r ein- und ausschaltet. (Standardwert: on) (Zulässig: off, on)
babelfishpg_tsql.explain_format	Gibt das Ausgabeformat für den `EXPLAIN`-Plan an. (Standardwert: text) (Zulässig: text, xml, json, yaml)
babelfishpg_tsql.explain_settings	Ein boolescher Wert, der die Aufnahme von Informationen zu Konfigurationsparametern in die EXPLAIN-Planausgabe ein- oder ausschalt et. (Standardwert: off) (Zulässig: off, on)
babelfishpg_tsql.explain_summary	Ein boolescher Wert, der zusammenfassende Informationen wie die Gesamtzeit nach dem Abfrageplan ein- oder ausschaltet. (Standard wert: on) (Zulässig: off, on)
babelfishpg_tsql.explain_timing	Ein boolescher Wert, der die tatsächliche Startzeit und die Zeit, die in jedem Knoten in der Ausgabe verbracht wird, ein- oder ausschaltet. (Standardwert: on) (Zulässig: off, on)
babelfishpg_tsql.explain_verbose	Ein boolescher Wert, der die detaillierteste Version eines Erläuterungsplans ein- oder ausschaltet. (Standardwert: off) (Zulässig: off, on)
babelfishpg_tsql.explain_wal	Ein boolescher Wert, der die Generierung von WAL-Datensatzinformationen im Rahmen eines Erläuterungsplans aktiviert (oder deaktiviert). (Standardwert: off) (Zulässig: off, on)

Sie können die Werte aller Babelfish-bezogenen Parameter in Ihrem System überprüfen, indem Sie entweder den PostgreSQL-Client oder den SQL-Server-Client verwenden. Führen Sie den folgenden Befehl aus, um Ihre aktuellen Parameterwerte abzurufen:

```
1> execute sp_babelfish_configure '%explain%';
2> GO
```

In der folgenden Ausgabe sehen Sie, dass alle Einstellungen für diesen speziellen Babelfish-DB-Cluster auf ihre Standardwerte festgelegt sind. In diesem Beispiel wird nicht die gesamte Ausgabe gezeigt.

```
             name                    setting                     short_desc
--------------------------------- --------
 ---------------------------------------------------------
babelfishpg_tsql.explain_buffers   off      Include information on buffer usage
babelfishpg_tsql.explain_costs     on       Include information on estimated startup
 and total cost
babelfishpg_tsql.explain_format    text     Specify the output format, which can be
 TEXT, XML, JSON, or YAML
babelfishpg_tsql.explain_settings  off      Include information on configuration
 parameters
babelfishpg_tsql.explain_summary   on       Include summary information (e.g.,totaled
 timing information) after the query plan
babelfishpg_tsql.explain_timing    on       Include actual startup time and time spent
 in each node in the output
babelfishpg_tsql.explain_verbose   off      Display additional information regarding
 the plan
babelfishpg_tsql.explain_wal       off      Include information on WAL record
 generation

(8 rows affected)
```

Sie können die Einstellung für diese Parameter mit `sp_babelfish_configure` ändern, wie im folgenden Beispiel veranschaulicht.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on';
2> GO
```

Wenn Sie die Einstellung auf einer clusterweiten Ebene dauerhaft festlegen möchten, schließen Sie das Schlüsselwort Server ein, wie im folgenden Beispiel gezeigt.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on', 'server';
2> GO
```

Verwenden von T-SQL-Abfragehinweise zur Verbesserung der Abfrageleistung von Babelfish

Ab Version 2.3.0 unterstützt Babelfish die Verwendung von Abfragehinweisen mit `pg_hint_plan`. In Aurora PostgreSQL ist `pg_hint_plan` standardmäßig installiert. Weitere Informationen zur PostgreSQL-Erweiterung `pg_hint_plan` finden Sie unter https://github.com/ossc-db/pg_hint_plan. Weitere Informationen zu der Version dieser Erweiterung, die von Aurora PostgreSQL unterstützt wird, finden Sie unter Versionen der Erweiterungen für Amazon Aurora PostgreSQL in den Versionshinweisen für Aurora PostgreSQL.

Mit dem Abfrageoptimierer kann für jede SQL-Anweisung der bestmögliche Ausführungsplans ermittelt werden. Bei der Auswahl eines Plans berücksichtigt der Abfrageoptimierer sowohl das Kostenmodell der Engine als auch die Spalten- und Tabellenstatistiken. Der vorgeschlagene Plan entspricht jedoch möglicherweise nicht den Anforderungen Ihrer Datensätze. Daher werden mit Abfragehinweisen die Leistungsprobleme angesprochen, um die Ausführungspläne zu verbessern. Ein `query hint` ist Syntax, die dem SQL-Standard hinzugefügt wurde und die Datenbank-Engine anweist, wie die Abfrage auszuführen ist. Beispielsweise kann ein Hinweis die Engine anweisen, einem sequenziellen Scan zu folgen und jeden Plan zu überschreiben, den der Abfrageoptimierer ausgewählt hat.

Aktivieren von T-SQL-Abfragehinweisen in Babelfish

Derzeit ignoriert Babelfish standardmäßig alle T-SQL-Hinweise. Wenn Sie T-SQL-Hinweise anwenden möchten, führen Sie den Befehl `sp_babelfish_configure` mit dem Wert enable_pg_hint auf ON (EIN) aus.

```
EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on' [, 'server']
```

Sie können die Einstellungen auf einer clusterweiten Ebene dauerhaft festlegen, indem Sie das Schlüsselwort Server einschließen. Verwenden Sie „Server“ nicht, wenn Sie die Einstellung nur für die aktuelle Sitzung festlegen möchten.

Nach Einstellung von `enable_pg_hint` auf ON (EIN) wendet Babelfish die folgenden T-SQL-Hinweise an.

- INDEX-Hinweise

- JOIN-Hinweise
- FORCE ORDER-Hinweis
- MAXDOP-Hinweis

Mit der folgenden Befehlssequenz wird beispielsweise `pg_hint_plan` aktiviert.

```
1> CREATE TABLE t1 (a1 INT PRIMARY KEY, b1 INT);
2> CREATE TABLE t2 (a2 INT PRIMARY KEY, b2 INT);
3> GO
1> EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on';
2> GO
1> SET BABELFISH_SHOWPLAN_ALL ON;
2> GO
1> SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2; --NO HINTS (HASH JOIN)
2> GO
```

Auf die SELECT-Anweisung wird kein Hinweis angewendet. Der Abfrageplan wird ohne Hinweis zurückgegeben.

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2
Hash Join (cost=60.85..99.39 rows=2260 width=16)
 Hash Cond: (t1.a1 = t2.a2)
 -> Seq Scan on t1 (cost=0.00..32.60 rows=2260 width=8)
 -> Hash (cost=32.60..32.60 rows=2260 width=8)
 -> Seq Scan on t2 (cost=0.00..32.60 rows=2260 width=8)
```

```
1> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.a1 = t2.a2;
2> GO
```

Der Abfragehinweis wird auf die SELECT-Anweisung angewendet. Die folgende Ausgabe zeigt, dass der Abfrageplan mit Merge Join zurückgegeben wird.

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT/*+ MergeJoin(t1 t2) Leading(t1 t2)*/ * FROM t1 INNER JOIN t2 ON
 t1.a1 = t2.a2
Merge Join (cost=0.31..190.01 rows=2260 width=16)
 Merge Cond: (t1.a1 = t2.a2)
 -> Index Scan using t1_pkey on t1 (cost=0.15..78.06 rows=2260 width=8)
 -> Index Scan using t2_pkey on t2 (cost=0.15..78.06 rows=2260 width=8)
```

```
1> SET BABELFISH_SHOWPLAN_ALL OFF;
2> GO
```

Einschränkungen

Beachten Sie bei der Verwendung der Abfragehinweise die folgenden Einschränkungen:

- Wenn ein Abfrageplan zwischengespeichert wird, bevor `enable_pg_hint` aktiviert wird, werden in derselben Sitzung keine Hinweise angewendet. Sie werden in der neuen Sitzung angewendet.
- Wenn Schemanamen explizit angegeben werden, können keine Hinweise angewendet werden. Sie können Tabellenaliase als Problemumgehung verwenden.
- Ein Abfragehinweis kann nicht auf Ansichten und Unterabfragen angewendet werden.
- Hinweise funktionieren nicht für UPDATE/DELETE-Anweisungen mit JOINs.
- Ein Indexhinweis für einen nicht existierenden Index oder eine Tabelle wird ignoriert.
- Der FORCE ORDER-Hinweis funktioniert nicht für HASH JOINs und Nicht-ANSI JOINS.

Verwenden von Aurora PostgreSQL-Erweiterungen mit Babelfish

Aurora PostgreSQL bietet Erweiterungen für die Arbeit mit anderen AWS Diensten. Dies sind optionale Erweiterungen, die verschiedene Anwendungsfälle unterstützen, z. B. die Verwendung von Amazon S3 mit Ihrem DB-Cluster zum Importieren oder Exportieren von Daten.

- Wenn Sie Daten aus einem Amazon S3 Bucket in Ihren Babelfish-DB-Cluster importieren möchten, richten Sie die Aurora-PostgreSQL-Erweiterung `aws_s3` ein. Mit dieser Erweiterung können Sie auch Daten aus Ihrem Aurora-PostgreSQL-DB-Cluster in einen Amazon S3 Bucket exportieren.

- AWS Lambda ist ein Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten. Sie können Lambda-Funktionen verwenden, um zum Beispiel Ereignisbenachrichtigungen von Ihrer DB-Instance zu verarbeiten. Weitere Informationen zu Lambda finden Sie unter Was ist AWS Lambda? im AWS Lambda -Entwicklerhandbuch. Zum Aufrufen von Lambda-Funktionen aus Ihrem Babelfish-DB-Cluster richten Sie die Aurora-PostgreSQL-Erweiterung `aws_lambda` ein.

Um diese Erweiterungen für Ihren Babelfish-Cluster einzurichten, müssen Sie zuerst dem internen Babelfish-Benutzer die Erlaubnis erteilen, die Erweiterungen zu laden. Nachdem Sie die Berechtigung erteilt haben, können Sie dann Aurora-PostgreSQL-Erweiterungen laden.

Aktivieren von Aurora-PostgreSQL-Erweiterungen in Ihrem Babelfish-DB-Cluster

Bevor Sie die `aws_s3`- oder `aws_lambda`-Erweiterungen laden können, müssen Sie Ihrem Babelfish-DB-Cluster die erforderlichen Berechtigungen gewähren.

In der folgenden Vorgehensweise wird das `psql`-PostgreSQL-Befehlszeilen-Tool für die Verbindung mit dem DB-Cluster verwendet. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit Ihrem psql. Sie können auch pgAdmin verwenden. Details hierzu finden Sie unter Herstellen einer Verbindung mit dem DB-Cluster mit pgAdmin.

Dieses Verfahren lädt sowohl `aws_s3` als auch `aws_lambda` nacheinander. Sie müssen nicht beides laden, wenn Sie nur eine dieser Erweiterungen verwenden möchten. Die `aws_commons`-Erweiterung wird von beiden Optionen benötigt und wird standardmäßig geladen, wie in der Ausgabe gezeigt.

Richten Sie Babelfish-DB-Cluster mit Berechtigungen für die Aurora-PostgreSQL-Erweiterungen ein

1. Stellen Sie eine Verbindung mit Ihrem Babelfish-DB-Cluster her. Verwenden Sie den Namen für den „Hauptbenutzer“ (-U), den Sie beim Erstellen des Babelfish-DB-Clusters angegeben haben. In den Beispielen wird der Standard (`postgres`) dargestellt.

   Für Linux, macOS oder Unix:

   ```
   psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com \
   -U postgres \
   -d babelfish_db \
   -p 5432
   ```

   Für Windows:

```
psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com ^
-U postgres ^
-d babelfish_db ^
-p 5432
```

Der Befehl antwortet mit einer Aufforderung zur Eingabe des Passworts für den Benutzernamen (-U).

```
Password:
```

Geben Sie das Passwort für den Benutzernamen (-U) für den DB-Cluster ein. Wenn die Verbindung erfolgreich ist, wird eine Ausgabe ähnlich der Folgenden angezeigt.

```
psql (13.4)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

2. Erteilen Sie dem internen Babelfish-Benutzer Berechtigungen zum Erstellen und Laden von Erweiterungen.

```
babelfish_db=> GRANT rds_superuser TO master_dbo;
GRANT ROLE
```

3. Erstellen und laden Sie die `aws_s3`-Erweiterung. Die `aws_commons`-Erweiterung ist erforderlich und wird automatisch mit `aws_s3` installiert.

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. Erstellen und laden Sie die `aws_lambda`-Erweiterung.

```
babelfish_db=> create extension aws_lambda cascade;
CREATE EXTENSION
babelfish_db=>
```

Verwenden von Babelfish mit Amazon S3

Wenn Sie noch keinen Amazon-S3-Bucket für Ihren Babelfish-DB-Cluster haben, können Sie einen erstellen. Gewähren Sie Zugriff für jeden Amazon-S3-Bucket, den Sie verwenden möchten.

Bevor Sie versuchen, Daten mit einem Amazon-S3-Bucket zu importieren oder zu exportieren, führen Sie die folgenden einmaligen Schritte aus.

Gewähren Sie Ihrer Babelfish-DB-Instance Zugriff auf Ihren Amazon-S3-Bucket

1. Erstellen Sie bei Bedarf einen Amazon-S3-Bucket für Ihre Babelfish-Instance. Befolgen Sie dazu die Anleitungen unter Erstellen eines Buckets im Benutzerhandbuch zum Amazon Simple Storage Service guide-s3-user;.
2. Laden Sie Dateien in Ihren Amazon-S3-Bucket hoch. Befolgen Sie dazu die Schritte unter Hinzufügen eines Objekts zu einem Bucket im Benutzerhandbuch zum Amazon Simple Storage Service guide-s3-user;.
3. Richten Sie Berechtigungen nach Bedarf ein:

   - Zum Importieren von Daten aus Amazon S3 benötigt der Babelfish-DB-Cluster die Berechtigung für den Zugriff auf den Bucket. Wir empfehlen die Verwendung einer AWS Identity and Access Management (IAM-) Rolle und das Anhängen einer IAM-Richtlinie an diese Rolle für Ihren Cluster. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Verwenden einer IAM-Rolle für den Zugriff auf einen Amazon S3-Bucket.
   - Wenn Sie Daten aus Ihrem Babelfish-DB-Cluster exportieren möchten, muss Ihrem Cluster Zugriff auf den Amazon S3 Bucket gewährt werden. Wie beim Importieren empfehlen wir die Verwendung einer IAM-Rolle und -Richtlinie. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Einrichten des Zugriffs auf einen Amazon-S3-Bucket.

Sie können Amazon S3 jetzt mit der `aws_s3`-Erweiterung mit Ihrem Babelfish-DB-Cluster verwenden.

Importieren von Daten aus Amazon S3 nach Babelfish und exportieren von Babelfish-Daten nach Amazon S3

1. Verwenden Sie die `aws_s3`-Erweiterung mit Ihrem Babellfish-DB-Cluster.

   Verweisen Sie in diesem Fall auf die Tabellen, wie sie in PostgreSQL existieren. Das heißt, beim Import in eine Babelfish-Tabelle namens `[database].[schema].[tableA]` bezeichnen Sie diese Tabelle in der `aws_s3`-Funktion als `database_schema_tableA`:

- Ein Beispiel für die Verwendung einer `aws_s3`-Funktion zum Importieren von Daten finden Sie unter Importieren von Daten aus Amazon S3 in Ihren Aurora PostgreSQL DB-Cluster.
- Beispiele für die Verwendung von `aws_s3`-Funktionen zum Exportieren von Daten finden Sie unter Exportieren von Abfragedaten mithilfe der Funktion aws_s3.query_export_to_s3.

2. Verweisen Sie mit der PostgreSQL-Benennung auf Babelfish-Tabellen, wenn Sie die `aws_s3`-Erweiterung und Amazon S3 wie in folgender Tabelle verwenden.

Babelfish-Tabelle	Aurora PostgreSQL-Tabelle
`database``.`*`schema`*`.`*`table`*	*`database`*`_`*`schema`*`_`*`table`*

Weitere Informationen zur Verwendung von Amazon S3 mit Aurora PostgreSQL finden Sie unter Importieren von Amazon S3 in einen Aurora-PostgreSQL-DB-Cluster und Exportieren von Daten aus einem/einer Aurora PostgreSQL-DB-Cluster zu Amazon S3.

Verwenden Sie Babelfish mit AWS Lambda

Nach dem die `aws_lambda`-Erweiterung in Ihrem Babelfish-DB-Cluster geladen ist und bevor Sie Lambda-Funktionen aufrufen, gewähren Sie Lambda wie folgt Zugriff auf Ihren DB-Cluster.

Richten Sie den Zugriff für Ihren Babelfish-DB-Cluster zur Funktion mit Lambda ein

Dieses Verfahren verwendet die AWS CLI , um die IAM-Richtlinie und -Rolle zu erstellen und diese dem Babelfish-DB-Cluster zuzuordnen.

1. Erstellen Sie eine IAM-Richtlinie, die den Zugriff auf Lambda von Ihrem Babelfish-DB-Cluster aus erlaubt.

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
 "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
```

```
    ]
}'
```

2. Erstellen Sie eine IAM-Rolle, die die Richtlinie zur Laufzeit annehmen kann.

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. Fügen Sie der Rolle die -Richtlinie an.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. Hängen Sie die Rolle an Ihren Babelfish-DB-Cluster an.

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

Nachdem Sie diese Aufgaben durchgeführt haben, können Sie Ihre Lambda-Funktionen aufrufen. Weitere Informationen und Beispiele für die Einrichtung AWS Lambda des Aurora PostgreSQL-DB-Clusters mit AWS Lambda finden Sie unter. Schritt 2: Konfigurieren Sie IAM für Ihren Aurora PostgreSQL-DB-Cluster, und AWS Lambda

Aufrufen einer Lambda-Funktion aus Ihrem Babelfish-DB-Cluster

AWS Lambda unterstützt Funktionen, die in Java, Node.js, Python, Ruby und anderen Sprachen geschrieben wurden. Wenn die Funktion beim Aufruf Text zurückgibt, können Sie sie von Ihrem

Babelfish-DB-Cluster aus aufrufen. Das folgende Beispiel ist eine Platzhalter-Python-Funktion, die eine Begrüßung zurückgibt.

```
lambda_function.py
import json
def lambda_handler(event, context):
    #TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
```

Derzeit wird JSON von Babelfish nicht unterstützt. Wenn Ihre Funktion JSON zurückgibt, verwenden Sie einen Wrapper, um den JSON-Code zu behandeln. Angenommen, die vorher gezeigte `lambda_function.py` ist in Lambda als `my-function` gespeichert.

1. Stellen Sie mit Hilfe des `psql`-Clients (oder des pgAdmin-Clients) eine Verbindung zu Ihrem Babelfish-DB-Cluster her. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit Ihrem psql.
2. Erstellen Sie den Wrapper. In diesem Beispiel wird die prozedurale Sprache von PostgreSQL für SQL verwendet, `PL/pgSQL`. Weitere Informationen hierzu finden Sie unter PL/pgSQL-SQL Procedural Language.

   ```
   create or replace function master_dbo.lambda_wrapper()
   returns text
   language plpgsql
   as
   $$
   declare
      r_status_code integer;
      r_payload text;
   begin
      SELECT payload INTO r_payload
        FROM aws_lambda.invoke(  aws_commons.create_lambda_function_arn('my-function',
    'us-east-1')
                               ,'{"body": "Hello from Postgres!"}'::json );
      return r_payload ;
   end;
   $$;
   ```

Die Funktion kann jetzt vom Babelfish-TDS-Port (1433) oder vom PostgreSQL-Port (5433) aus ausgeführt werden.

a. So rufen Sie diese Funktion von Ihrem PostgreSQL-Port aus auf:

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}'::json );
```

Die Ausgabe sieht folgendermaßen oder ähnlich aus:

```
status_code |                         payload                          |
 executed_version | log_result
-------------+----------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
         |
(1 row)
```

b. Um diese Funktion vom TDS-Port aus aufzurufen, stellen Sie eine Verbindung mit dem Port über den `sqlcmd`-Befehlszeilen-Client von SQL Server her. Details hierzu finden Sie unter Verbinden mit Ihrem DB-Cluster mithilfe eines SQL Server-Clients. Wenn eine Verbindung hergestellt ist, führen Sie Folgendes aus:

```
1> select lambda_wrapper();
2> go
```

Daraufhin erhalten Sie ein Ergebnis, das dem hier dargestellten entspricht:

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}
```

Weitere Informationen zur Verwendung von Lambda mit Aurora PostgreSQL finden Sie unter Aufrufen einer AWS Lambda Funktion aus einem Aurora PostgreSQL-DB-Cluster (PostgreSQL-DB-Instance). Weitere Informationen über die Nutzung von Lambda-Funktionen finden Sie unter Erste Schritte mit Lambda im AWS Lambda -Entwicklerhandbuch.

Verwenden von pg_stat_statements in Babelfish

Babelfish für Aurora PostgreSQL unterstützt die `pg_stat_statements`-Erweiterung ab 3.3.0. Weitere Informationen finden Sie unter pg_stat_statements.

Einzelheiten zur von Aurora PostgreSQL unterstützten Version dieser Erweiterung finden Sie unter Erweiterungsversionen.

Erstellen der pg_stat_statements-Erweiterung

Zum Aktivieren von `pg_stat_statements` müssen Sie die Berechnung der Abfrage-ID aktivieren. Dies erfolgt automatisch, wenn `compute_query_id` auf `on` oder `auto` in der Parametergruppe gesetzt ist. Der Standardwert für den `compute_query_id`-Parameter ist `auto`. Sie müssen auch diese Erweiterung erstellen, um diese Funktion zu aktivieren. Verwenden Sie den folgenden Befehl, um die Erweiterung vom T-SQL-Endpunkt aus zu installieren:

```
1>EXEC sp_execute_postgresql 'CREATE EXTENSION pg_stat_statements WITH SCHEMA sys';
```

Sie können mit der folgenden Abfrage auf die Abfragestatistiken zugreifen:

```
postgres=>select * from pg_stat_statements;
```

> **Note**
>
> Wenn Sie während der Installation den Schemanamen für die Erweiterung nicht angeben, wird dieser standardmäßig im öffentlichen Schema erstellt. Um darauf zuzugreifen, müssen Sie eckige Klammern mit Schemaqualifizierer verwenden, wie unten gezeigt:

```
postgres=>select * from [public].pg_stat_statements;
```

Sie können die Erweiterung auch vom PSQL-Endpunkt aus erstellen.

Autorisieren der Erweiterung

Standardmäßig können Sie die Statistiken für Abfragen sehen, die in Ihrer T-SQL-Datenbank ausgeführt wurden, ohne dass eine Autorisierung erforderlich ist.

Um auf Abfragestatistiken zugreifen zu können, die von anderen erstellt wurden, benötigen Sie eine `pg_read_all_stats`-PostgreSQL-Rolle. Folgen Sie den unten genannten Schritten, um den Befehl GRANT pg_read_all_stats zu erstellen.

1. Verwenden Sie in T-SQL die folgende Abfrage, die den internen PG-Rollennamen zurückgibt.

   ```
   SELECT rolname FROM pg_roles WHERE oid = USER_ID();
   ```

2. Stellen Sie mit der Berechtigung rds_superuser eine Verbindung mit der Babelfish-für-Aurora-PostgreSQL-Datenbank her und verwenden Sie den folgenden Befehl:

   ```
   GRANT pg_read_all_stats TO <rolname_from_above_query>
   ```

Beispiel

Vom T-SQL-Endpunkt aus:

```
1>SELECT rolname FROM pg_roles WHERE oid = USER_ID();
2>go
```

```

rolname
-------
master_dbo
(1 rows affected)
```

Vom PSQL-Endpunkt aus:

```
babelfish_db=# grant pg_read_all_stats to master_dbo;
```

```
GRANT ROLE
```

Sie können mit der Ansicht pg_stat_statements auf die Abfragestatistiken zugreifen:

```
1>create table t1(cola int);
2>go
1>insert into t1 values (1),(2),(3);
2>go
```

```
(3 rows affected)
```

```
1>select userid, dbid, queryid, query from pg_stat_statements;
2>go
```

```
userid dbid queryid             query
------ ---- -------             -----
37503 34582 6487973085327558478 select * from t1
37503 34582 6284378402749466286 SET QUOTED_IDENTIFIER OFF
37503 34582 2864302298511657420 insert into t1 values ($1),($2),($3)
10    34582 NULL                <insufficient privilege>
37503 34582 5615368793313871642 SET TEXTSIZE 4096
37503 34582 639400815330803392  create table t1(cola int)
(6 rows affected)
```

Zurücksetzen der Abfragestatistiken

Sie können `pg_stat_statements_reset()` verwenden, um die bisher von pg_stat_statements erfassten Statistiken zurückzusetzen. Weitere Informationen finden Sie unter pg_stat_statements. Dies wird derzeit nur über den PSQL-Endpunkt unterstützt. Stellen Sie mit der `rds_superuser`-Berechtigung eine Verbindung zu Babelfish für Aurora PostgreSQL her und verwenden Sie den folgenden Befehl:

```
SELECT pg_stat_statements_reset();
```

Einschränkungen

- Derzeit wird `pg_stat_statements()` nicht über den T-SQL-Endpunkt unterstützt. `pg_stat_statements`-View ist die empfohlene Methode zum Erfassen von Statistiken.

- Einige der Abfragen werden möglicherweise vom T-SQL-Parser neu geschrieben, der von der Aurora PostgreSQL-Engine implementiert wurde. In der `pg_stat_statements`-Ansicht wird die neu geschriebene Abfrage und nicht die ursprüngliche Abfrage angezeigt.

  Beispiel

  ```
  select next value for [dbo].[newCounter];
  ```

  Die obige Abfrage wird in der Ansicht pg_stat_statements wie folgt umgeschrieben.

  ```
  select nextval($1);
  ```

- Aufgrund des Ausführungsablaufs der Anweisungen werden einige Abfragen möglicherweise nicht von pg_stat_statements verfolgt und sind daher in der Ansicht nicht sichtbar. Dazu gehören die folgenden Anweisungen: `use dbname`, `goto`, `print`, `raise error`, `set`, `throw`, `declare cursor`.
- Bei den Anweisungen CREATE LOGIN und ALTER LOGIN werden Abfrage und Abfrage-ID nicht angezeigt. Es werden unzureichende Berechtigungen angezeigt.
- Die `pg_stat_statements`-Ansicht enthält immer die folgenden zwei Einträge, da diese intern vom `sqlcmd`-Client ausgeführt werden.
  - SET QUOTED_IDENTIFIER OFF
  - SET TEXTSIZE 4096

Verwenden von pgvector in Babelfish

Mit pgvector, einer Open-Source-Erweiterung, können Sie direkt in Ihrer Postgres-Datenbank nach ähnlichen Daten suchen. Babelfish unterstützt diese Erweiterung jetzt ab den Versionen 15.6 und 16.2. Weitere Informationen finden Sie in der Open-Source-Dokumentation von pgvector.

Voraussetzungen

Um die pgvector-Funktion zu aktivieren, installieren Sie die Erweiterung im sys-Schema mit einer der folgenden Methoden:

- Führen Sie den folgenden Befehl im sqlcmd-Client aus:

  ```
  exec sys.sp_execute_postgresql 'CREATE EXTENSION vector WITH SCHEMA sys';
  ```

- Stellen Sie eine Verbindung mit `babelfish_db` her und führen Sie den folgenden Befehl im psql-Client aus:

```
CREATE EXTENSION vector WITH SCHEMA sys;
```

> **Note**
>
> Nach der Installation der pgvector-Erweiterung ist der Vektordatentyp nur in neuen Datenbankverbindungen verfügbar, die Sie herstellen. Bestehende Verbindungen erkennen den neuen Datentyp nicht.

Unterstützte Funktionen

Babelfish erweitert die T-SQL-Funktionen, um Folgendes zu unterstützen:

- Speicherung

  Babelfish unterstützt jetzt eine mit Vektordatentypen kompatible Syntax und verbessert so die T-SQL-Kompatibilität. Weitere Informationen zur Speicherung von Daten mit pgvector finden Sie unter Speicherung.

- Abfrage

  Babelfish erweitert die Unterstützung von T-SQL-Ausdrücken um Operatoren zur Vektorähnlichkeit. Für alle anderen Abfragen ist jedoch weiterhin die standardmäßige T-SQL-Syntax erforderlich.

  > **Note**
  >
  > T-SQL unterstützt keinen Array-Typ und die Datenbanktreiber bieten keine Schnittstelle zu deren Verarbeitung. Zur Umgehung dieses Problems verwendet Babelfish Textzeichenfolgen (varchar/nvarchar) zum Speichern von Vektordaten. Wenn Sie beispielsweise einen Vektorwert [1,2,3] anfordern, gibt Babelfish als Antwort die Zeichenfolge '[1,2,3]' zurück. Sie können diese Zeichenfolge auf Anwendungsebene nach Ihren Anforderungen analysieren und aufteilen.

  Weitere Informationen zur Abfrage von Daten mit pgvector finden Sie unter Abfrage.

- Indizierung

  T-SQL-`Create Index` unterstützt jetzt die Syntax `USING INDEX_METHOD`. Sie können jetzt einen Operator zur Ähnlichkeitssuche definieren, der bei der Erstellung eines Indexes für eine bestimmte Spalte verwendet wird.

  Die Grammatik wurde auch erweitert, um Operationen zur Vektorähnlichkeit für die benötigte Spalte zu unterstützen (Überprüfen Sie die Grammatik column_name_list_with_order_for_vector.).

```
CREATE [UNIQUE] [clustered] [COLUMNSTORE] INDEX <index_name> ON <table_name> [USING
 vector_index_method] (<column_name_list_with_order_for_vector>)
Where column_name_list_with_order_for_vector is:
    <column_name> [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS | VECTOR_L2_OPS]
 (COMMA simple_column_name [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS |
 VECTOR_L2_OPS])
```

  Weitere Informationen zur Indizierung von Daten mit pgvector finden Sie unter Indizierung.

- Leistung
  - Verwenden Sie `SET BABELFISH_STATISTICS PROFILE ON`, um Abfragepläne vom T-SQL-Endpunkt aus zu debuggen.
  - Erhöhen Sie `max_parallel_workers_get_gather` mithilfe der in T-SQL unterstützten `set_config`-Funktion.
  - Verwenden Sie `IVFFlat` für ähnliche Suchen. Weitere Informationen finden Sie unter IVFFlat.

  Informationen zur Verbesserung der Leistung mit pgvector finden Sie unter Leistung.

Einschränkungen

- Babelfish unterstützt keine Volltextsuche für die Hybridsuche. Weitere Informationen finden Sie unter Hybridsuche.
- Babelfish unterstützt derzeit keine Neuindizierungsfunktionen. Sie können jedoch weiterhin den PostgreSQL-Endpunkt für die Neuindizierung verwenden. Weitere Informationen finden Sie unter Bereinigung.

Verwenden von Amazon Aurora Machine Learning mit Babelfish

Sie können die Funktionen Ihres DB-Clusters von Babelfish für Aurora PostgreSQL erweitern, indem Sie ihn in Amazon Aurora Machine Learning integrieren. Diese nahtlose Integration gewährt Ihnen Zugriff auf eine Reihe leistungsstarker Dienste wie Amazon Comprehend oder Amazon SageMaker AI oder Amazon Bedrock, die jeweils auf unterschiedliche Anforderungen im Bereich maschinelles Lernen zugeschnitten sind.

Als Babelfish-Benutzer können Sie bei der Arbeit mit Aurora Machine Learning vorhandenes Wissen über T-SQL-Syntax und -Semantik nutzen. Folgen Sie den Anweisungen in der AWS Dokumentation für Aurora PostgreSQL. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora PostgreSQL.

Voraussetzungen

- Bevor Sie versuchen, Ihren DB-Cluster von Babelfish für Aurora PostgreSQL für die Verwendung von Aurora Machine Learning einzurichten, müssen Sie die zugehörigen Anforderungen und Voraussetzungen verstehen. Weitere Informationen finden Sie unter Voraussetzungen für die Verwendung von Aurora Machine Learning mit Aurora PostgreSQL.
- Stellen Sie sicher, dass Sie die `aws_ml`-Erweiterung entweder mithilfe des Postgres-Endpunkts oder der gespeicherten Prozedur `sp_execute_postgresql` installieren.

```
exec sys.sp_execute_postgresql 'Create Extension aws_ml'
```

> **Note**
>
> Derzeit unterstützt Babelfish keine Kaskadenoperationen mit `sp_execute_postgresql` in Babelfish. Da sich `aws_ml` auf `aws_commons` stützt, müssen Sie es separat mithilfe des Postgres-Endpunkts installieren.

```
create extension aws_common;
```

Umgang mit T-SQL-Syntax und -Semantik mit **`aws_ml`**-Funktionen

In den folgenden Beispielen wird erklärt, wie T-SQL-Syntax und -Semantik auf die Amazon-ML-Services angewendet werden:

Example : aws_bedrock.invoke_model – Eine einfache Abfrage mit Amazon-Bedrock-Funktionen

```
aws_bedrock.invoke_model(
   model_id      varchar,
   content_type  text,
   accept_type   text,
   model_input   text)
Returns Varchar(MAX)
```

Das folgende Beispiel zeigt, wie Sie mit invoke_model ein Modell von Anthropic Claude 2 für Bedrock aufrufen.

```
SELECT aws_bedrock.invoke_model (
    'anthropic.claude-v2', -- model_id
    'application/json', -- content_type
    'application/json', -- accept_type
    '{"prompt": "\n\nHuman:
    You are a helpful assistant that answers questions directly
    and only using the information provided in the context below.
    \nDescribe the answerin detail.\n\nContext: %s \n\nQuestion:
    %s \n\nAssistant:","max_tokens_to_sample":4096,"temperature"
    :0.5,"top_k":250,"top_p":0.5,"stop_sequences":[]}' -- model_input
);
```

Example : aws_comprehend.detect_sentiment – Eine einfache Abfrage mit Amazon-Comprehend-Funktionen

```
aws_comprehend.detect_sentiment(
   input_text varchar,
   language_code varchar,
   max_rows_per_batch int)
Returns table (sentiment varchar, confidence real)
```

Im folgenden Beispiel wird gezeigt, wie Sie den Amazon-Comprehend-Service aufrufen.

```
select sentiment from aws_comprehend.detect_sentiment('This is great', 'en');
```

Example : aws_sagemaker.invoke_endpoint — Eine einfache Abfrage mit Amazon-Funktionen SageMaker

```
aws_sagemaker.invoke_endpoint(
  endpoint_name varchar,
  max_rows_per_batch int,
  VARIADIC model_input "any") -- Babelfish inherits PG's variadic parameter type
Rerurns Varchar(MAX)
```

Da model_input als VARIADIC markiert und vom Typ „any“ ist, können Benutzer eine Liste mit beliebiger Länge und beliebigem Datentyp an die Funktion übergeben, die als Eingabe für das Modell dient. Das folgende Beispiel zeigt, wie der SageMaker Amazon-Service aufgerufen wird.

```
SELECT CAST (aws_sagemaker.invoke_endpoint(
    'sagemaker_model_endpoint_name',
    NULL,
    arg1, arg2 -- model inputs are separate arguments )
AS INT) -- cast the output to INT
```

Weitere Informationen zur Verwendung von Aurora Machine Learning mit Aurora PostgreSQL finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora PostgreSQL.

Einschränkungen

- Babelfish erlaubt zwar nicht das Erstellen von Arrays, kann aber dennoch Daten verarbeiten, die Arrays repräsentieren. Wenn Sie Funktionen wie `aws_bedrock.invoke_model_get_embeddings` verwenden, die Arrays zurückgeben, werden die Ergebnisse als Zeichenfolge geliefert, die die Array-Elemente enthält.

Babelfish unterstützt verknüpfte Server

Babelfish für Aurora PostgreSQL unterstützt verknüpfte Server mithilfe der PostgreSQL-Erweiterung `tds_fdw` in Version 3.1.0. Sie müssen die Erweiterung `tds_fdw` installieren, um mit verknüpften Servern arbeiten zu können. Weitere Informationen über die Erweiterung

`tds_fdw` finden Sie unter Arbeiten mit den unterstützten Fremddaten-Wrapper für Amazon Aurora PostgreSQL.

Installieren der Erweiterung **`tds_fdw`**

Sie können die Erweiterung `tds_fdw` mit den folgenden Methoden installieren.

Verwendung von CREATE EXTENSION vom PostgreSQL-Endpunkt aus

1. Stellen Sie in der Babelfish-Datenbank im PostgreSQL-Port eine Verbindung mit Ihrer PostgreSQL-DB-Instance her. Verwenden Sie ein Konto, das über die Rolle rds_superuser verfügt.

```
psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
username=test --dbname=babelfish_db --password
```

2. Installieren Sie die `tds_fdw`-Erweiterung. Dies ist ein einmaliger Installationsvorgang. Sie müssen keine erneute Installation vornehmen, wenn der DB-Cluster neu gestartet wird.

```
babelfish_db=> CREATE EXTENSION tds_fdw;
CREATE EXTENSION
```

Aufruf der gespeicherten Prozedur **`sp_execute_postgresql`** vom TDS-Endpunkt aus

Babelfish unterstützt die Installation der Erweiterung `tds_fdw` durch Aufrufen der Prozedur `sp_execute_postgresql` ab Version 3.3.0. Sie können PostgreSQL-Anweisungen vom T-SQL-Endpunkt aus ausführen, ohne den T-SQL-Port zu verlassen. Weitere Informationen finden Sie unter Arbeiten mit Prozeduren von Babelfish für Aurora PostgreSQL.

1. Stellen Sie in der Babelfish-Datenbank im T-SQL-Port eine Verbindung zu Ihrer PostgreSQL-DB-Instance her.

```
sqlcmd -S your-DB-instance.aws-region.rds.amazonaws.com -U test -P password
```

2. Installieren Sie die `tds_fdw`-Erweiterung.

```
1>EXEC sp_execute_postgresql N'CREATE EXTENSION tds_fdw';
2>go
```

Unterstützte Funktionen

Babelfish unterstützt das Hinzufügen von Remote-Endpunkten von RDS für SQL Server oder Babelfish für Aurora PostgreSQL als verknüpften Server. Sie können auch andere Remote-Instances von SQL Server als verknüpfte Server hinzufügen. Verwenden Sie dann `OPENQUERY()`, um Daten von diesen verknüpften Servern abzurufen. Ab Babelfish Version 3.2.0 werden auch vierteilige Namen unterstützt.

Die folgenden gespeicherten Prozeduren und Katalogansichten werden unterstützt, um die verknüpften Server zu verwenden.

Gespeicherte Prozeduren

- sp_addlinkedserver – Babelfish unterstützt den Parameter `@provstr` nicht.
- sp_addlinkedsrvlogin
  - Sie müssen einen expliziten Remote-Benutzernamen und ein Passwort angeben, um eine Verbindung zu der Remote-Datenquelle herzustellen. Sie können keine Verbindung mit den eigenen Anmeldeinformationen des Benutzers herstellen. Babelfish unterstützt nur `@useself = false`.
  - Babelfish unterstützt den Parameter `@locallogin` nicht, da die Konfiguration des Remote-Serverzugriffs für die lokale Anmeldung nicht unterstützt wird.
- sp_linkedservers
- sp_helplinkedsrvlogin
- sp_dropserver
- sp_droplinkedsrvlogin – Babelfish unterstützt den Parameter `@locallogin` nicht, da die Konfiguration des Remote-Serverzugriffs für die lokale Anmeldung nicht unterstützt wird.
- sp_serveroption – Babelfish unterstützt die folgenden Serveroptionen:
  - Abfrage-Timeout (ab Babelfish-Version 3.2.0)
  - Verbindungs-Timeout (ab Babelfish-Version 3.3.0)
- sp_testlinkedserver (ab Babelfish-Version 3.3.0)
- sp_enum_oledb_providers (ab Babelfish-Version 3.3.0)

Katalogansichten

- sys.servers
- sys.linked_logins

Verwenden der Verschlüsselung während der Übertragung für die Verbindung

Die Verbindung vom Quellserver von Babelfish für Aurora PostgreSQL mit dem Remote-Zielserver verwendet je nach Datenbankkonfiguration des Remote-Servers die Verschlüsselung während der Übertragung (TLS/SSL). Wenn der Remote-Server nicht für die Verschlüsselung konfiguriert ist, bleibt der Babelfish-Server, der die Anforderung an die Remote-Datenbank stellt, unverschlüsselt.

So erzwingen Sie die Verbindungsverschlüsselung

- Wenn der verknüpfte Zielserver eine Instance von RDS für SQL Server ist, legen Sie `rds.force_ssl = on` für die Ziel-Instance von SQL Server fest. Weitere Informationen zur SSL/TLS-Konfiguration für RDS für SQL Server finden Sie unter Verwenden von SSL mit einer Microsoft-SQL-Server-DB-Instance.
- Wenn es sich bei dem verknüpften Zielserver um einen Cluster von Babelfish für Aurora PostgreSQL handelt, legen Sie `babelfishpg_tds.tds_ssl_encrypt = on` und `ssl = on` für den Zielserver fest. Weitere Informationen zu SSL/TLS finden Sie unter SSL-Einstellungen und Clientverbindungen für Babelfish.

Hinzufügen von Babelfish als verknüpften Server über SQL Server

Babelfish für Aurora PostgreSQL kann über einen SQL Server als verknüpfter Server hinzugefügt werden. In einer SQL-Server-Datenbank können Sie Babelfish als verknüpften Server hinzufügen, indem Sie den Anbieter von Microsoft OLE DB für ODBC verwenden: MSDASQL.

Es gibt zwei Möglichkeiten, Babelfish mithilfe des Anbieters MSDASQL über SQL Server als verknüpften Server zu konfigurieren:

- Angeben der ODBC-Verbindungszeichenfolge als Anbieterzeichenfolge
- Angeben des System-DSN der ODBC-Datenquelle, während der verknüpfte Server hinzugefügt wird

Einschränkungen

- OPENQUERY() funktioniert nur für SELECT und nicht für DML.
- Vierteilige Objektnamen funktionieren nur zum Lesen und nicht zum Ändern der entfernten Tabelle. Ein UPDATE kann in der FROM-Klausel auf eine entfernte Tabelle verweisen, ohne sie zu ändern.
- Die Ausführung von gespeicherten Prozeduren auf mit Babelfish verknüpften Servern wird nicht unterstützt.

- Das Upgrade der Hauptversion von Babelfish funktioniert möglicherweise nicht, wenn Objekte von `OPENQUERY()` abhängig sind oder Objekte durch vierteilige Namen referenziert werden. Sie müssen sicherstellen, dass alle Objekte, die auf `OPENQUERY()` oder vierteilige Namen verweisen, vor einem Upgrade der Hauptversion gelöscht werden.
- Die folgenden Datentypen funktionieren auf dem Babelfish-Remote-Server nicht wie erwartet: `nvarchar(max)`, `varchar(max)`, `varbinary(max)`, `binary(max)` und `time`. Wir empfehlen, die CAST-Funktion zu verwenden, um diese in unterstützte Datentypen zu konvertieren.

Beispiel

Im folgenden Beispiel stellt eine Babelfish-für-Aurora-PostgreSQL-Instance eine Verbindung zu einer Instance von RDS für SQL Server in der Cloud her.

```
EXEC master.dbo.sp_addlinkedserver @server=N'rds_sqlserver', @srvproduct=N'',
 @provider=N'SQLNCLI', @datasrc=N'myserver.CB2XKFSFFMY7.US-WEST-2.RDS.AMAZONAWS.COM';
EXEC master.dbo.sp_addlinkedsrvlogin
 @rmtsrvname=N'rds_sqlserver',@useself=N'False',@locallogin=NULL,@rmtuser=N'username',@rmtpassw
```

Wenn der verbundene Server eingerichtet ist, können Sie T-SQL OPENQUERY() oder eine vierteilige Standardbenennung verwenden, um auf eine Tabelle, Ansicht oder andere unterstützte Objekte auf dem Remote-Server zu verweisen:

```
SELECT * FROM OPENQUERY(rds_sqlserver, 'SELECT * FROM TestDB.dbo.t1');
SELECT * FROM rds_sqlserver.TestDB.dbo.t1;
```

So löschen Sie den verbundenen Server und alle verbundenen Logins:

```
EXEC master.dbo.sp_dropserver @server=N'rds_sqlserver', @droplogins=N'droplogins';
```

Fehlerbehebung

Sie können für Quell- und Remoteserver dieselbe Sicherheitsgruppe verwenden, um eine Kommunikation beider Server zu ermöglichen. Die Sicherheitsgruppe sollte nur eingehenden

Datenverkehr am TDS-Port (standardmäßig 1433) zulassen und die Quell-IP in der Sicherheitsgruppe kann als Sicherheitsgruppen-ID selbst festgelegt werden. Weitere Informationen zum Einrichten der Regeln für die Verbindung zu einer Instance von einer anderen Instance mit derselben Sicherheitsgruppe finden Sie unter Regeln für die Verbindung mit Instances von einer Instance mit der gleichen Sicherheitsgruppe aus.

Wenn der Zugriff nicht richtig konfiguriert ist, wird eine Fehlermeldung ähnlich wie das folgende Beispiel angezeigt, wenn Sie versuchen, den Remote-Server abzufragen.

```
TDS client library error: DB #: 20009, DB Msg: Unable to connect: server is unavailable
 or does not exist (mssql2019.aws-region.rds.amazonaws.com), OS #: 110, OS Msg:
 Connection timed out, Level: 9
```

Verwenden der Volltextsuche in Babelfish

Ab Version 4.0.0 bietet Babelfish eingeschränkte Unterstützung für die Volltextsuche (Full Text Search, FTS). FTS ist eine leistungsstarke Funktion in relationalen Datenbanken, die eine effiziente Suche und Indizierung von textlastigen Daten ermöglicht. Sie können damit komplexe Textsuchen durchführen und relevante Ergebnisse schnell abrufen. FTS ist besonders wertvoll für Anwendungen, die große Mengen an Textdaten verarbeiten, wie Content-Management-Systeme, E-Commerce-Plattformen und Dokumentenarchive.

Verstehen der in Babelfish unterstützten Funktionen der Volltextsuche

Babelfish unterstützt die folgenden Funktionen der Volltextsuche:

- Die CONTAINS-Klausel:
  - Die grundlegende Unterstützung für die CONTAINS-Klausel

```
CONTAINS (
     {
        column_name
     }
     , '<contains_search_condition>'
)
```

> **Note**
>
> Derzeit wird nur die englische Sprache unterstützt.

  - Die umfassende Bearbeitung und Übersetzung von `simple_term`-Suchzeichenfolgen
- Die `FULLTEXT INDEX`-Klausel:
  - Unterstützt nur die Anweisung `CREATE FULLTEXT INDEX ON table_name(column_name [...n]) KEY INDEX index_name`
  - Unterstützt die vollständige Anweisung `DROP FULLTEXT INDEX`

> **Note**
>
> Wenn Sie den Volltextindex neu indizieren möchten, müssen Sie ihn löschen und einen neuen für dieselbe Spalte erstellen.

- Sonderzeichen in der Suchbedingung:
  - Babelfish stellt sicher, dass einzelne Vorkommen von Sonderzeichen in Suchzeichenfolgen effektiv verarbeitet werden.

> **Note**
>
> Babelfish identifiziert zwar jetzt Sonderzeichen in der Suchzeichenfolge, es ist jedoch wichtig zu wissen, dass die erzielten Ergebnisse von denen mit T-SQL abweichen können.

- Den Tabellenalias in column_name:
  - Dank der Unterstützung von Tabellenalias können Benutzer präzisere und lesbarere SQL-Abfragen für die Volltextsuche erstellen.

Einschränkungen bei der Volltextsuche in Babelfish

- Derzeit werden die folgenden Optionen in Babelfish für die `CONTAINS`-Klausel nicht unterstützt.
  - Sonderzeichen und andere Sprachen als Englisch werden nicht unterstützt. Sie erhalten die allgemeine Fehlermeldung für nicht unterstützte Zeichen und Sprachen.

```
Full-text search conditions with special characters or languages other than English
 are not currently supported in Babelfish
```

  - Mehrere Spalten wie `column_list`
  - Das PROPERTY-Attribut
  - `prefix_term`, `generation_term`, `generic_proximity_term`, `custom_proximity_term`, und `weighted_term`
  - Boolesche Operatoren werden nicht unterstützt und Sie erhalten die folgende Fehlermeldung, wenn Sie sie verwenden:

    ```
    boolean operators not supported
    ```

  - Bezeichnernamen mit Punkten werden nicht unterstützt.
- Derzeit werden die folgenden Optionen in Babelfish für die `CREATE FULLTEXT INDEX`-Klausel nicht unterstützt.
  - [TYPE COLUMN type_column_name]
  - [LANGUAGE language_term]
  - [STATISTICAL_SEMANTICS]
  - Optionen für Katalogdateigruppen
  - mit Optionen
- Das Erstellen eines Volltextkatalogs wird nicht unterstützt. Für das Erstellen eines Volltextindexes ist kein Volltextkatalog erforderlich.
- `CREATE FULLTEXT INDEX` unterstützt keine Bezeichnernamen mit Punkten.
- Babelfish unterstützt derzeit keine aufeinanderfolgenden Sonderzeichen in Suchzeichenfolgen. Bei der Verwendung erhalten Sie die folgende Fehlermeldung:

```
Consecutive special characters in the full-text search condition are not currently
 supported in Babelfish
```

Babelfish unterstützt raumbezogene Datentypen

Ab den Versionen 3.5.0 und 4.1.0 bietet Babelfish Unterstützung für die folgenden zwei räumlichen Datentypen:

- Geometrie-Datentyp – Dieser Datentyp ist für die Speicherung planarer oder euklidischer (flachbezogener) Daten vorgesehen.
- Geografie-Datentyp – Dieser Datentyp ist für die Speicherung ellipsoidaler oder kugelbezogener Daten wie GPS-Längen- und Breitengradkoordinaten vorgesehen.

Diese Datentypen ermöglichen die Speicherung und Bearbeitung von Geodaten, allerdings mit Einschränkungen.

Verstehen der raumbezogenen Datentypen in Babelfish

- Raumbezogene Datentypen werden in verschiedenen Datenbankobjekten wie Ansichten, Prozeduren und Tabellen unterstützt.

- Unterstützt den 2D-Punktdatentyp zum Speichern von Standortdaten als Punkte, die durch Breitengrad, Längengrad und einen gültigen Spatial Reference System Identifier (SRID) definiert sind
- Anwendungen, die über Treiber wie JDBC, ODBC, DOTNET und PYTHON eine Verbindung zu Babelfish herstellen, können diese Geodatenfunktion nutzen.

Unterstützte Funktionen für Geometriedatentypen in Babelfish

- STGeomFromText (***geometry_tagged_text***, SRID) – Erzeugt eine Geometrie-Instance unter Verwendung der bekannten Textdarstellung (Well-Known Text, WKT)
- STPointFromText (***point_tagged_text***, SRID) – Erzeugt eine Punkt-Instance mithilfe der WKT-Darstellung
- Point (X, Y, SRID) – Erzeugt eine Punkt-Instance unter Verwendung von Gleitkommawerten der X- und Y-Koordinaten
- <geometry_instance>.STAsText ( ) – Extrahiert die WKT-Darstellung aus einer Geometrie-Instance
- <geometry_instance>.STDistance (other_geometry) – Berechnet die Entfernung zwischen zwei Geometrie-Instances
- <geometry_instance>.STX – Extrahiert die X-Koordinate (Längengrad) für die Geometrie-Instance
- <geometry_instance>.STY – Extrahiert die Y-Koordinate (Breitengrad) für die Geometrie-Instance

Unterstützte Funktionen für Geografie-Datentypen in Babelfish

- STGeomFromText (***geography_tagged_text***, SRID) – Erzeugt eine Geografie-Instance unter Verwendung der WKT-Darstellung
- STPointFromText (***point_tagged_text***, SRID) – Erzeugt eine Punkt-Instance mithilfe der WKT-Darstellung
- Point (Lat, Long, SRID) – Erzeugt eine Punkt-Instance unter Verwendung von Gleitkommawerten für Breitengrad und Längengrad
- <geography_instance>.STAsText ( ) – Extrahiert die WKT-Darstellung aus einer Geografie-Instance
- <geography_instance>.STDistance (other_geography) – Berechnet die Entfernung zwischen zwei Geografie-Instances
- <geography_instance>.Lat – Extrahiert den Wert des Breitengrads für die Geografie-Instance
- <geography_instance>.Long – Extrahiert den Wert des Längengrads für die Geografie-Instance

Einschränkungen in Babelfish für raumbezogene Datentypen

- Derzeit unterstützt Babelfish keine erweiterten Funktionen wie Z-M-Flags für Punkt-Instances von raumbezogenen Datentypen.
- Andere Geometrietypen als Punkt-Instances werden derzeit nicht unterstützt:
  - LineString
  - CircularString
  - CompoundCurve
  - Polygon
  - CurvePolygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection
- Derzeit wird die räumliche Indizierung für raumbezogene Datentypen nicht unterstützt.
- Derzeit werden nur die aufgelisteten Funktionen für diese Datentypen unterstützt. Weitere Informationen erhalten Sie unter Unterstützte Funktionen für Geometriedatentypen in Babelfish und Unterstützte Funktionen für Geografie-Datentypen in Babelfish.

- Die Ausgabe der STDistance-Funktion für geografische Daten kann im Vergleich zu T-SQL geringfügige Genauigkeitsabweichungen aufweisen. Dies ist auf die zugrunde liegende PostGIS-Implementierung zurückzuführen. Weitere Informationen finden Sie unter ST_Distance.
- Für eine optimale Leistung sollten Sie integrierte raumbezogene Datentypen verwenden, ohne zusätzliche Abstraktionsebenen in Babelfish zu erstellen.

> **Tip**
>
> Sie können zwar benutzerdefinierte Datentypen erstellen, es wird jedoch nicht empfohlen, diese zusätzlich zu Geodaten zu erstellen. Dies könnte zu Komplexitäten führen und aufgrund der eingeschränkten Unterstützung möglicherweise unerwartetes Verhalten verursachen.

- In Babelfish werden Geodaten-Funktionsnamen als Schlüsselwörter verwendet und führen räumliche Operationen nur dann durch, wenn sie in der vorgesehenen Weise verwendet werden.

> **Tip**
>
> Vermeiden Sie bei der Erstellung benutzerdefinierter Funktionen und Prozeduren in Babelfish die Verwendung derselben Namen wie bei integrierten Geodatenfunktionen. Wenn Sie bereits Datenbankobjekte mit denselben Namen haben, verwenden Sie `sp_rename`, um sie umzubenennen.

Verstehen der Partitionierung in Babelfish

Ab Version 4.3.0 führt Babelfish die Tabellen- und Indexpartitionierung mit eingeschränkter Unterstützung ein. Die folgenden Abschnitte enthalten Details zur Erstellung von Partitionsfunktionen, zur Definition von Partitionsschemas und zur Implementierung partitionierter Tabellen und Indizes in Babelfish.

Themen

- Einführung in die Partitionierung in Babelfish
- Einschränkungen und Problemumgehungen

Einführung in die Partitionierung in Babelfish

- Partitionsfunktionen:
  - `CREATE PARTITION FUNCTION`: Definiert, wie eine Tabelle oder ein Index partitioniert wird, indem der Partitionsspaltentyp und der Wertebereich für jede Partition angegeben werden
  - `DROP PARTITION FUNCTION`: Entfernt eine vorhandene Partitionsfunktion
- Partitionsschemas:
  - `CREATE PARTITION SCHEME`: Definiert die Zuordnung zwischen Partitionen und Dateigruppen

    > **Note**
    >
    > In Babelfish werden Dateigruppen als Dummy-Objekte behandelt und stellen keine physischen Speicherorte dar.

  - `DROP PARTITION SCHEME`: Entfernt ein vorhandenes Partitionsschema
- Systemfunktionen:
  - `$PARTITION`: Diese Systemfunktion gibt die Partitionsnummer zurück, zu der ein bestimmter Wert in einer Partitionierungsspalte in einer angegebenen partitionierten Tabelle gehören würde.
- Partitionierte Tabellen und Indizes:
  - `CREATE TABLE ... ON partition_scheme_name (partition_column_name)`: Erstellt eine partitionierte Tabelle basierend auf einem angegebenen Partitionsschema und einer Partitionsspalte
  - `CREATE INDEX ... ON partition_scheme_name (partition_column_name)`: Erstellt einen partitionierten Index basierend auf einem angegebenen Partitionsschema und einer Partitionsspalte
- Systemansichten für die Partitionierung von Metadaten:

  Die folgenden Systemansichten wurden hinzugefügt, um Metadaten zur Partitionierung bereitzustellen:

  - `sys.destination_data_spaces`
  - `sys.partitions`
  - `sys.partition_functions`
  - `sys.partition_parameters`
  - `sys.partition_range_values`

- `sys.partition_schemes`

Einschränkungen und Problemumgehungen

Die folgenden SQL-Server-Partitionierungsfunktionen werden von Babelfish noch nicht unterstützt:

- `ALTER PARTITION FUNCTION` und `ALTER PARTITION SCHEME`.

> **Note**
>
> Babelfish unterstützt keine Operationen zum Aufteilen und Zusammenführen. Definieren Sie bei der Erstellung alle Partitionen in den Partitionsfunktionen, da Sie später keine Partitionen hinzufügen oder entfernen können.

- Berechnete Spalten als Partitionierungsspalten
- Die Dienstprogramme `INSERT BULK` und `BCP` für partitionierte Tabellen
- Die Grenzoption `LEFT` für Partitionsfunktionen
- Der Datentyp `SQL_VARIANT` für Partitionsfunktionen
- `TRUNCATE TABLE ... WITH PARTITION`.
- `ALTER TABLE ... SWITCH PARTITION`.
- Nicht ausgerichtete partitionierte Indizes, wie etwa ein abweichendes Partitionierungsschema oder eine andere Partitionierungsspalte als in der partitionierten Tabelle
- Die DMS-Migration aus der Babelfish-Quelle wird nur für Volllastaufgaben in partitionierten Tabellen unterstützt.
- Babelfish unterstützt diese Syntaxoptionen nicht, bietet aber Behelfslösungen:
  - Verwenden eines Partitionsschemas mit Einschränkungen oder Indizes in der CREATE TABLE-Anweisung
  - ALTER TABLE ... ADD CONSTRAINT ... ON partition_scheme_name (partition_column_name)

> **Note**
>
> Die Notfalloption `babelfishpg_tsql.escape_hatch_storage_on_partition` kann auf „ignore“ gesetzt werden. Dadurch kann der Parser die Partitionsschemaoption ignorieren, die mit Einschränkungen oder Indizes verwendet wird, und das Backend erstellt individuelle Einschränkungen oder Indizes für jede Partition.

Fehlerbehebung bei Babelfish

Im Folgenden finden Sie Tipps zur Fehlerbehebung und Workarounds für einige Probleme mit Babelfish-DB-Clustern.

Themen

- Verbindungsfehler

Verbindungsfehler

Häufige Ursachen für Verbindungsfehler mit einem neuen Aurora-DB-Cluster umfassen Folgende:

- Sicherheitsgruppe lässt keinen Zugriff zu — Wenn Sie Probleme haben, sich mit einem Babelfish zu verbinden, stellen Sie sicher, dass Sie Ihre IP-Adresse der Amazon EC2-Sicherheitsgruppe hinzugefügt haben. Sie können https://checkip.amazonaws.com/ nutzen, um Ihre IP-Adresse zu ermitteln und sie dann Ihrer Inbound-Regel für den TDS-Port und den PostgreSQL-Port hinzuzufügen. Weitere Informationen finden Sie unter Hinzufügen von Regeln zur Sicherheitsgruppe im Amazon EC2-Benutzerhandbuch.
- Nicht übereinstimmende SSL-Konfigurationen – Wenn der `rds.force_ssl`-Parameter in Aurora PostgreSQL aktiviert (auf 1 eingestellt) ist, dann müssen sich Clients über SSL mit Babelfish verbinden. Wenn Ihr Client nicht richtig eingerichtet ist, wird Ihnen eine Fehlermeldung angezeigt, z. B. die folgende:

  ```
  Cannot connect to your-Babelfish-DB-cluster, 1433
  ---------------------
  ADDITIONAL INFORMATION:
  no pg_hba_conf entry for host "256.256.256.256", user "your-user-name",
  "database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
  ...
  ```

  Dieser Fehler weist auf ein mögliches SSL-Konfigurationsproblem zwischen Ihrem lokalen Client und dem Babelfish-DB-Cluster hin und darauf, dass der Cluster von Clients die Verwendung von SSL verlangt (der zugehörige `rds.force_ssl`-Parameter ist auf 1 festgelegt). Weitere Informationen zum Konfigurieren von SSL finden Sie unter Verwenden von SSL mit einer PostgreSQL-DB-Instance im Amazon RDS-Benutzerhandbuch.

  Wenn Sie SQL Server Management Studio (SSMS) verwenden, um eine Verbindung mit Babelfish herzustellen und dieser Fehler angezeigt wird, können Sie die Verbindungsoptionen Encrypt

connection (Verbindung verschlüsseln) und Trust server certificate (Trust-Serverzertifikat) im Bereich „Connection Properties“ (Verbindungseigenschaften) auswählen und es erneut versuchen. Diese Einstellungen übernehmen die SSL-Verbindungsanforderung für SSMS.

Weitere Informationen zur Fehlerbehebung von Aurora-DB-Verbindungsproblemen finden Sie unter Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden.

Deaktivieren von Babelfish

Wenn Sie Babelfish nicht mehr benötigen, können Sie die Babelfish-Funktionalität deaktivieren.

Achten Sie auf einige Überlegungen:

- In einigen Fällen können Sie Babelfish deaktivieren, bevor Sie eine Migration zu Aurora PostgreSQL abschließen. Wenn Sie dies tun und Ihre DDL von SQL Server-Datentypen abhängt oder Sie eine T-SQL-Funktionalität in Ihrem Code verwenden, schlägt Ihr Code fehl.
- Wenn Sie nach der Bereitstellung einer Babelfish-Instanz die Babelfish-Erweiterung deaktivieren, können Sie dieselbe Datenbank nicht erneut auf demselben Cluster bereitstellen.

Um Babelfish zu deaktivieren, ändern Sie Ihre Parametergruppe und stellen Sie `rds.babelfish_status` auf `OFF`. Sie können Ihre SQL Server-Datentypen weiterhin mit Babelfish off verwenden, indem Sie `rds.babelfish_status` auf `datatypesonly` stellen.

Wenn Sie Babelfish in der Parametergruppe deaktivieren, verlieren alle Cluster, die diese Parametergruppe verwenden, die Babelfish-Funktionalität.

Weitere Informationen zu Aurora-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora. Informationen zum Festlegen von Babelfish-spezifischen Parametern finden Sie unter Einstellungen der DB-Cluster-Parametergruppe für Babelfish.

Verwalten von Versionsupdates von Babelfish für Aurora PostgreSQL

Babelfish ist eine Option, die mit Aurora PostgreSQL Version 13.4 und höher verfügbar ist. Updates für Babelfish werden mit bestimmten neuen Versionen der Aurora-PostgreSQL-Datenbank-Engine verfügbar. Weitere Informationen finden Sie unter Versionshinweise für Aurora PostgreSQL.

> **Note**
>
> Babelfish-DB-Cluster, die auf einer beliebigen Version von Aurora PostgreSQL 13 ausgeführt werden, können nicht auf Aurora PostgreSQL 14.3, 14.4 und 14.5 aktualisiert werden. Außerdem unterstützt Babelfish kein direktes Upgrade von 13.x auf 15.x. Sie müssen zuerst Ihren 13.x-DB-Cluster auf 14.6 und höher und dann auf die 15.x-Version aktualisieren.

Eine Liste der unterstützten Funktionen für verschiedene Babelfish-Versionen finden Sie unter Unterstützte Funktionen in Babelfish nach Version.

Eine Liste der nicht unterstützten Funktionen finden Sie unter Nicht unterstützte Funktionen in Babelfish.

Sie können eine Liste der Aurora-PostgreSQL-Versionen abrufen, die Babelfish unterstützen, indem Sie Ihre AWS-Region mit dem AWS CLI-Befehl describe-db-engine-versions wie folgt abfragen.

Für Linux, macOS oder Unix:

```
$ aws rds describe-db-engine-versions --region us-east-1 \
    --engine aurora-postgresql \
    --query '*[]|[?SupportsBabelfish==`true`].[EngineVersion]' \
    --output text
```

Weitere Informationen finden Sie unter describe-db-engine-versions in der Referenz zum AWS CLI-Befehl.

In den folgenden Themen erfahren Sie, wie Sie die Version von Babelfish identifizieren, die auf Ihrem DB-Cluster von Aurora PostgreSQL ausgeführt wird, und wie Sie auf eine neue Version aktualisieren.

Inhalt

- Identifizieren Ihrer Babelfish-Version
- Durchführen eines Upgrades Ihres Babelfish-Clusters auf eine neue Version
  - Durchführen eines Upgrades von Babelfish auf eine neue Nebenversion

- Durchführen von Upgrades von Babelfish auf eine neue Hauptversion
  - Vor dem Durchführen von Upgrades von Babelfish auf eine neue Hauptversion
  - Durchführen eines Hauptversions-Upgrades
  - Nach dem Durchführen von Upgrades auf eine neue Hauptversion
  - Beispiel: Durchführen eines Upgrades des Babelfish-DB-Clusters auf eine Hauptversion
- Verwenden des Babelfish-Produktversionsparameters
  - Konfigurieren des Babelfish-Produktversionsparameters
  - Betroffene Abfragen und Parameter
  - Schnittstelle mit dem Parameter babelfishpg_tsql.version

Identifizieren Ihrer Babelfish-Version

Sie können Babelfish abfragen, um Details zur Babelfish-Version, der Aurora-PostgreSQL-Version und der kompatiblen Microsoft SQL Server-Version zu finden. Sie können den TDS-Port oder den PostgreSQL-Port verwenden.

- To use the TDS port to query for version information
- To use the PostgreSQL port to query for version information

So fragen Sie Versionsinformationen mit dem TDS-Port ab

1. Verwenden Sie `sqlcmd` oder `ssms`, um eine Verbindung mit dem Endpunkt Ihres Babelfish-DB-Clusters herzustellen.

   ```
   sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
       login-id -P password -d db_name
   ```

2. Um die Babelfish-Version zu identifizieren, führen Sie die folgende Abfrage aus:

   ```
   1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
   2> GO
   ```

   Die Abfrage gibt Ergebnisse wie die Folgenden zurück:

   ```
   serverproperty
   ------------------------------
   ```

```
3.4.0

(1 rows affected)
```

3. Um die Version des Aurora-PostgreSQL-DB-Clusters zu identifizieren, führen Sie die folgende Abfrage aus:

   ```
   1> SELECT aurora_version() AS aurora_version
   2> GO
   ```

   Die Abfrage gibt Ergebnisse wie die Folgenden zurück:

   ```
   aurora_version


   -----------------------------------------------
   15.5.0

   (1 rows affected)
   ```

4. Führen Sie die folgende Abfrage aus, um die kompatible Microsoft SQL Server-Version zu identifizieren:

   ```
   1> SELECT @@VERSION AS version
   2> GO
   ```

   Die Abfrage gibt Ergebnisse wie die Folgenden zurück:

   ```
   Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
   Dec 7 2023 09:43:06
   Copyright (c) Amazon Web Services
   PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)

   (1 rows affected)
   ```

Als Beispiel für einen kleinen Unterschied zwischen Babelfish und Microsoft SQL Server können Sie die folgende Abfrage ausführen. Bei Babelfish gibt die Abfrage 1 und auf Microsoft SQL Server `NULL` zurück.

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

Sie können auch den PostgreSQL-Port verwenden, um Versionsinformationen abzurufen, wie im Folgenden beschrieben.

So fragen Sie Versionsinformationen mit dem PostgreSQL-Port ab

1. Verwenden Sie `psql` oder `pgAdmin`, um eine Verbindung mit dem Endpunkt Ihres Babelfish-DB-Clusters herzustellen.

   ```
   psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
        port=5432 dbname=babelfish_db user=sa
   ```

2. Schalten Sie das erweiterte Feature (`\x`) von `psql` ein, damit die Ausgabe besser lesbar ist.

   ```
   babelfish_db=> \x
   babelfish_db=> SELECT
   babelfish_db=> aurora_version() AS aurora_version,
   babelfish_db=> version() AS postgresql_version,
   babelfish_db=> sys.version() AS Babelfish_compatibility,
   babelfish_db=> sys.SERVERPROPERTY('BabelfishVersion') AS Babelfish_Version;
   ```

   Die Abfrage gibt dann eine Ausgabe ähnlich der folgenden zurück:

   ```
   -[ RECORD 1 ]-----------
   +-------------------------------------------------------------------------------------------------
   aurora_version          | 15.5.0
   postgresql_version      | PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by
    x86_64-pc-linux-gnu-gcc (GCC) 9.5.0, 64-bit
   babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server
    Compatibility - 12.0.2000.8                    +
                           | Dec 7 2023 09:43:06
                                              +
                           | Copyright (c) Amazon Web Services
                                               +
                           | PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)
   babelfish_version       | 3.4.0
   ```

Durchführen eines Upgrades Ihres Babelfish-Clusters auf eine neue Version

Neue Versionen von Babelfish werden mit einigen neuen Versionen der Datenbank-Engine von Aurora PostgreSQL-Datenbank nach Version 13.4 verfügbar. Jede neue Version von Babelfish hat eine eigene Versionsnummer. Wie bei Aurora PostgreSQL verwendet Babelfish das Benennungsschema *major*.*minor*.*patch* für Versionen. Zum Beispiel wurde die erste Babelfish-Version, Babelfish Version 1.0.0, im Rahmen von Aurora PostgreSQL 13.4.0 verfügbar.

Babelfish benötigt keinen separaten Installationsvorgang. Wie unter Erstellen eines DB-Clusters von Babelfish for Aurora PostgreSQL beschrieben, wählen Sie die Option Turn on Babelfish (Babelfish aktivieren) aus, wenn Sie einen DB-Cluster von Aurora PostgreSQL erstellen.

Dementsprechend können Sie Babelfish nicht unabhängig vom unterstützenden Aurora-DB-Cluster aktualisieren. Wenn Sie einen vorhandenen DB-Cluster von Babelfish für Aurora PostgreSQL auf eine neue Version von Babelfish aktualisieren möchten, führen Sie ein Upgrade des DB-Clusters von Aurora PostgreSQL auf eine neue Version durch, die die Version von Babelfish unterstützt, die Sie verwenden möchten. Das Verfahren, das Sie für das Upgrade befolgen, hängt wie folgt von der Version von Aurora PostgreSQL ab, die Ihre Babelfish-Bereitstellung unterstützt.

Hauptversions-Upgrades

Sie müssen die folgenden Aurora-PostgreSQL-Versionen vor dem Upgrade auf Aurora PostgreSQL 15.2 und höher zunächst auf Aurora PostgreSQL 14.6 und höher aktualisieren.

- Aurora PostgreSQL 13.8 und alle höheren Versionen
- Aurora PostgreSQL 13.7.1 und alle höheren Nebenversionen
- Aurora PostgreSQL 13.6.4 und alle höheren Nebenversionen

Sie können Aurora PostgreSQL 14.6 und höhere Versionen auf Aurora PostgreSQL 15.2 und höhere Versionen aktualisieren.

Das Upgrade eines DB-Clusters von Aurora PostgreSQ auf eine neue Hauptversion erfordert mehrere vorbereitende Aufgaben. Weitere Informationen finden Sie unter Durchführen eines Hauptversions-Upgrades. Damit Sie Ihren DB-Cluster von Babelfish für Aurora PostgreSQL erfolgreich aktualisieren können, müssen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe für die neue Aurora-PostgreSQL-Version erstellen. Diese neue Parametergruppe muss dieselben Babelfish-Parametereinstellungen enthalten wie die des Clusters, den Sie aktualisieren. Weitere Informationen und eine Tabelle mit Quellen und Zielen für das Upgrade von Hauptversionen finden Sie unter Durchführen von Upgrades von Babelfish auf eine neue Hauptversion.

Nebenversions-Upgrades und -Patches

Für das Upgrade von Nebenversionen und Patches muss keine neue DB-Cluster-Parametergruppe erstellt werden. Nebenversionen und Patches können den Upgrade-Prozess für Nebenversionen verwenden, unabhängig davon, ob dieser automatisch oder manuell erfolgt. Weitere Informationen und eine Tabelle mit Versionsquellen und -zielen finden Sie unter Durchführen eines Upgrades von Babelfish auf eine neue Nebenversion.

> **Note**
>
> Bevor Sie ein Haupt- oder Nebenversions-Upgrade durchführen, führen Sie alle ausstehenden Wartungsaufgaben für Ihren Cluster von Babelfish für Aurora PostgreSQL aus.

Themen

- Durchführen eines Upgrades von Babelfish auf eine neue Nebenversion
- Durchführen von Upgrades von Babelfish auf eine neue Hauptversion

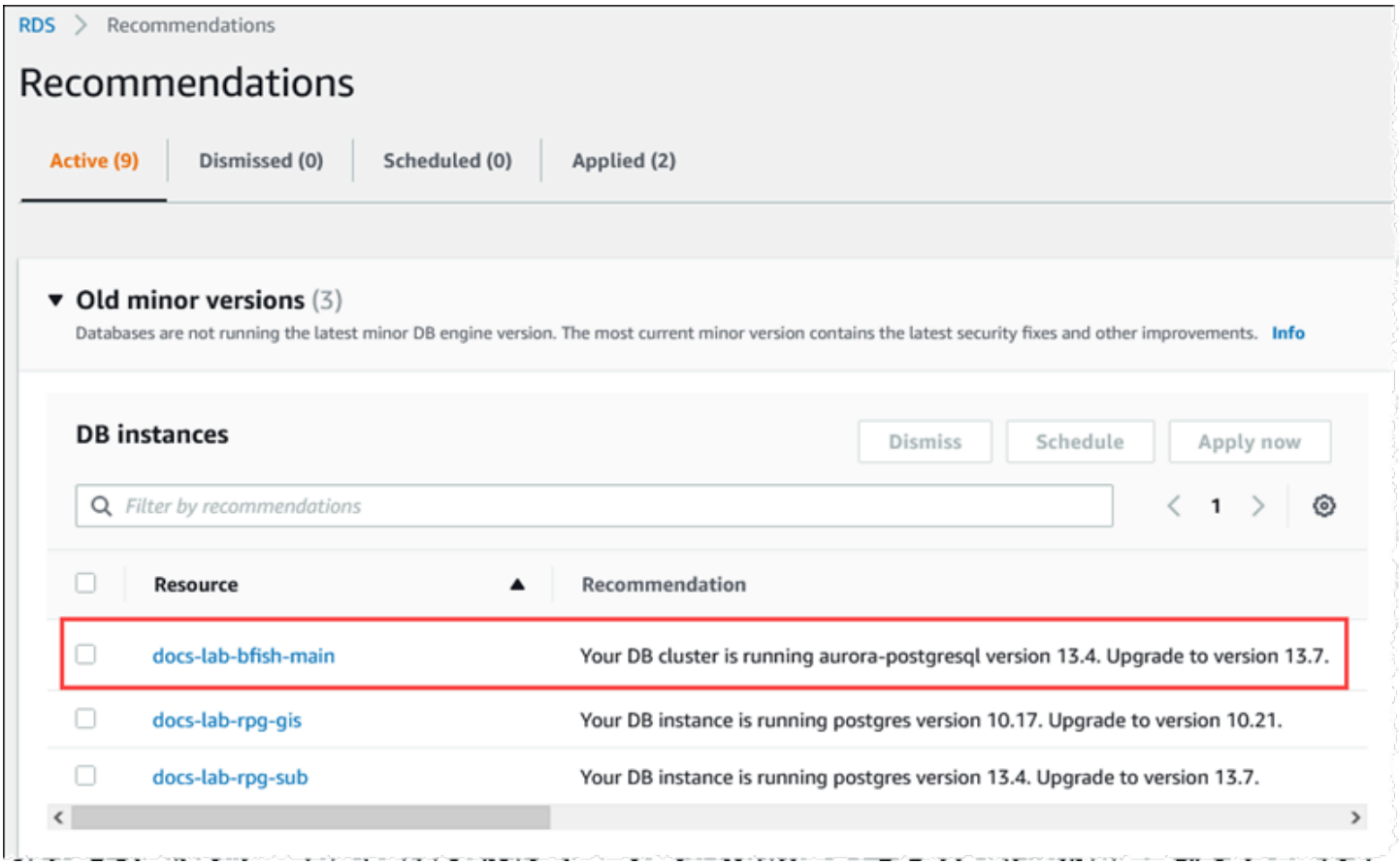
Durchführen eines Upgrades von Babelfish auf eine neue Nebenversion

Ein Nebenversionsupdate zielt darauf ab, die Abwärtskompatibilität aufrechtzuerhalten. In einigen Fällen sind kritische Sicherheitskorrekturen oder wichtige Bugfixes jedoch möglicherweise nicht vollständig abwärtskompatibel. Eine Patch-Version enthält wichtige Korrekturen, die einer Nebenversion nach ihrer Veröffentlichung hinzugefügt werden. Die Versionsbezeichnung für die erste Version von Aurora PostgreSQL 13.4 lautete beispielsweise Aurora PostgreSQL 13.4.0. Bisher wurden mehrere Patches für diese Nebenversion veröffentlicht, darunter Aurora PostgreSQL 13.4.1, 13.4.2 und 13.4.4. Die für jede Aurora-PostgreSQL-Version verfügbaren Patches finden Sie in der Liste Patch releases (Patch-Versionen) oben in den Versionshinweisen von Aurora PostgreSQL für die entsprechende Version. Ein Beispiel finden Sie unter PostgreSQL 14.3 in den Versionshinweisen für Aurora PostgreSQL.

Wenn Ihr DB-Cluster von Aurora PostgreSQL mit der Option Auto minor version upgrade (Automatisches Unterversion-Upgrade) konfiguriert ist, wird Ihr DB-Cluster von Babelfish für Aurora PostgreSQL während des Wartungsfensters des Clusters automatisch aktualisiert. Weitere Informationen zum automatischen Nebenversions-Upgrade (AmVU) und wie Sie es verwenden, finden Sie unter Automatische Nebenversions-Upgrades für Aurora-DB-Cluster. Wenn Ihr Cluster AmVU nicht verwendet, können Sie Ihren DB-Cluster von Babelfish für Aurora PostgreSQL auf neue

Nebenversionen manuell aktualisieren, indem Sie entweder auf Wartungsaufgaben reagieren oder den Cluster so ändern, dass er die neue Version verwendet.

Wenn Sie eine Aurora-PostgreSQL-Version zur Installation auswählen und einen vorhandenen DB-Cluster von Aurora PostgreSQL in der AWS-Managementkonsole anzeigen, zeigt die Version nur die *major.minor*-Ziffern an. In der folgenden Abbildung aus der Konsole wird für einen vorhandenen DB-Cluster von Babelfish für Aurora PostgreSQL mit Aurora PostgreSQL 13.4 beispielsweise ein Upgrade des Clusters auf Version 13.7 empfohlen, eine neue Nebenversion von Aurora PostgreSQL.



Um vollständige Versionsdetails, einschließlich der *Patch*-Ebene, zu erhalten, können Sie den DB-Cluster von Aurora PostgreSQL mithilfe der Aurora-PostgreSQL-Funktion `aurora_version` abfragen. Weitere Informationen finden Sie unter aurora_version im Aurora-PostgreSQL-Funktionsreferenz. Ein Beispiel für die Verwendung der Funktion finden Sie in der Prozedur To use the PostgreSQL port to query for version information unter Identifizieren Ihrer Babelfish-Version.

Die folgende Tabelle zeigt die Versionen von Aurora PostgreSQL und Babelfish sowie die verfügbaren Zielversionen, die den Upgrade-Prozess für Nebenversionen unterstützen können.

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
Aurora PostgreSQL (Babelfish)	Aurora PostgreSQL (Babelfish)
17.4 (5.1)	17.5 (5.2)
16.8 (4.5)	16.9 (4.6)
16.6 (4.4.0)	16.9 (4.6), 16.8 (4.5.0)
16.4 (4.3.0)	16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0)
16.3 (4.2.0)	16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0)
16.2 (4.1.0)	16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0)
16.1 (4.0.0)	16.9 (4.6), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0)
15.12 (3.9.0)	15.13 (3.10)
15.10 (3.8.0)	15.13 (3.10), 15.12 (3.9.0)
15.8 (3.7.0)	15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0)
15.7 (3.6.0)	15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0)
15.6 (3.5.0)	15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0)
15.5 (3.4.0)	15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0)
15.4 (3.3.0)	15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0)

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
15.12 (3.9.0), 15.3 (3.2.0)	15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0)
15.2 (3.1.0)	15.13 (3.10), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0)
14.17 (2.12.0)	14.18 (2.13.0)
14.15 (2.11.0)	14.18 (2.13.0), 14.17 (2.12.0)
14.13 (2.10.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0)
14.12 (2.9.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0)
14.11 (2.8.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0)
14.10 (2.7.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0)
14.9 (2.6.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0)
14.8 (2.5.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0)
14.7 (2.4.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0), 14.8 (2.5.0)

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
14.6 (2.3.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0), 14.8 (2.5.0), 14.7 (2.4.0)
14.5 (2.2.0)	14.18 (2.13.0), 14.17 (2.12.0), 14.15 (2.11.0), 14.13 (2.10.0), 14.12 (2.9.0), 14.11 (2.8.0), 14.10 (2.7.0), 14.9 (2.6.0), 14.8 (2.5.0), 14.7 (2.4.0), 14.6 (2.3.0)
14.3 (2.1.0)	14.18 (2.13.0), 14.6 (2.3.0)
13.8 (1.4.0)	13.9 (1.5.0)
13.7 (1.3.0)	13.9 (1.5.0), 13.8 (1.4.0)

Durchführen von Upgrades von Babelfish auf eine neue Hauptversion

Für ein Hauptversions-Upgrade müssen Sie zuerst Ihren DB-Cluster von Babelfish für Aurora PostgreSQL auf eine Version aktualisieren, die das Hauptversions-Upgrade unterstützt. Wenden Sie dazu Patch-Updates oder Nebenversions-Upgrades auf Ihren DB-Cluster an. Weitere Informationen finden Sie unter Durchführen eines Upgrades von Babelfish auf eine neue Nebenversion.

Die folgende Tabelle zeigt die Versionen von Aurora PostgreSQL und Babelfish, die ein Hauptversions-Upgrade unterstützen können.

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
Aurora PostgreSQL (Babelfish)	Aurora PostgreSQL (Babelfish)
16.9 (4.6.0)	17.5 (5.2.0)
16.8 (4.5.0)	17.5 (5.2.0), 17.4 (5.1.0)
16.6 (4.4.0)	17.5 (5.2.0), 17.4 (5.1.0)
16.4 (4.3.0)	17.5 (5.2.0), 17.4 (5.1.0)

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
16.3 (4.2.0)	17.5 (5.2.0), 17.4 (5.1.0)
16.2 (4.1.0)	17.5 (5.2.0), 17.4 (5.1.0)
16.1 (4.0.0)	17.5 (5.2.0), 17.4 (5.1.0)
15.13 (3.10)	17.5 (5.2.0) 16.9 (4.6.0)
15.12 (3.9.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0)
15.10 (3.8.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0)
15.8 (3.7.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0)
15.7 (3.6.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0)
15.6 (3.5.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0)
15.5 (3.4.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
15.4 (3.3.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)
15.3 (3.2.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)
15.2 (3.1.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0)
14.18 (2.13.0)	17.5 (5.2.0) 16.9 (4.6.0) 15.13 (3.10.0)
14.17 (2.12.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0) 15.13 (3.10.0), 15.12 (3.9.0)
14.15 (2.11.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0)
14.13 (2.10.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0)

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
14.12 (2.9.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0)
14.11 (2.8.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0)
14.10 (2.7.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0)
14.9 (2.6.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0)

Aktuelle Quellversionen	Neuestes Upgrade-Ziele
14.8 (2.5.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0)
14.7 (2.4.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0), 15.2 (3.1.0)
14.6 (2.3.0)	17.5 (5.2.0), 17.4 (5.1.0) 16.9 (4.6.0), 16.8 (4.5.0), 16.6 (4.4.0), 16.4 (4.3.0), 16.3 (4.2.0), 16.2 (4.1.0), 16.1 (4.0.0) 15.13 (3.10.0), 15.12 (3.9.0), 15.10 (3.8.0), 15.8 (3.7.0), 15.7 (3.6.0), 15.6 (3.5.0), 15.5 (3.4.0), 15.4 (3.3.0), 15.3 (3.2.0), 15.2 (3.1.0)
13.9 (1.5.0)	14.6 (2.3.0)
13.8 (1.4.0)	14.6 (2.3.0)
13.7 (1.3.0)	14.6 (2.3.0)

Vor dem Durchführen von Upgrades von Babelfish auf eine neue Hauptversion

Ein Upgrade kann zu kurzen Ausfällen führen. Aus diesem Grund empfehlen wir, dass Sie Upgrades während Ihres Wartungsfensters oder in anderen Zeiträumen geringer Auslastung durchführen oder planen.

Vor dem Durchführen eines Hauptversions-Upgrades

1. Ermitteln Sie die Babelfish-Version Ihres vorhandenen DB-Clusters von Aurora PostgreSQL mithilfe der unter Identifizieren Ihrer Babelfish-Version beschriebenen Befehle. Die Versionsinformationen von Aurora PostgreSQL und Babelfish werden von PostgreSQL verarbeitet. Folgen Sie daher den im Verfahren To use the PostgreSQL port to query for version information beschriebenen Schritten, um die Details zu erhalten.
2. Überprüfen Sie, ob Ihre Version das Hauptversions-Upgrade unterstützt. Eine Liste der Versionen, die die Funktion zum Upgraden der Hauptversion unterstützen, finden Sie unter Durchführen eines Upgrades von Babelfish auf eine neue Nebenversion. Führen Sie außerdem die vor dem Upgrade erforderlichen Aufgaben durch.

   Wenn Ihre Babelfish-Version beispielsweise auf einem DB-Cluster von Aurora PostgreSQL 13.5 läuft und Sie ein Upgrade auf Aurora PostgreSQL 15.2 durchführen möchten, wenden Sie zuerst alle Nebenversionen und Patches an, um Ihren Cluster auf Aurora PostgreSQL 14.6 zu aktualisieren. Wenn Ihr Cluster Version 14.6. oder höher ausführt, fahren Sie mit dem Upgrade der Hauptversion fort.
3. Erstellen Sie einen manuellen Snapshot Ihres aktuellen Babelfish-DB-Clusters als Backup. Mit dem Backup können Sie den Cluster auf seine Aurora-PostgreSQL- und Babelfish-Version zurücksetzen und alle Daten auf den Zustand vor dem Upgrade wiederherstellen. Weitere Informationen finden Sie unter Erstellen eines DB-Cluster-Snapshots. Stellen Sie sicher, dass Sie Ihre vorhandene benutzerdefinierte DB-Cluster-Parametergruppe beibehalten, um sie erneut zu verwenden, wenn Sie diesen Cluster auf den Zustand vor dem Upgrade wiederherstellen möchten. Weitere Informationen erhalten Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot und Überlegungen zu Parametergruppen.
4. Bereiten Sie eine benutzerdefinierte DB-Cluster-Parametergruppe für die DB-Zielversion von Aurora PostgreSQL vor. Duplizieren Sie die Einstellungen für die Babelfish-Parameter aus Ihrem aktuellen DB-Cluster von Babelfish für Aurora PostgreSQL. Eine Liste aller Babelfish-Parameter finden Sie unter Einstellungen der DB-Cluster-Parametergruppe für Babelfish. Für ein Upgrade der Hauptversion sind für die folgenden Parameter dieselben Einstellungen wie für den Quell-DB-Cluster erforderlich. Damit das Upgrade erfolgreich ist, müssen alle Einstellungen identisch sein.

   - rds.babelfish_status
   - babelfishpg_tds.tds_default_numeric_precision
   - babelfishpg_tds.tds_default_numeric_scale

- babelfishpg_tsql.database_name
- babelfishpg_tsql.default_locale
- babelfishpg_tsql.migration_mode
- babelfishpg_tsql.server_colation_name

> **Warning**
>
> Wenn die Einstellungen für die Babelfish-Parameter in der benutzerdefinierten DB-Cluster-Parametergruppe für die neue Aurora-PostgreSQL-Version nicht mit den Parameterwerten des Clusters übereinstimmen, den Sie aktualisieren, schlägt die Operation `ModifyDBCluster` fehl. Die Fehlermeldung `InvalidParameterCombination` erscheint in der AWS-Managementkonsole oder in der Ausgabe des AWS CLI-Befehls `modify-db-cluster`.

5. Verwenden Sie die AWS-Managementkonsole oder die AWS CLI, um die benutzerdefinierte Parametergruppe für den DB-Cluster zu erstellen. Wählen Sie die entsprechende Aurora-PostgreSQL-Familie für die Version von Aurora PostgreSQL aus, die Sie für das Upgrade benötigen.

> **Tip**
>
> Parametergruppen werden auf der Ebene der AWS-Region verwaltet. Wenn Sie mit der AWS CLI arbeiten, können Sie eine Standardregion für die Konfiguration verwenden, anstatt `--region` im Befehl anzugeben. Weitere Informationen zur Verwendung der AWS CLI finden Sie unter Schnelleinrichtung im AWS Command Line Interface-Benutzerhandbuch.

Durchführen eines Hauptversions-Upgrades

1. Führen Sie ein Upgrade des DB-Clusters von Aurora PostgreSQL auf eine neue Hauptversion durch. Weitere Informationen finden Sie unter Upgrade der Aurora-PostgreSQL-Engine auf eine neue Hauptversion.
2. Starten Sie die Writer-Instance des Clusters neu, damit die Parametereinstellungen wirksam werden.

Nach dem Durchführen von Upgrades auf eine neue Hauptversion

Nach einem Upgrade der Hauptversion auf eine neue Aurora-PostgreSQL-Version ist der `IDENTITY`-Wert in Tabellen mit einer `IDENTITY`-Spalte möglicherweise größer (+32) als vor dem Upgrade. Wenn die nächste Zeile in solche Tabellen eingefügt wird, springt der generierte Identitätsspaltenwert demzufolge zur Zahl +32 und die Sequenz beginnt von dort aus. Dieser Zustand wirkt sich nicht negativ auf die Funktionen Ihres Babelfish-DB-Clusters aus. Wenn Sie möchten, können Sie das Sequenzobjekt jedoch auf der Grundlage des Maximalwerts der Spalte zurücksetzen. Stellen Sie dazu eine Verbindung zum T-SQL-Port auf Ihrer Babelfish-Writer-Instance mit `sqlcmd` oder einem anderen SQL-Server-Client her. Weitere Informationen finden Sie unter Verbinden mit Ihrem DB-Cluster mithilfe eines SQL Server-Clients.

```
sqlcmd -S bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
     sa -P ******** -d dbname
```

Wenn die Verbindung hergestellt ist, verwenden Sie den folgenden SQL-Befehl, um Anweisungen zu generieren, mit denen Sie das zugehörige Sequenzobjekt als Ausgangswert verwenden können. Dieser SQL-Befehl funktioniert sowohl für Babelfish-Konfigurationen mit einer einzelnen Datenbank als auch mit mehreren Datenbanken. Weitere Informationen zu diesen beiden Bereitstellungsmodellen finden Sie unter Verwenden von Babelfish mit einer einzigen Datenbank oder mehreren Datenbanken.

```
DECLARE @schema_prefix NVARCHAR(200) = ''
IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
    SET @schema_prefix = db_name() + '_'
SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id)
    + '.' + tables.name + ''', ''' + columns.name + '''),(select max(' + columns.name +
 ')
    FROM ' + schema_name(tables.schema_id) + '.' + tables.name + '));
    'FROM sys.tables tables JOIN sys.columns
    columns ON tables.object_id = columns.object_id
    WHERE columns.is_identity = 1
GO
```

Die Abfrage generiert eine Reihe von SELECT-Anweisungen, die Sie dann ausführen können, um den maximalen IDENTITY-Wert zurückzusetzen und etwaige Lücken zu schließen. Im Folgenden wird die Ausgabe gezeigt, wenn die SQL-Server-Beispieldatenbank Northwind verwendet wird, die auf einem Babelfish-Cluster ausgeführt wird.

```
-----------------------------------------------------
SELECT setval(pg_get_serial_sequence('northwind_dbo.categories', 'categoryid'),(select
 max(categoryid)
    FROM dbo.categories));

SELECT setval(pg_get_serial_sequence('northwind_dbo.orders', 'orderid'),(select
 max(orderid)
    FROM dbo.orders));


SELECT setval(pg_get_serial_sequence('northwind_dbo.products', 'productid'),(select
 max(productid)
    FROM dbo.products));


SELECT setval(pg_get_serial_sequence('northwind_dbo.shippers', 'shipperid'),(select
 max(shipperid)
    FROM dbo.shippers));


SELECT setval(pg_get_serial_sequence('northwind_dbo.suppliers', 'supplierid'),(select
 max(supplierid)
    FROM dbo.suppliers));


(5 rows affected)
```

Führen Sie die Anweisungen nacheinander aus, um die Sequenzwerte zurückzusetzen.

Beispiel: Durchführen eines Upgrades des Babelfish-DB-Clusters auf eine Hauptversion

In diesem Beispiel finden Sie eine Reihe von AWS CLI-Befehlen, die erklären, wie ein DB-Cluster von Aurora PostgreSQL 13.6.4, auf dem Babelfish Version 1.2.2 ausgeführt wird, zu Aurora PostgreSQL 14.6 aktualisiert wird. Zunächst erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe für Aurora PostgreSQL 14. Als Nächstes ändern Sie die Parameterwerte so, dass sie denen von Aurora PostgreSQL Version 13 als Quelle entsprechen. Schließlich führen Sie das Upgrade durch, indem Sie den Quell-Cluster ändern. Weitere Informationen finden Sie unter Einstellungen der DB-Cluster-Parametergruppe für Babelfish. In diesem Thema finden Sie auch Informationen zur Verwendung der AWS-Managementkonsole zur Durchführung des Upgrades.

Verwenden Sie den CLI-Befehl create-db-cluster-parameter-group, um die DB-Cluster-Parametergruppe für die neue Version zu erstellen.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
    --db-parameter-group-family aurora-postgresql14 \
    --description 'New custom parameter group for upgrade to new major version' \
    --region us-west-1
```

Wenn Sie diesen Befehl ausführen, wird die benutzerdefinierte DB-Cluster-Parametergruppe in der AWS-Region erstellt. Die Ausgabe entspricht weitgehend der folgenden.

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14",
        "DBParameterGroupFamily": "aurora-postgresql14",
        "Description": "New custom parameter group for upgrade to new major version",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-west-1:111122223333:cluster-
pg:docs-lab-babelfish-apg-14"
    }
}
```

Weitere Informationen finden Sie unter Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora.

Verwenden Sie den CLI-Befehl modify-db-cluster-parameter-group, um die Einstellungen so zu ändern, dass sie dem Quell-Cluster entsprechen.

Für Windows:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name docs-lab-
babelfish-apg-14 ^
  --parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_precision,ParameterValue=38,ApplyMethod=pen
reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_scale,ParameterValue=8,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.database_name,ParameterValue=babelfish_db,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.default_locale,ParameterValue=en-
US,ApplyMethod=pending-reboot" ^
```

```
 "ParameterName=babelfishpg_tsql.migration_mode,ParameterValue=single-
db,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.server_collation_name,ParameterValue=sql_latin1_general_cp1_ci
reboot"
```

Die Antwort sieht in etwa so aus:

```
{
    "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14"
}
```

Verwenden Sie den CLI-Befehl modify-db-cluster, um den Cluster so zu ändern, dass er die neue Version und die neue benutzerdefinierte DB-Cluster-Parametergruppe verwendet. Sie geben auch das Argument `--allow-major-version-upgrade` an, wie im folgenden Beispiel gezeigt.

```
aws rds modify-db-cluster \
--db-cluster-identifier docs-lab-bfish-apg-14 \
--engine-version 14.6 \
--db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
--allow-major-version-upgrade \
--region us-west-1 \
--apply-immediately
```

Verwenden Sie den CLI-Befehl reboot-db-instance, um die Writer-Instance des Clusters neu zu starten, damit die Parametereinstellungen wirksam werden.

```
aws rds reboot-db-instance \
--db-instance-identifier docs-lab-bfish-apg-14-instance-1\
--region us-west-1
```

Verwenden des Babelfish-Produktversionsparameters

Ein neuer Grand Unified Configuration (GUC)-Parameter namens `babelfishpg_tds.product_version` wird in den Versionen Babelfish 2.4.0 und 3.1.0 eingeführt. Mit diesem Parameter können Sie die Versionsnummer des SQL-Server-Produkts als Ausgabe von Babelfish festlegen.

Der Parameter ist eine 4-teilige Versions-ID-Zeichenfolge und jeder Teil sollte durch „.“ getrennt werden.

Syntax

```
Major.Minor.Build.Revision
```

- Hauptversion: Eine Zahl zwischen 11 und 16.
- Nebenversion: Eine Zahl zwischen 0 und 255.
- Build-Version: Eine Zahl zwischen 0 und 65535.
- Revision: 0 und eine beliebige positive Zahl.

Konfigurieren des Babelfish-Produktversionsparameters

Sie müssen die Cluster-Parametergruppe verwenden, um den Parameter babelfishpg_tds.product_version in der Konsole festzulegen. Weitere Informationen zum Ändern des DB-Cluster-Parameters finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

Wenn Sie den Produktversionsparameter auf einen ungültigen Wert festlegen, wird die Änderung nicht wirksam. Die Konsole zeigt Ihnen möglicherweise den neuen Wert an, der Parameter behält jedoch den vorherigen Wert bei. Details zu den Fehlermeldungen können Sie der Engine-Protokolldatei entnehmen.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Betroffene Abfragen und Parameter

Abfrage/Parameter	Ergebnis	Wirksamkeit
SELECT @@VERSION	Gibt die benutzerdefinierte SQL-Server-Version zurück (babelfishpg_tsql.version value = Standard)	Sofort
SELECT SERVERPROPERTY('ProductVersion')	Gibt die benutzerdefinierte SQL-Server-Version zurück	Sofort
SELECT SERVERPROPERTY('ProductMajorVersion')	Gibt die Hauptversion der benutzerdefinierte SQL-Server-Version zurück	Sofort
VERSION-Token in der PRELOGIN-Antwortnachricht	Der Server gibt PRELOGIN-Nachrichten mit der benutzerdefinierten SQL-Server-Version zurück	Wird wirksam, wenn ein Benutzer eine neue Sitzung erstellt
SQLServerVersion in LoginAck bei Verwendung von JDBC	DatabaseMetadata.getDatabaseProductVersion() gibt die benutzerdefinierte SQL-Server-Version zurück	Wird wirksam, wenn ein Benutzer eine neue Sitzung erstellt

Schnittstelle mit dem Parameter babelfishpg_tsql.version

Sie können die Ausgabe von @@VERSION mit den Parametern babelfishpg_tsql.version und babelfishpg_tds.product_version festlegen. In den folgenden Beispielen wird gezeigt, wie sich diese beiden Parameter zueinander verhalten.

- Wenn der Parameter babelfishpg_tsql.version „default“ ist und der Parameter babelfishpg_tds.product_version 15.0.2000.8 lautet.
  - Ausgabe von @@version – 15.0.2000.8.
- Wenn der Parameter babelfishpg_tsql.version auf 13.0.2000.8 festgelegt ist und der Parameter babelfishpg_tds.product_version 15.0.2000.8 lautet.

- Ausgabe von @@version – 13.0.2000.8.

Referenz für Babelfish for Aurora PostgreSQL

Themen

- Unterstützte Funktionen in Babelfish nach Version
- Nicht unterstützte Funktionen in Babelfish
- Arbeiten mit Prozeduren von Babelfish für Aurora PostgreSQL

Unterstützte Funktionen in Babelfish nach Version

In den folgenden Tabellen finden Sie die Liste aller Funktionen, die von verschiedenen Babelfish-Versionen unterstützt werden. Listen der nicht unterstützten Funktionen finden Sie unter Nicht unterstützte Funktionen in Babelfish. Weitere Informationen zu verschiedenen Babelfish-Versionen finden Sie unter Versionshinweise für Aurora PostgreSQL.

Aurora- und PostgreSQL-Funktionen

In der folgenden Tabelle finden Sie Aurora- and PostgreSQL-Funktionen, die von verschiedenen Babelfish-Versionen unterstützt werden.

| | | |

Aurora- und PostgreSQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	J	N	N	N	N	N	N	nein
pgvector	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
Zero-Downtime-Patching (ZDP) (Patchen ohne Ausfallzeiten)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	nein

T-SQL-Funktionen oder -Syntax

In der folgenden Tabelle finden Sie von verschiedenen Babelfish-Versionen unterstützte T-SQL-Funktionen oder -Syntax.

T-SQL-Funktionen oder -Syntax	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
4-teilige Objektnamenreferenzen für SELECT-Anweisungen	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	N	nein
Syntax ALTER AUTHORIZATION zum Ändern des Datenbankeigentümers	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
ALTER DATABASE *<db_name>* MODIFY NAME = *<new_db_name>*	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
ALTER FUNCTION																																	

| | | |

T-SQL-Funktionen oder -Syntax	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
CREATE INDEX ... ON partition _scheme_name (partition_column_ name)	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
CREATE anstelle von Triggern (DML) in SQL-Server-Ansichten	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
CREATE PARTITION FUNCTION	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
CREATE PARTITION SCHEME	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
CREATE ROLE	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
CREATE TABLE ... ON partition _scheme_name (partition_column_ name)	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
CREATE TRIGGER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
CREATE OR ALTER VIEW	J	N	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein

| | | |

T-SQL-Funktionen oder -Syntax	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DEFAULT-Schlüsselwort in Aufrufen von gespeicherten Prozeduren und Funktionen	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
DBCC CHECKIDENT	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
DROP DATABASE	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	nein
DROP IF EXISTS (für SCHEMA-, DATABASE- und USER-Objekte)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
DROP INDEX-Index ON schema.table	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
DROP INDEX schema.table.index	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
DROP PARTITION FUNCTION	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
DROP PARTITION SCHEME	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein

T-SQL-Funktionen oder -Syntax	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DROP ROLE	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
ENABLE/DISABLE TRIGGER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
FULL TEXT SEARCH	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
Volltextsuche mit CONTAINS-Klausel	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
Räumliche Datentypen für Geometrie und Geografie	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
GRANT	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
GRANT-Berechtigung [,...n] ON SCHEMA	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
GUC babelfish pg_tds.product_version	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	Nein
Bezeichner mit führendem Punktzeichen	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja

| | | |

T-SQL-Funktionen oder -Syntax	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SELECT FOR JSON AUTO	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
SET BABELFISH _SHOWPLAN_ALL ON (und OFF)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
SET BABELFISH _STATISTICS PROFILE ON (OFF)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
SET CONTEXT_I NFO	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	N	nein
SET LOCK_TIME OUT	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
SET NO_BROWSE TABLE	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein
SET rowcount	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	N	nein
SET SHOWPLAN_ ALL	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein
SET STATISTICS IO	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein

| | | |

| | | |

Integrierte T-SQL-Funktionen

In der folgenden Tabelle finden Sie integrierte T-SQL-Funktionen, die von verschiedenen Babelfish-Versionen unterstützt werden.

Integrierte T-SQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	1.0
APP_NAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	Nein
ATN2	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	Nein
CHARINDEX	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
CHOOSE	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
COL_LENGTH	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
COL_NAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
COLUMNS_UPDATED	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
COLUMNPROPERTY (nur CharMaxLen, AllowsNull)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
CONCAT_WS																															

Integrierte T-SQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
CONTEXT_INFO	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	N	nein
CURSOR_STATUS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
DATABASE_PRINCIPAL_ID	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	N	nein
DATEADD	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	nein
DATEDIFF	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	nein
DATEDIFF_BIG	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	nein
DATEFROMPARTS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
DATENAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	nein
DATEPART	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	nein

Integrierte T-SQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
DATETIMEFROMPARTS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
DATETIME2FROMPARTS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	Nein
DATETIMEOFFSETFROMPARTS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	Nein
DATETRUNC	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	J	J	J	J	J	J	N	N	N	N	N	N	Nein
DATE_BUCKET	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	J	J	J	J	J	J	N	N	N	N	N	N	Nein
EOMONTH	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	J	J	J	J	J	J	N	N	N	N	N	N	Nein
EXECUTE AS CALLER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	Nein
fn_listextendedproperty	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	J	N	N	N	N	N	N	Nein
FOR JSON	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	Nein
FULLTEXTSERVICEPROPERTY	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja

Integrierte T-SQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
HAS_DBACCESS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
HAS_PERMS_BY_NAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
HOST_NAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein
HOST_ID	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	J	N	N	N	N	N	Nein
IDENTITY	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Nein
IS_MEMBER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
IS_ROLEMEMBER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
IS_SRVROLEMEMBER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
ISJSON	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
JSON_MODIFY	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	Nein

Integrierte T-SQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
JSON_QUERY	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
JSON_VALUE	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
NEXT VALUE FOR	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	ein
OBJECT_DEFINITION	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	ein
OBJECT_SCHEMA_NAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	ein
OPENJSON	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
OPENQUERY	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
ORIGINAL_LOGIN	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
PARSENAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	J	N	N	N	N	N	N	ein
PATINDEX	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja

Integrierte T-SQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
ROWCOUNT_BIG	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	Nein	
SCHEMA_NAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja	
SESSION_CONTEXT	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	Nein	
SESSION_USER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja	
SID_BINARY (gibt immer NULL zurück)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein	
SMALLDATETIMEFROMPARTS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	J	N	N	N	N	N	Nein	
SQUARE	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja	
STR	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	Nein	
STRING_AGG	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	J	J	J	J	N	N	N	N	N	N	N	N	Nein	
STRING_SPLIT	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja	

Integrierte T-SQL-Funktionen	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
SUSER_SID	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
SUSER_SNAME	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
SWITCHOFFSET	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
SYSTEM_USER	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	nein
TIMEFROMPARTS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	J	J	J	J	J	J	J	N	N	N	N	N	N	nein
TODATETIMEOFFSET	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
TO_CHAR	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	nein
TRIGGER_NESTLEVEL (nur ohne Argumente)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
TRY_CONVERT	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	nein
TYPE_ID	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein

| | | |

| T-SQL-INFORMATION_SCHEMA-Kataloge | 5.5.4 | 4.4.4 | 4.4.4 | 4.4.3 | 3.3.3 | 3.3.3 | 3.3.3 | 3.3.2 | 2.2.2 | 2.2.2 | 2.2

| Systemdefinierte T-SQL-@@-Variablen | 5 5 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3 3 2 2 2

| Systemdefinierte T-SQL-@@-Variablen | 5.5 | 5.4 | 4.4 | 4.4 | 4.4 | 4.4 | 4.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 | 3.2 | 2.2 | 2.2 | 2

Systemgespeicherte T-SQL-Prozeduren

In der folgenden Tabelle finden Sie von verschiedenen Babelfish-Versionen unterstützte systemgespeicherte T-SQL-Prozeduren.

Systemgespeicherte T-SQL-Prozeduren	5.5.4	5.4.4	5.4.3	5.3.3	5.3.2	5.2.2	5.2.1	5.1.0
sp_addextendedproperty	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Nein
sp_addlinkedserver	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Nein
sp_addlinkedsrvlogin	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Nein
sp_addrole	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Nein
sp_addrolemember	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Nein
sp_babelfish_volatility	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Nein
sp_column_privileges	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
sp_columns	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
sp_columns_100								

Systemgespeicherte T-SQL-Prozeduren	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_columns_managed	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursor	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursor_list	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursorclose	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursorexecute	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursorfetch	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursoropen	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursoroption	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_cursorprepare	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja

| | | |

Systemgespeicherte T-SQL-Prozeduren	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_droprole	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	Nein	
sp_droprolemember	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	Nein	
sp_dropserver	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	Nein	
sp_enum_oledb_providers	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	J	N	N	N	N	N	Nein
sp_execute	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_execute_postgresql(CREATE, ALTER, DROP)	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	J	N	N	N	N	N	Nein
sp_executesql	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_fkeys	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_getapplock	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_helpdb	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja

Systemgespeicherte T-SQL-Prozeduren	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sp_helpdbfixedrole	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	Nein
sp_helplinkedsrvlogin	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	Nein
sp_helprole	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_helprolemember	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_helpsrvrolemember	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein
sp_helpuser	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_linkedservers	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	Nein
sp_oledb_ro_usrname	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_pkeys	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sp_prefix	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein

| | | |

| | | |

| | | |

| | | |

| Von der OBJECTPROPERTY-Systemfunktion unterstützte T-SQL-Eigenschaften | 5.5.4 | 4.4.4 | 4.4.4 | 4.4.4 | 4.4.3 | 3.3.3 | 3.3.3 | 3.3.3 | 3.3.2 | 2.2.2 | 2.2.2 | 2.2

| | | |

Von der SERVERPROPERTY-Funktion unterstützte T-SQL-Eigenschaften	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
IsPolyBaseInstalled	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
IsSingleUser	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
IsXTPSupported	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
Japanese_CI_AI	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
Japanese_CI_AS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
Japanese_CS_AS	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
LicenseType	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
MachineName	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Nein
ProductLevel	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	Nein

| | | |

Von Babelfish unterstützte SQL-Server-Ansichten	5.1.0
information_schema.routines	Nein
information_schema.schemata	Nein
information_schema.sequences	Nein
sys.all_columns	Ja
sys.all_objects	Ja
sys.all_parameters	Nein
sys.all_sql_modules	Ja
sys.all_views	Ja
sys.columns	Ja

| Von Babelfish unterstützte SQL-Server-Ansichten | 5.5.4 | 5.4.4 | 5.4.4 | 5.4.4 | 5.4.4 | 5.4.3 | 5.3.3 | 5.3.3 | 5.3.3 | 5.3.3 | 5.3.3 | 5.3.3 | 5.3.3 | 5.3.3 | 5.3.2 | 5.2.2 | 5.2.2 | 5.2

Von Babelfish unterstützte SQL-Server-Ansichten	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sys.dm_exec_sessions	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.dm_hadr_database_replica_states	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.dm_os_host_info	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.dm_os_sys_info	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
sys.endpoints	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.extended_properties	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	J	J	J	J	J	J	N	N	N	N	N	N	N	nein
sys.indexes	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.partitions	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
sys.partition_functions	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein

Von Babelfish unterstützte SQL-Server-Ansichten	5	5	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	1.0
sys.partition_parameters	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
sys.partition_range_values	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
sys.partition_schemes	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
sys.schemas	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.server_principals	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.server_role_members	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	nein
sys.sql_modules	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.sysconfigures	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja
sys.syscurconfigs	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	Ja

Von Babelfish unterstützte SQL-Server-Ansichten	5.5.4.4.4.4.4.4.4.3.3.3.3.3.3.3.3.3.3.2.2.2.2.2.2.2.2.2.2.2.2.2.2.2.1.0
sys.syslogins	J J J J J J J J J J J J J J J J J J J N J J J J J J J N N N N N Nein
sys.sysprocesses	J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J Ja
sys.sysusers	J J J J J J J J J J J J J J J J J J J N J J J J J J J N N N N N Nein
sys.table_types	J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J Ja
sys.tables	J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J Ja
sys.types	J J J J J J J J J J J J J J J J J J J J J J J J J J J J J N N N Nein
sys.xml_schema_collections	J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J Ja
syslanguages	J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J Ja
sysobjects.crdate	J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J J N N Nein

Nicht unterstützte Funktionen in Babelfish

In den folgenden Tabellen und Listen finden Sie Funktionen, die derzeit in Babelfish nicht unterstützt werden. Updates für Babelfish sind in Aurora-PostgreSQL-Versionen enthalten. Weitere Informationen finden Sie unter Versionshinweise für Aurora PostgreSQL.

Themen

- Derzeit nicht unterstützte Funktionen
- Nicht unterstützte Einstellungen
- Befehle, die nicht unterstützt werden
- Nicht unterstützte Spaltennamen oder Attribute
- Nicht unterstützte Datentypen
- Nicht unterstützte Objekttypen
- Nicht unterstützte Funktionen
- Nicht unterstützte Syntax

Derzeit nicht unterstützte Funktionen

In der Tabelle finden Sie Informationen zu bestimmten Funktionen, die derzeit nicht unterstützt werden.

Funktionalität oder Syntax	Beschreibung
Assembly-Module und SQL Common Language Runtime (CLR)-Routinen	Funktionen im Zusammenhang mit Baugruppenmodulen und CLR-Routinen werden nicht unterstützt.
Spalten-Attribute	ROWGUIDCOL, SPARSE, FILESTREAM und MASKED werden nicht unterstützt.
Eingeschlossene	Enthaltene Datenbanken mit Logins, die auf Datenbankebene und nicht auf Serverebene authentifiziert sind, werden nicht unterstützt.

Funktionalität oder Syntax	Beschreibung
Datenbankübergreifende DDL	Die Ausführung von DDL-Anweisungen, die auf Objekte in mehreren Datenbanken verweisen oder diese bearbeiten, wird derzeit noch nicht unterstützt.
Cursoren (aktualisierbar)	Aktualisierbare Cursor werden nicht unterstützt.
Cursoren (global)	GLOBALE Cursor werden nicht unterstützt.
Cursor (Verhalten abrufen)	Die folgenden Cursor-Abrufverhalten werden nicht unterstützt: FETCH PRIOR, FIRST, LAST, ABSOLUTE, und RELATIVE
Cursor-typisierte Ausgabepa rameter	Variablen und Parameter vom Cursor-Typ werden für Ausgabeparameter nicht unterstützt (ein Fehler wird ausgelöst).
Cursor-Optionen	SCROLL, KEYSET, DYNAMIC, FAST_FORWARD, SCROLL_LOCKS, OPTIMISTIC, TYPE_WARNING und FOR UPDATE
Datenverschlüsselung	Die Datenverschlüsselung wird nicht unterstützt.
Datenschichtanwendungen (DAC)	Import- oder Exportvorgänge für Datenschichtanwendungen (DAC) mit DAC-Paketdateien (.dacpac) oder DAC-Backu pdateien (.bacpac) werden nicht unterstützt.
DBCC-Befehle	Microsoft SQL Server Database Console Commands (DBCC) werden nicht unterstützt. DBCC CHECKIDENT wird in Babelfish 3.4.0 und höheren Versionen unterstützt.
DROP FALLS VORHANDEN	Diese Syntax wird für USER- und SCHEMA-Objekte nicht unterstützt. Es wird für die Objekte TABLE, VIEW, PROCEDURE, FUNCTION und DATABASE unterstützt.
Verschlüsselung	Eingebaute Funktionen und Anweisungen unterstützen keine Verschlüsselung.
ENCRYPT_CLIENT_CERT-Verbindungen	Verbindungen mit Clientzertifikaten werden nicht unterstützt.

Funktionalität oder Syntax	Beschreibung
EXECUTE AS-Anweisung	Diese Aussage wird nicht unterstützt.
EXECUTE AS SELF-Klausel	Diese Klausel wird in Funktionen, Prozeduren oder Triggern nicht unterstützt.
EXECUTE AS USER-Klausel	Diese Klausel wird in Funktionen, Prozeduren oder Triggern nicht unterstützt.
Fremdschlüsseleinschränkungen, die auf den Datenbanknamen verweisen	Fremdschlüsseleinschränkungen, die auf den Datenbanknamen verweisen, werden nicht unterstützt.
FORMAT	Benutzerdefinierte Typen werden nicht unterstützt.
Funktionsdeklarationen mit mehr als 100 Parametern	Funktionsdeklarationen, die mehr als 100 Parameter enthalten, werden nicht unterstützt.
Funktionsaufrufe, die DEFAULT als Parameterwert enthalten	DEFAULT ist kein unterstützter Parameterwert für einen Funktionsaufruf. DEFAULT als Parameterwert für einen Funktionsaufruf wird ab Babelfish 3.4.0 und höheren Versionen unterstützt.
Funktionen, extern definiert	Externe Funktionen, einschließlich SQL CLR-Funktionen, werden nicht unterstützt.
Globale temporäre Tabellen (Tabellen mit Namen, die mit ## beginnen)	Globale temporäre Tabellen werden nicht unterstützt.
Graph-Funktion	Alle SQL-Graph-Funktionalität werden nicht unterstützt.
Allgemeine erweiterte gespeicherte Prozeduren	Gespeicherte Systemprozeduren, die eine Schnittstelle von einer Instance von SQL Server zu externen Programmen für verschiedene Wartungsaktivitäten bereitstellen, werden nicht unterstützt. Dazu gehören `xp_cmdshell` und andere gespeicherte Systemprozeduren. Weitere Informationen finden Sie unter Allgemeine erweiterte gespeicherte Prozeduren.

Funktionalität oder Syntax	Beschreibung
Bezeichner (Variablen oder Parameter) mit mehreren führenden @ Zeichen	Identifikatoren, die mit mehr als einem führenden @ beginnen, werden nicht unterstützt.
Bezeichner, Tabellen- oder Spaltennamen, welche die Zeichen @ oder]] enthalten	Tabellen- oder Spaltennamen, die ein @- Zeichen oder eckige Klammern enthalten, werden nicht unterstützt.
Eingebreitete Indizes	Inline-Indizes werden nicht unterstützt.
Aufrufen einer Prozedur, deren Name in einer Variablen enthalten ist	Die Verwendung einer Variablen als Prozedurnamen wird nicht unterstützt.
Materialisierte Ansichten	Materialisierte Ansichten werden nicht unterstützt.
NICHT FÜR REPLICATION-Klausel	Diese Syntax wird akzeptiert und ignoriert.
ODBC Escape-Funktionen	ODBC-Escape-Funktionen werden nicht unterstützt.
Prozeduraufrufe mit DEFAULT als Parameterwert	DEFAULT ist kein unterstützter Parameterwert. DEFAULT als Parameterwert für einen Funktionsaufruf wird ab Babelfish 3.4.0 und höheren Versionen unterstützt.
Verfahrenserklärungen mit mehr als 100 Parametern	Deklarationen mit mehr als 100 Parametern werden nicht unterstützt.
Prozeduren, extern definiert	Extern definierte Prozeduren, einschließlich SQL CL), werden nicht unterstützt.
Versionierung der Prozedur	Prozedur Versionierung wird nicht unterstützt.
Prozeduren MIT RECOMPILE	WITH RECOMPILE (wenn es in Verbindung mit den Anweisungen DECLARE und EXECUTE verwendet wird) wird nicht unterstützt.

Funktionalität oder Syntax	Beschreibung
Remote-Objektreferenzen	Die Ausführung von gespeicherten Prozeduren auf mit Babelfish verknüpften Servern wird nicht unterstützt. Vierteilige Objektnamen funktionieren nur zum Lesen und nicht zum Ändern der Remote-Tabelle. Ein UPDATE kann in der FROM-Klausel auf eine entfernte Tabelle verweisen, ohne sie zu ändern. Weitere Informationen finden Sie unter Babelfish unterstützt verknüpfte Server.
Sicherheit auf Zeilenebene	Die Sicherheit auf Zeilenebene mit CREATE SECURITY POLICY und Funktionen mit Inline-Tabellenwert wird nicht unterstützt.
Service-Broker-Funktionalität	Die Service-Broker-Funktionalität wird nicht unterstützt.
SESSIONPROPERTY	Nicht unterstützte Eigenschaften: ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, ARITHABORT, CONCAT_NULL_YIELDS_NULL und NUMERIC_ROUNDABORT
SET-LANGUAGE	Diese Syntax wird mit keinem anderen Wert als `english` oder `us_english` unterstützt.
SP_CONFIGURE	Diese gespeicherte Systemprozedur wird nicht unterstützt.
SQL-Schlüsselwort `SPARSE`	Das Schlüsselwort SPARSE wird akzeptiert und ignoriert.
Syntax des Tabellenwert-Konstruktors (FROM-Klausel)	Die nicht unterstützte Syntax gilt für eine abgeleitete Tabelle, die mit der FROM-Klausel erstellt wurde.
Temporäre Tabellen	Temporäre Tabellen werden nicht unterstützt.
Temporäre Verfahren werden nicht automatisch gelöscht	Diese Funktion wird nicht unterstützt.
Trigger, extern definiert	Diese Trigger werden nicht unterstützt, einschließlich SQL Common Language Runtime (CLR).

Funktionalität oder Syntax	Beschreibung
Ohne SCHEMABINDING-Klausel	Das Erstellen einer Ansicht ohne SCHEMABINDING wird nicht unterstützt, die Ansicht wird jedoch erstellt, als ob WITH SCHEMABINDING angegeben wurde. Die Verwendung von SCHEMABINDING beim Erstellen von Funktionen, Prozeduren und Triggern wird stillschweigend ignoriert.

Nicht unterstützte Einstellungen

Die folgenden Einstellungen werden nicht unterstützt:

- SETZE ANSI_NULL_DFLT_OFF EIN
- SETZE ANSI_NULL_DFLT_ON AUS
- SETZE ANSI_PADDING AUS
- SETZE ANSI_WARNINGS AUS
- AKTIVIEREN SIE ARITHABORT AUS
- SETZE ARITHIGNORE AUF
- SETZE CURSOR_CLOSE_ON_COMMIT AUF
- SET NUMERIC_ROUNDABORT ON
- SET PARSEONLY ON (Befehl funktioniert nicht wie erwartet)
- SET FMTONLY ON (Der Befehl funktioniert nicht wie erwartet. Er unterdrückt nur die Ausführung von SELECT-Anweisungen, von anderen jedoch nicht.)

Befehle, die nicht unterstützt werden

Für folgende Befehle werden bestimmte Funktionen nicht unterstützt:

- SIGNATURE HINZUFÜGEN
- DATENBANK ÄNDERN, DATENBANK ÄNDERN
- BACKUP/RESTORE DATABASE/LOG
- BACPAC und DACPAC FILES RESTORE
- CREATE, ALTER, DROP AUTHORIZATION. ALTER AUTHORIZATION wird für Datenbankobjekte unterstützt.

- ERSTELLE, ÄNDERUNG, DROP VERFÜGBARKEITSGRUPPE
- BROKER-PRIORITÄT ERSTELLEN, ÄNDERN, LÖSCHEN
- SPALTENVERSCHLÜSSELUNGSSCHLÜSSEL ERSTELLEN, ÄNDERN, LÖSCHEN
- DATENBANK-VERSCHLÜSSELUNGSSCHLÜSSEL ERSTELLEN, ÄNDERN, LÖSCHEN
- ZERTIFIKAT ERSTELLEN, ÄNDERN, LÖSCHEN, SICHERN
- CREATE AGGREGATE
- VERTRAG ERSTELLEN
- CHECKPOINT

Nicht unterstützte Spaltennamen oder Attribute

Die folgenden Spaltennamen werden nicht unterstützt:

- $IDENTITY
- $ROWGUID
- IDENTITYCOL

Nicht unterstützte Datentypen

Die folgenden Datentypen werden nicht unterstützt:

- HIERARCHYID

Nicht unterstützte Objekttypen

Die folgenden Objekttypen werden nicht unterstützt:

- SPALTEN-MASTERSCHLÜSSEL
- EXTERNE DATENQUELLE ERSTELLEN, ÄNDERN
- DATENBANK-AUDIT-SPEZIFIKATION ERSTELLEN, ÄNDERN, LÖSCHEN
- EXTERNE BIBLIOTHEK ERSTELLEN, ÄNDERN, LÖSCHEN
- SERVER-AUDIT ERSTELLEN, ÄNDERN, LÖSCHEN
- ERSTELLEN, ÄNDERN, LÖSCHEN DER SERVER-AUDIT-SPEZIFIKATION
- SYMMETRISCHEN SCHLÜSSEL ERSTELLEN, ÄNDERN, LÖSCHEN, ÖFFNEN/SCHLIESSEN

- CREATE, DROP DEFAULT
- ANMELDEINFORMATION
- KRYPTOGRAFISCHER ANBIETER
- DIAGNOSESITZUNG
- Indizierte Ansichten
- SERVICE-MASTERSCHLÜSSELS
- SYNONYM

Nicht unterstützte Funktionen

Folgende integrierte Funktionen werden nicht unterstützt:

Aggregationsfunktionen

- APPROX_COUNT_DISTINCT
- CHECKSUM_AGG
- GROUPING_ID
- STRING_AGG unter Verwendung der WITHIN GROUP-Klausel

Kryptografische Funktionen

- CERTENCODED-Funktion
- CERTID-Funktion
- CERTPROPERTY-Funktion

Metadatenfunktionen

- COLUMNPROPERTY
- TYPEPROPERTY
- SERVERPROPERTY-Funktion – Die folgenden Eigenschaften werden nicht unterstützt:
  - BuildClrVersion
  - ComparisonStyle
  - ComputerNamePhysicalNetBIOS

- HadrManagerStatus
- InstanceDefaultDataPath
- InstanceDefaultLogPath
- IsClustered
- IsHadrEnabled
- LCID
- NumLicenses
- ProcessID
- ProductBuild
- ProductBuildType
- ProductUpdateReference
- ResourceLastUpdateDateTime
- ResourceVersion
- ServerName
- SqlCharSet
- SqlCharSetName
- SqlSortOrder
- SqlSortOrderName
- FilestreamShareName
- FilestreamConfiguredLevel
- FilestreamEffectiveLevel

Sicherheitsfunktionen

- CERTPRIVATEKEY
- LOGINPROPERTY

Anweisungen, Operatoren, andere Funktionen

- EVENTDATA-Funktion
- GET_TRANSMISSION_STATUS
- OPENXML

Nicht unterstützte Syntax

Die folgende Syntax wird nicht unterstützt:

- ALTER DATABASE
- ALTER DATABASE SCOPED CONFIGURATION
- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER INDEX
- ALTER PARTITION FUNCTION
- ALTER PARTITION SCHEME
- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- ALTER SERVICE, BACKUP/RESTORE SERVICE MASTER KEY-Klausel
- BEGIN CONVERSATION TIMER
- STARTEN SIE VERTEILTE TRANSAKTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- ANWENDUNGSROLLE ERSTELLEN, ÄNDERN, LÖSCHEN
- BAUGRUPPE ERSTELLEN, ÄNDERN, LÖSCHEN
- ASYMMETRISCHEN SCHLÜSSEL ERSTELLEN, ÄNDERN, LÖSCHEN
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- EVENT-SITZUNG ERSTELLEN, ÄNDERN, LÖSCHEN
- CREATE, ALTER, DROP EXTERNAL LANGUAGE
- EXTERNEN RESSOURCENPOOL ERSTELLEN, ÄNDERN, LÖSCHEN
- ERSTELLEN, ÄNDERN, LÖSCHEN VOLLTEXTKATALOG
- ERSTELLEN, ÄNDERN, LÖSCHEN VOLLTEXTINDEX

- ERSTELLEN, ÄNDERN, LÖSCHEN VOLLTEXT-STOPLIST
- CREATE, ALTER, DROP MESSAGE TYPE
- ERSTELLEN, ÄNDERN, LÖSCHEN, ÖFFNEN/SCHLIESSEN, HAUPTSCHLÜSSEL SICHERN/WIEDERHERSTELLEN
- CREATE, ALTER, DROP QUEUE
- RESSOURCENGOUVERNEUR ERSTELLEN, ÄNDERN, LÖSCHEN
- CREATE, ALTER, DROP RESOURCE POOL
- ERSTELLEN, ÄNDERN, BENUTZER LÖSCHEN
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- CREATE, ALTER, DROP SELECTIVE XML INDEX clause
- CREATE, ALTER, DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- REGEL ERSTELLEN/LÖSCHEN
- CREATE, DROP WORKLOAD CLASSIFIER
- WORKLOAD-GRUPPE ERSTELLEN, ÄNDERN, LÖSCHEN
- ALTER TRIGGER
- TABELLE ERSTELLEN… GRANT-Klausel
- TABELLE ERSTELLEN… IDENTITY-Klausel
- CREATE USER – Diese Syntax wird nicht unterstützt. Die PostgreSQL-Anweisung CREATE USER erstellt keinen Benutzer, der der SQL Server CREATE USER Syntax entspricht. Weitere Informationen finden Sie unter T-SQL-Unterschiede bei Babelfish.
- DENY
- END, MOVE CONVERSATION
- EXECUTE with AS LOGIN or AT option
- GET CONVERSATION GROUP
- GROUP BY ALL clause
- GROUP BY CUBE clause

- GROUP BY ROLLUP clause
- INSERT... DEFAULT VALUES
- MERGE
- READTEXT
- REVERT
- SELECT TOP x PERCENT WHERE x **<>** 100
- SELECT TOP... WITH TIES
- SELECT... FOR BROWSE
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SELECT... FOR XML PATH
- SEND
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NUMERIC_ROUNDABORT ON
- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS PROFILE
- SET STATISTICS TIME
- SET STATISTICS XML
- SHUTDOWN statement
- UPDATE STATISTICS
- UPDATETEXT
- Using EXECUTE to call a SQL function
- VIEW... CHECK OPTION clause

- VIEW... VIEW_METADATA clause
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR, RECEIVE
- WITH XMLNAMESPACES construct
- WRITETEXT
- XPATH expressions

Arbeiten mit Prozeduren von Babelfish für Aurora PostgreSQL

Übersicht

Sie können das folgende Verfahren für DB-Instances von Amazon RDS verwenden, auf denen Babelfish für Aurora PostgreSQL ausgeführt wird, um eine bessere Abfrageleistung zu erzielen:

- sp_babelfish_volatility
- sp_execute_postgresql

sp_babelfish_volatility

Die Volatilität von PostgreSQL-Funktionen hilft dem Optimierer bei der Ausführung von Abfragen. Dies hat bei Verwendung in bestimmten Klauseln erhebliche Auswirkungen auf die Abfrageleistung.

Syntax

```
sp_babelfish_volatility 'function_name', 'volatility'
```

Argumente

function_name (optional)

Sie können den Wert dieses Arguments entweder mit einem zweiteiligen Namen wie `schema_name.function_name` oder nur mit `function_name` angeben. Wenn Sie nur `function_name` angeben, ist der Schemaname das Standardschema für den aktuellen Benutzer.

volatility (optional)

Die gültigen PostgreSQL-Werte für die Volatilität sind `stable`, `volatile` oder `immutable`. Weitere Informationen finden Sie unter https://www.postgresql.org/docs/current/xfunc-volatility.html.

> **Note**
>
> Wenn `sp_babelfish_volatility` mit `function_name` aufgerufen wird und dieser mehrere Definitionen hat, wird ein Fehler ausgegeben.

Ergebnismenge

Wenn die Parameter nicht angegeben werden, wird die Ergebnismenge unter den folgenden Spalten angezeigt: `schemaname`, `functionname`, `volatility`.

Nutzungshinweise

Die Volatilität von PostgreSQL-Funktionen hilft dem Optimierer bei der Ausführung von Abfragen. Dies hat bei Verwendung in bestimmten Klauseln erhebliche Auswirkungen auf die Abfrageleistung.

Beispiele

Die folgenden Beispielen veranschaulichen das Erstellen einfacher Funktionen und die Anwendung von `sp_babelfish_volatility` auf diese Funktionen mit verschiedenen Methoden.

```
1> create function f1() returns int as begin return 0 end
2> go
```

```
1> create schema test_schema
2> go
```

```
1> create function test_schema.f1() returns int as begin return 0 end
2> go
```

Das folgende Beispiel zeigt die Volatilität der Funktionen:

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           volatile
test_schema f1           volatile
```

Das folgende Beispiel zeigt, wie Sie die Volatilität der Funktionen ändern können:

```
1> exec sp_babelfish_volatility 'f1','stable'
2> go
1> exec sp_babelfish_volatility 'test_schema.f1','immutable'
2> go
```

Wenn Sie nur function_name angeben, werden der Schemaname, der Funktionsname und die Volatilität dieser Funktion angezeigt. Das folgende Beispiel zeigt die Volatilität der Funktionen nach Änderung der Werte:

```
1> exec sp_babelfish_volatility 'test_schema.f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
test_schema f1           immutable
```

```
1> exec sp_babelfish_volatility 'f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
```

Wenn Sie kein Argument angeben, wird eine Liste der Funktionen (Schemaname, Funktionsname, Volatilität der Funktionen) angezeigt, die in der aktuellen Datenbank vorhanden sind:

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
test_schema f1           immutable
```

sp_execute_postgresql

Sie können PostgreSQL-Anweisungen vom T-SQL-Endpunkt aus ausführen. Dies vereinfacht Ihre Anwendungen, da Sie den T-SQL-Port nicht verlassen müssen, um diese Anweisungen auszuführen.

Syntax

```
sp_execute_postgresql [ @stmt = ] statement
```

Argumente

[@stmt]-Anweisung

Das Argument hat den Datentyp „varchar“. Dieses Argument akzeptiert Anweisungen im PG-Dialekt.

> **Note**
>
> Sie können nur eine Anweisung im PG-Dialekt als Argument übergeben, andernfalls wird der folgende Fehler ausgelöst.

```
1>exec sp_execute_postgresql 'create extension pg_stat_statements; drop extension
 pg_stat_statements'
2>go
```

```

Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
expected 1 statement but got 2 statements after parsing
```

Nutzungshinweise

CREATE EXTENSION

Erstellt eine neue Erweiterung und lädt diese in die aktuelle Datenbank.

```
1>EXEC sp_execute_postgresql 'create extension [ IF NOT EXISTS ] <extension name>
 [ WITH ] [SCHEMA schema_name] [VERSION version]';
2>go
```

Im folgenden Beispiel sehen Sie, wie eine Erweiterung erstellt wird:

```
1>EXEC sp_execute_postgresql 'create extension pg_stat_statements with schema sys
 version "1.10"';
```

```
2>go
```

Verwenden Sie den folgenden Befehl für den Zugriff auf Erweiterungsobjekte:

```
1>select * from pg_stat_statements;
2>go
```

> **Note**
>
> Wenn der Schemaname bei der Erstellung der Erweiterung nicht explizit angegeben wird, werden die Erweiterungen standardmäßig im öffentlichen Schema installiert. Sie müssen den Schemaqualifizierer angeben, um auf die Erweiterungsobjekte zuzugreifen, wie unten dargestellt:

```
1>select * from [public].pg_stat_statements;
2>go
```

Unterstützte Erweiterungen

Die folgenden mit Aurora PostgreSQL verfügbaren Erweiterungen können mit Babelfish verwendet werden.

- `pg_stat_statements`
- `tds_fdw`
- `fuzzystrmatch`

Einschränkungen

- Benutzer müssen in T-SQL über die Rolle „sysadmin“ und in Postgres über die Rolle „rds_superuser“ verfügen, um die Erweiterungen installieren zu können.
- Es ist nicht möglich, Erweiterungen in vom Benutzer erstellten Schemas sowie in dbo- und Gastschemas für die Masterdatenbank, die Datenbank „tempdb“ und die Datenbank „msdb“ zu installieren.
- Die Option CASCADE wird nicht unterstützt.

ALTER EXTENSION

Mit ALTER EXTENSION können Sie eine Aktualisierung auf eine neue Erweiterungsversion vornehmen.

```
1>EXEC sp_execute_postgresql 'alter extension <extension name> UPDATE TO
 <new_version>';
2>go
```

Einschränkungen

- Sie können die Version Ihrer Erweiterung nur mit der Anweisung ALTER EXTENSION aktualisieren. Andere Operationen werden nicht unterstützt.

DROP EXTENSION

Löscht die angegebene Erweiterung. Sie können auch die Optionen `if exists` oder `restrict` verwenden, um die Erweiterung zu löschen.

```
1>EXEC sp_execute_postgresql 'drop extension <extension name>';
2>go
```

Einschränkungen

- Die Option CASCADE wird nicht unterstützt.

Leistung und Skalierung für Amazon Aurora PostgreSQL

Im folgenden Abschnitt werden die Verwaltung der Performance und Skalierung für einen Amazon-Aurora-PostgreSQL-DB-Cluster erläutert. Es umfasst auch Informationen über andere Wartungsaufgaben.

Themen

- Skalierung von Aurora PostgreSQL-DB-Instances
- Maximale Verbindungen zu einer Aurora PostgreSQL-DB-Instance
- Temporäre Speicherlimits für Aurora PostgreSQL
- Huge Pages für PostgreSQL
- Testen von Amazon Aurora PostgreSQL unter Verwendung von Fehlersimulationsabfragen

- Anzeigen des Volume-Status für einen Aurora PostgreSQL-DB-Cluster
- RAM-Datenträger für das stats_temp_directory
- Verwalten temporärer Dateien mit PostgreSQL

Skalierung von Aurora PostgreSQL-DB-Instances

Sie können Aurora PostgreSQL-DB-Instances auf zwei Arten skalieren, durch Skalierung der Instance oder durch Skalierung der Lesevorgänge. Weitere Informationen zur Lese-Skalierung finden Sie unter Skalierung von Lesevorgängen.

Sie können Ihr Aurora-PostgreSQL-DB-Cluster skalieren, indem Sie die DB-Instance-Klasse für jede DB-Instance im DB-Cluster ändern. Aurora PostgreSQL unterstützt mehrere DB-Instance-Klassen, die für Aurora optimiert sind. Verwenden Sie nicht die Instance-Klasse db.t2 oder db.t3 für größere Aurora-Cluster mit einer Größe von mehr als 40 Terabyte (TB).

> **Note**
>
> Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Einzelheiten zu den T-Instance-Klassen finden Sie unter DB-Instance-Klassenarten.

Die Skalierung erfolgt nicht augenblicklich. Es kann 15 Minuten oder länger dauern, bis die Änderung zu einer anderen DB-Instance-Klasse abgeschlossen ist. Wenn Sie diesen Ansatz zum Ändern der DB-Instance-Klasse verwenden, sollten Sie die Änderung während des nächsten geplanten Wartungsfensters (und nicht sofort) anwenden, um Auswirkungen auf die Benutzer zu vermeiden.

Alternativ zur direkten Änderung der DB-Instance-Klasse können Sie Ausfallzeiten minimieren, indem Sie die Hochverfügbarkeitsfunktionen von Amazon Aurora verwenden. Fügen Sie Ihrem Cluster zuerst ein Aurora-Replikat hinzu. Wählen Sie beim Erstellen des Replikats die DB-Instance-Klassengröße, die Sie für Ihren Cluster verwenden möchten. Wenn das Aurora Replica mit dem Cluster synchronisiert wird, können Sie das neu hinzugefügte Replikat ausfallen. Weitere Informationen hierzu finden Sie unter Aurora-Replikate und Schnelles Failover mit Amazon Aurora PostgreSQL.

Detaillierte Angaben zu den von Aurora PostgreSQL unterstützten DB-Instance-Klassen finden Sie unter Unterstützte DB-Engines für DB-Instance-Klassen.

Maximale Verbindungen zu einer Aurora PostgreSQL-DB-Instance

Ein Aurora-PostgreSQL-DB-Cluster weist Ressourcen basierend auf der DB-Instance-Klasse und ihrem verfügbaren Speicher zu. Jede Verbindung mit dem DB-Cluster verbraucht inkrementelle Mengen dieser Ressourcen wie Speicher und CPU. Der pro Verbindung verbrauchte Speicher variiert je nach Abfragetyp, Anzahl und der Tatsache, ob temporäre Tabellen verwendet werden. Selbst eine inaktive Verbindung verbraucht Speicher und CPU. Das liegt daran, dass bei Abfragen für eine Verbindung mehr Speicher für jede Abfrage zugewiesen wird und der Speicher nicht vollständig freigegeben wird, auch wenn die Verarbeitung beendet wird. Daher empfehlen wir Ihnen, sicherzustellen, dass Ihre Anwendungen keine inaktiven Verbindungen aufrechterhalten: Jede dieser Verbindungen verschwendet Ressourcen und wirkt sich negativ auf die Leistung aus. Weitere Informationen finden Sie unter Ressourcen, die von inaktiven PostgreSQL-Verbindungen verbraucht werden.

Die maximale Anzahl der zulässigen Verbindungen einer Aurora-PostgreSQL-DB-Instance wird durch den Wert des Parameters `max_connections` in der Parametergruppe für diese DB-Instance festgelegt. Der ideale Rahmen für den `max_connections`-Parameter unterstützt alle Client-Verbindungen, die Ihre Anwendung benötigt, ohne einen Überschuss an ungenutzten Verbindungen, sowie mindestens 3 weitere Verbindungen für AWS-Automatisierung. Vor dem Ändern des Parameters `max_connections` sollten Sie Folgendes berücksichtigen:

- Wenn der Wert von `max_connections` zu niedrig ist, verfügt die Aurora-PostgreSQL-DB-Instance möglicherweise nicht über ausreichende Verbindungen, wenn Clients versuchen, eine Verbindung herzustellen. In diesem Fall werden durch Verbindungsversuche mit `psql` Fehlermeldungen wie die folgenden ausgelöst:

  ```
  psql: FATAL: remaining connection slots are reserved for non-replication superuser
   connections
  ```

- Wenn der Wert von `max_connections` die Anzahl der benötigten Verbindungen übersteigt, können die nicht verwendeten Verbindungen die Leistung beeinträchtigen.

Der Standardwert von `max_connections` wird von der folgenden Aurora-PostgreSQL–FunktionLEAST abgeleitet:

`LEAST({DBInstanceClassMemory/9531392},5000)`.

Wenn Sie den Wert für `max_connections` ändern möchten, müssen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe erstellen und dort ihren Wert ändern. Wenn Sie Ihre benutzerdefinierte

DB-Parametergruppe auf Ihren Cluster angewendet haben, müssen Sie die primäre Instance neu starten, damit der neue Wert wirksam wird. Weitere Informationen erhalten Sie unter Amazon-Aurora-PostgreSQL-Parameter und Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora.

> **Tip**
>
> Wenn Ihre Anwendungen häufig Verbindungen öffnen und schließen oder langlebige Verbindungen in großer Zahl offen lassen, empfehlen wir Ihnen, Amazon-RDS-Proxy zu verwenden. RDS-Proxy ist ein vollständig verwalteter, hochverfügbarer Datenbank-Proxy, der Datenbankverbindungen sicher und effizient per Verbindungspooling freigibt. Weitere Informationen zu RDS-Proxy finden Sie unter Amazon-RDS-Proxy für Aurora.

Weitere Informationen dazu, wie Aurora Serverless v2-Instances diesen Parameter behandeln, finden Sie unter Maximale Anzahl der Verbindungen für Aurora Serverless v2.

Temporäre Speicherlimits für Aurora PostgreSQL

Aurora PostgreSQL speichert Tabellen und Indizes im Aurora-Speichersubsystem. Aurora PostgreSQL verwendet separaten temporären Speicher für nicht persistente temporäre Dateien. Dazu gehören Dateien, die für Zwecke wie das Sortieren großer Datensätze während der Abfrageverarbeitung oder für Indexerstellungsvorgänge verwendet werden. Weitere Informationen finden Sie im Artikel Wie kann ich lokale Speicherprobleme in Aurora-PostgreSQL-kompatiblen Instances beheben?.

Diese lokalen Speicher-Volumes werden von Amazon Elastic Block Store gestützt und können durch Einsatz einer größeren DB-Instance-Klasse erweitert werden. Weitere Informationen über Speicher finden Sie unter Amazon-Aurora-Speicher. Sie können Ihren lokalen Speicher für temporäre Objekte auch vergrößern, indem Sie einen Instance-Typ mit NVMe-Unterstützung und temporäre Objekte mit aktivierten Aurora-optimierten Lesevorgängen verwenden. Weitere Informationen finden Sie unter Verbesserung der Abfrageleistung für Aurora PostgreSQL mit Aurora-optimierten Lesevorgängen.

> **Note**
>
> Beim Skalieren von DB-Instances, z. B. von db.r5.2xlarge auf db.r5.4xlarge, treten unter Umständen `storage-optimization`-Ereignisse auf.

Die folgende Tabelle zeigt die maximale Menge an temporärem Speicher, die für jede Aurora PostgreSQL-DB-Instance-Klasse verfügbar ist. Informationen zur Unterstützung der DB-Instance-Klasse für Aurora finden Sie unter DB-Instance-Klassen von Amazon Aurora.

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer Speicher (GiB)
db.x2g.16xlarge	1829
db.x2g.12xlarge	1606
db.x2g.8xlarge	1071
db.x2g.4xlarge	535
db.x2g.2xlarge	268
db.x2g.xlarge	134
db.x2g.large	67
db.r8g.48xlarge	3072
db.r8g.24xlarge	1536
db.r8g.16xlarge	998
db.r8g.12xlarge	749
db.r8g.8xlarge	499
db.r8g.4xlarge	250
db.r8g.2xlarge	125
db.r8g.xlarge	63
db.r8g.large	31
db.r7g.16xlarge	1008
db.r7g.12xlarge	756

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer Speicher (GiB)
db.r7g.8xlarge	504
db.r7g.4xlarge	252
db.r7g.2xlarge	126
db.r7g.xlarge	63
db.r7g.large	32
db.r7i.48xlarge	3072
db.r7i.24xlarge	1500
db.r7i.16xlarge	1008
db.r7i.12xlarge	748
db.r7i.8xlarge	504
db.r7i.4xlarge	249
db.r7i.2xlarge	124
db.r7i.xlarge	62
db.r7i.large	31
db.r6g.16xlarge	1008
db.r6g.12xlarge	756
db.r6g.8xlarge	504
db.r6g.4xlarge	252
db.r6g.2xlarge	126
db.r6g.xlarge	63

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer Speicher (GiB)
db.r6g.large	32
db.r6i.32xlarge	1829
db.r6i.24xlarge	1500
db.r6i.16xlarge	1008
db.r6i.12xlarge	748
db.r6i.8xlarge	504
db.r6i.4xlarge	249
db.r6i.2xlarge	124
db.r6i.xlarge	62
db.r6i.large	31
db.r5.24xlarge	1500
db.r5.16xlarge	1008
db.r5.12xlarge	748
db.r5.8xlarge	504
db.r5.4xlarge	249
db.r5.2xlarge	124
db.r5.xlarge	62
db.r5.large	31
db.r4.16xlarge	960
db.r4.8xlarge	480

DB-Instance-Klasse (DB instance class)	Maximal verfügbarer temporärer Speicher (GiB)
db.r4.4xlarge	240
db.r4.2xlarge	120
db.r4.xlarge	60
db.r4.large	30
db.t4g.large	16.5
db.t4g.medium	8.13
db.t3.large	16
db.t3.medium	7,5

> **Note**
>
> Instance-Typen mit NVMe-Unterstützung können den verfügbaren temporären Speicherplatz um bis zur gesamten NVMe-Größe erhöhen. Weitere Informationen finden Sie unter Verbesserung der Abfrageleistung für Aurora PostgreSQL mit Aurora-optimierten Lesevorgängen.

Sie können den für eine DB-Instance verfügbaren temporären Speicher mit der CloudWatch-Metrik `FreeLocalStorage` überwachen, die unter CloudWatch Amazon-Metriken für Amazon Aurora beschrieben wird. (Dies gilt nicht für Aurora Serverless v2.)

Bei einigen Workloads können Sie die Menge an temporärem Speicher reduzieren, indem Sie den Prozessen, die die Operation ausführen, mehr Arbeitsspeicher zuweisen. Um den für einen Vorgang verfügbaren Speicher zu erhöhen, erhöhen Sie die PostgreSQL-Werte der Parameter work_mem oder maintenance_work_mem.

Huge Pages für PostgreSQL

Huge Pages ist eine Arbeitsspeicher-Verwaltungsfunktion, die den Overhead reduziert, wenn eine DB-Instance mit großen, zusammenhängenden Arbeitsspeicherblöcken arbeitet, wie sie von

gemeinsam genutzten Puffern verwendet werden. Diese PostgreSQL-Funktion wird von allen derzeit verfügbaren Versionen von Aurora PostgreSQL unterstützt.

Der Parameter `Huge_pages` ist standardmäßig für alle DB-Instance-Klassen aktiviert, außer t3.medium, db.t3.large, db.t4g.medium, db.t4g.large. In den unterstützten Instance-Klassen von Aurora PostgreSQL können Sie den Parameterwert `huge_pages` nicht ändern oder diese Funktion deaktivieren.

Auf DB-Instances von Aurora PostgreSQL, die die Arbeitsspeicherfunktion für große Seiten nicht unterstützen, kann die Speichernutzung bestimmter Prozesse zunehmen, ohne dass sich der Workload entsprechend ändert.

Das System weist beim Serverstart gemeinsam genutzte Speichersegmente wie den Puffer-Cache zu. Wenn große Speicherseiten nicht verfügbar sind, belastet das System diese Zuweisungen nicht dem Postmaster-Prozess. Stattdessen bezieht es den Speicher in den Prozess mit ein, der zuerst auf jede 4-KB-Seite im gemeinsamen Speichersegment zugegriffen hat.

> **Note**
>
> Aktive Verbindungen teilen sich den zugewiesenen Speicher nach Bedarf, unabhängig davon, wie die Nutzung des gemeinsamen Speichers prozessübergreifend verfolgt wird.

Testen von Amazon Aurora PostgreSQL unter Verwendung von Fehlersimulationsabfragen

Sie können die Fehlertoleranz Ihres Aurora PostgreSQL-DB-Clusters testen, indem Sie Fehlersimulationsabfragen verwenden. Fehlersimulationsabfragen werden als SQL-Befehle an eine Amazon Aurora-Instance ausgegeben. Mit Fehlersimulationsabfragen können Sie die Instance zum Absturz bringen, sodass Sie Failover und Wiederherstellung testen können. Sie können auch Aurora Replica-Ausfall, Festplattenausfall und Datenträgerüberlastung simulieren. Fehlersimulationsabfragen werden von allen verfügbaren Aurora-PostgreSQL-Versionen wie folgt unterstützt.

- Aurora PostgreSQL 12, 13, 14 und höhere Versionen
- Aurora PostgreSQL 11.7 und höhere Versionen
- Aurora PostgreSQL 10.11 und höhere Versionen

Themen

- Testen eines Instance-Ausfalls
- Testen eines Aurora-Replikatausfalls
- Testen eines Festplattenausfalls
- Testen einer Festplattenüberlastung

Wenn eine Fehlersimulationsabfrage einen Absturz angibt, erzwingt sie einen Absturz der Aurora PostgreSQL-DB-Instance. Die anderen Fehler-Injection-Abfragen erzeugen Simulationen von Ausfallereignissen, lösen aber keine Ereignisse aus. Wenn Sie eine Fehlersimulationsabfrage senden, geben Sie auch einen Zeitraum vor, in dem die Fehlerereignissimulation ablaufen soll.

Sie können eine Fehlersimulationsabfrage an eine Ihrer Aurora Replica-Instances senden, indem Sie eine Verbindung mit dem Endpunkt der Aurora Replica herstellen. Weitere Informationen finden Sie unter Amazon-Aurora-Endpunktverbindungen.

Testen eines Instance-Ausfalls

Sie können den Absturz einer Aurora PostgreSQL-Instance erzwingen, indem Sie die Fehlersimulationsabfrage-Funktion verwende `aurora_inject_crash()`.

Für diese Fehlersimulationsabfrage tritt kein Failover auf. Wenn Sie einen Failover testen möchten, können Sie die Instance-Operation Failover für Ihren DB-Cluster in der RDS-Konsole ausführen oder den AWS CLI-Befehl failover-db-cluster bzw. die RDS-API-Operation FailoverDBCluster verwenden.

Syntax

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

Optionen

Diese Fehlersimulationsabfrage löst einen der folgenden Ausfalltypen aus. Bei der Absturzart wird die Groß-/Kleinschreibung nicht berücksichtigt:

'instance'

Es wird ein Absturz der PostgreSQL-kompatiblen Datenbank für die Amazon Aurora-Instance simuliert.

'Disponent'

Es wird ein Absturz des Dispatchers auf primären Instance für das Aurora-DB-Cluster simuliert. Der Dispatcher schreibt Updates zum Cluster-Volume für ein Amazon Aurora-DB-Cluster.

'Knoten'

Es wird ein Absturz sowohl der PostgreSQL-kompatiblen Datenbank als auch des Dispatchers für die Amazon Aurora-Instance simuliert.

Testen eines Aurora-Replikatausfalls

Sie können den Ausfall eines Aurora-Replikats mittels der Fehlersimulationsabfrage-Funktion simuliere `aurora_inject_replica_failure()`.

EinAurora Replikatfehler blockiert die Replikation auf dasAurora Replikat oder alle Aurora-Replikate im DB-Cluster um den angegebenen Prozentsatz für das angegebene Zeitintervall. Wenn das Zeitintervall abgeschlossen ist, sind die betroffenen Aurora-Replikate automatisch mit der primären Instance synchronisiert.

Syntax

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   time_interval,
   'replica_name'
);
```

Optionen

Diese Fehlersimulationsabfrage verwendet die folgenden Parameter:

prozent_an_ausfällen

Der Prozentsatz der Replikate, die während des Ausfallereignisses blockiert werden sollen. Dieser Wert kann ein Duplikat zwischen 0 und 100 sein. Wenn Sie 0 angeben, wird keine Replikation blockiert. Wenn Sie 100 angeben, wird die gesamte Replikation blockiert.

time_intervall

Die Zeitspanne für die Simulation des Ausfalls des Aurora-Replikats. Das Intervall ist in Sekunden. Wenn der Wert beispielsweise 20 ist, wird die Simulation 20 Sekunden lang ausgeführt.

> **Note**
>
> Seien Sie vorsichtig bei der Angabe des Zeitintervalls für das Ausfallereignis in Ihrem Aurora-Replikat. Wenn Sie ein zu langes Intervall festlegen und Ihre Writer-Instance eine große Datenmenge während des Ausfallereignisses schreibt, könnte Ihr Aurora-DB-Cluster annehmen, dass Ihr Aurora-Replikat ausgefallen ist, und es ersetzen.

replica_name

Das Aurora-Replikat, in das die Ausfallsimulation injiziert werden soll. Geben Sie den Namen eines Aurora-Replikats an, um einen Ausfall des einzelnen Aurora-Replikats zu simulieren. Geben Sie eine leere Zeichenfolge an, um Ausfälle für alle Aurora-Replikate im DB-Cluster zu simulieren.

Informationen zur Identifizierung von Replikatnamen finden Sie in der Spalte `server_id` der Funktion `aurora_replica_status()`. Zum Beispiel:

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

Testen eines Festplattenausfalls

Sie können einen Festplattenausfall für einen Aurora PostgreSQL-DB-Cluster mittels der Fehlersimulationsabfrage-Funktion simuliere `aurora_inject_disk_failure()`.

Während einer Simulation eines Festplattenausfalls markiert der Aurora PostgreSQL-DB-Cluster zufällige Festplattensegmente als fehlerhaft. Anfragen an diese Segmente werden für die Dauer der Simulation blockiert.

Syntax

```
SELECT aurora_inject_disk_failure(
   percentage_of_failure,
   index,
   is_disk,
   time_interval
);
```

Optionen

Diese Fehlersimulationsabfrage verwendet die folgenden Parameter:

prozent_an_ausfällen

Der Prozentsatz des Datenträgers, der während des Ausfallereignisses als ausgefallen markiert werden soll. Dieser Wert kann ein Duplikat zwischen 0 und 100 sein. Wenn Sie 0 festlegen, wird kein Segment der Festplatte als fehlerhaft markiert. Wenn Sie 100 festlegen, wird die gesamte Festplatte als fehlerhaft markiert.

index

Ein spezifischer logischer Datenblock, in dem das Ausfallereignis simuliert werden soll. Wenn Sie den Bereich der verfügbaren logischen Blöcke oder Speichernodedaten überschreiten, erhalten Sie eine Fehlermeldung, in der Ihnen der maximal festlegbare Indexwert angezeigt wird. Informationen zur Vermeidung dieses Fehlers finden Sie unter Anzeigen des Volume-Status für einen Aurora PostgreSQL-DB-Cluster.

ist_festplatte

Gibt an, ob der Injektionsfehler auf einen logischen Block oder einen Speicherknoten zurückzuführen ist. Die Angabe von „true“ bedeutet, dass Injektionsfehler auf einen logischen Block zurückzuführen sind. Die Angabe von „false“ bedeutet, dass Injektionsfehler auf einen Speicherknoten zurückzuführen sind.

time_intervall

Die Zeitspanne für die Simulation des Datenträgerausfalls. Das Intervall ist in Sekunden. Wenn der Wert beispielsweise 20 ist, wird die Simulation 20 Sekunden lang ausgeführt.

Testen einer Festplattenüberlastung

Sie können eine Datenträgerüberlastung für einen DB-Cluster von Aurora PostgreSQL mittels der Funktion zur Fehlerinjektionsabfrage `aurora_inject_disk_congestion()` simulieren.

Während einer Simulation einer Festplattenüberlastung markiert der Aurora PostgreSQL-DB-Cluster zufällige Festplattensegmente als überlastet. Anfragen an diese Segmente werden für die Dauer der angegebenen minimalen und maximalen Verzögerungszeit während der Simulation verzögert.

Syntax

```
SELECT aurora_inject_disk_congestion(
   percentage_of_failure,
   index,
   is_disk,
```

```
    time_interval,
    minimum,
    maximum
);
```

Optionen

Diese Fehlersimulationsabfrage verwendet die folgenden Parameter:

prozent_an_ausfällen

Der Prozentsatz des Datenträgers, der während des Ausfallereignisses als überlastet markiert werden soll. Dies ist ein doppelter Wert zwischen 0 und 100. Wenn Sie 0 festlegen, wird kein Segment der Festplatte als überlastet markiert. Wenn Sie 100 festlegen, wird die gesamte Festplatte als überlastet markiert.

index

Ein spezifischer logischer Datenblock oder Speicherknoten, der zum Simulieren des Ausfallereignisses verwendet werden soll.

Wenn Sie den Bereich der verfügbaren logischen Datenblöcke oder Speichernodes überschreiten, erhalten Sie eine Fehlermeldung, die Ihnen den maximalen Indexwert angibt, den Sie angeben können. Informationen zur Vermeidung dieses Fehlers finden Sie unter Anzeigen des Volume-Status für einen Aurora PostgreSQL-DB-Cluster.

ist_festplatte

Gibt an, ob der Injektionsfehler auf einen logischen Block oder einen Speicherknoten zurückzuführen ist. Die Angabe von „true“ bedeutet, dass Injektionsfehler auf einen logischen Block zurückzuführen sind. Die Angabe von „false“ bedeutet, dass Injektionsfehler auf einen Speicherknoten zurückzuführen sind.

time_intervall

Die Zeitspanne für die Simulation der Datenträgerüberlastung. Das Intervall ist in Sekunden. Wenn der Wert beispielsweise 20 ist, wird die Simulation 20 Sekunden lang ausgeführt.

Minimum, Maximum

Der Mindest- und Höchstwert in Millisekunden für die Überlastungsverzögerung. Gültige Werte reichen von 0,0 bis 100,0 Millisekunden. Festplattensegmente, die als überlastet gekennzeichnet sind, werden für die Dauer der Simulation um eine zufällige Zeit innerhalb des minimalen und maximalen Bereichs verzögert. Der Höchstwert muss größer als der Mindestwert sein.

Anzeigen des Volume-Status für einen Aurora PostgreSQL-DB-Cluster

In Amazon Aurora besteht ein DB-Cluster-Volume aus einer Sammlung von logischen Blöcken. Jeder von ihnen stellt 10 Gigabyte des zugeteilten Arbeitsspeichers bereit. Diese Blöcke werden als Schutzgruppen bezeichnet.

Die Daten in den einzelnen Schutzgruppen werden über sechs physische Speichereinheiten, so genannte Speicherknoten, repliziert. Diese Speicherknoten werden in drei Availability Zones (AZs) in der Region zugeteilt, in der sich das DB-Cluster befindet. Jeder Speicherknoten wiederum besteht aus einem oder mehreren logischen Datenblöcken für das DB-Cluster-Volume. Weitere Informationen zu Schutzgruppen und Speicherknoten finden Sie unter Introducing the Aurora Storage Engine im AWS Database Blog. Weitere Informationen über Aurora-Cluster-Volumes im Allgemeinen finden Sie unter Amazon-Aurora-Speicher.

Verwenden Sie die Funktion `aurora_show_volume_status()`, um die folgenden Serverstatusvariablen zurückzugeben:

- `Disks` – Die Gesamtanzahl der logischen Datenblöcke für das DB-Cluster-Volume.
- `Nodes` — Die Gesamtanzahl der Speicherknoten für das DB-Cluster-Volume.

Sie können die Funktion `aurora_show_volume_status()` verwenden, um einen Fehler zu vermeiden, wenn Sie die Fehlersimulationsfunktion `aurora_inject_disk_failure()` verwenden. Die Fehlersimulationsfunktion `aurora_inject_disk_failure()` simuliert den Ausfall eines gesamten Speicherknotens oder eines einzelnen logischen Datenblocks innerhalb eines Speicherknotens. In der Funktion geben Sie den Indexwert eines spezifischen logischen Datenblocks oder Speicherknotens an. Die Anweisung gibt jedoch einen Fehler zurück, wenn Sie einen Indexwert angeben, der größer ist als die Anzahl der logischen Datenblöcke oder Speicherknoten, die vom DB-Cluster-Volume verwendet werden. Weitere Informationen über Fehlersimulationsabfragen finden Sie unter Testen von Amazon Aurora PostgreSQL unter Verwendung von Fehlersimulationsabfragen.

> **Note**
>
> Die Funktion `aurora_show_volume_status()` steht für Aurora PostgreSQL-Version 10.11 zur Verfügung. Weitere Informationen zu den Aurora PostgreSQL-Versionen erhalten Sie unter Amazon Aurora PostgreSQL Releases und Engine Versionen.

Syntax

```
SELECT * FROM aurora_show_volume_status();
```

Beispiel

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-------+-------
    96 |    45
```

RAM-Datenträger für das stats_temp_directory

Sie können den Aurora-PostgreSQL-Parameter `rds.pg_stat_ramdisk_size` verwenden, um den Systemspeicher anzugeben, der einer RAM-Disk für die Speicherung von PostgreSQL `stats_temp_directory`. Der RAM-Disk-Parameter ist für alle Aurora-PostgreSQL-14-Versionen und niedriger verfügbar.

Bei bestimmten Workloads kann durch die Einstellung dieses Parameters die Leistung verbessert und die I/O-Anforderungen können gesenkt werden. Weitere Informationen über `stats_temp_directory` finden Sie unter Laufzeitstatistik in der PostgreSQL-Dokumentation. Ab PostgreSQL Version 15 hat die PostgreSQL-Community auf die Verwendung von dynamischem gemeinsam genutztem Speicher Memory umgestellt. Eine Einstellung von `stats_temp_directory` ist also nicht erforderlich.

Um einen RAM-Datenträger für Ihr `stats_temp_directory` zu aktivieren, legen Sie den Parameter `rds.pg_stat_ramdisk_size` in der von Ihrer DB-Cluster verwendeten Parametergruppe auf einen Wert ungleich Null fest. Dieser Parameter wird in MB angegeben, daher müssen Sie einen ganzzahligen Wert verwenden. Ausdrücke, Formeln und Funktionen sind für den Parameter `rds.pg_stat_ramdisk_size` nicht gültig. Stellen Sie sicher, dass Sie den DB-Cluster neu starten, damit die Änderungen wirksam werden. Weitere Informationen zum Festlegen von Parametern finden Sie unter Parametergruppen für Amazon Aurora. Weitere Informationen zum Neustart einer Verbindung mit dem DB-Cluster finden Sie unter Neustart eines Amazon Aurora DB-Clusters oder einer Amazon Aurora DB-Instance.

Mit dem folgenden AWS CLI-Befehl wird beispielsweise der RAM-Datenträgerparameter auf 256 MB festgelegt.

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
```

```
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

Nachdem Sie den DB-Cluster neu gestartet haben, führen Sie den folgenden Befehl aus, um den Status des 1:`stats_temp_directory`

```
postgres=> SHOW stats_temp_directory;
```

Der Befehl sollte Folgendes zurückgeben:

```
stats_temp_directory
----------------------------
/rdsdbramdisk/pg_stat_tmp
(1 row)
```

Verwalten temporärer Dateien mit PostgreSQL

In PostgreSQL kann eine komplexe Abfrage möglicherweise mehrere Sortier- und Hash-Vorgänge gleichzeitig durchführen, wobei jeder Vorgang Instance-Speicher verwendet, um Ergebnisse bis zu dem im `work_mem` angegebenen Wert zu speichern. Wenn der Instance-Speicher nicht ausreicht, werden temporäre Dateien erstellt, um die Ergebnisse zu speichern. Diese werden auf die Festplatte geschrieben, um die Abfrageausführung abzuschließen. Später werden diese Dateien automatisch entfernt, nachdem die Abfrage abgeschlossen ist. In In Aurora PostgreSQL teilen sich diese Dateien den lokalen Speicher mit anderen Protokolldateien. Sie können den lokalen Speicherplatz Ihres DB-Clusters von Aurora PostgreSQL überwachen, indem Sie sich die Amazon-CloudWatch-Metrik für `FreeLocalStorage` ansehen. Weitere Informationen finden Sie unter Behebung lokaler Speicherprobleme.

Wir empfehlen die Verwendung von Clustern für Aurora-optimierte Lesevorgänge für Workloads mit mehreren gleichzeitigen Abfragen, die mehr temporäre Dateien nutzen. Diese Cluster verwenden lokalen auf Non-Volatile Memory Express (NVMe) basierenden Solid-State-Drive (SSD)-Speicher auf Blockebene, um die temporären Dateien zu speichern. Weitere Informationen finden Sie unter Verbesserung der Abfrageleistung für Aurora PostgreSQL mit Aurora-optimierten Lesevorgängen.

Sie können die folgenden Parameter und Funktionen verwenden, um die temporären Dateien in Ihrer Instance zu verwalten.

- **`temp_file_limit`** – Dieser Parameter bricht jede Abfrage ab, die die Größe von temp_files in KB überschreitet. Dieses Limit verhindert, dass Abfragen endlos ausgeführt werden und Speicherplatz

mit temporären Dateien belegen. Sie können den Wert anhand der Ergebnisse des Parameters `log_temp_files` schätzen. Es hat sich bewährt, das Workload-Verhalten zu untersuchen und das Limit der Schätzung entsprechend festzulegen. Das folgende Beispiel zeigt, wie eine Abfrage abgebrochen wird, wenn sie das Limit überschreitet.

```
postgres=>select * from pgbench_accounts, pg_class, big_table;
```

```
ERROR: temporary file size exceeds temp_file_limit (64kB)
```

- **log_temp_files** – Dieser Parameter sendet Nachrichten an die Datei postgresql.log, wenn die temporären Dateien einer Sitzung entfernt werden. Dieser Parameter erstellt Protokolle, nachdem eine Abfrage erfolgreich abgeschlossen wurde. Daher ist er bei der Fehlerbehebung aktiver, lang andauernder Abfragen möglicherweise nicht hilfreich.

  Das folgende Beispiel zeigt, dass nach erfolgreichem Abschluss der Abfrage die Einträge in der Datei postgresql.log protokolliert werden, während die temporären Dateien bereinigt werden.

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** – Diese Funktion, die von RDS für PostgreSQL 13 und höher verfügbar ist, bietet Einblick in die aktuelle Nutzung temporärer Dateien. Die abgeschlossene Abfrage erscheint nicht in den Ergebnissen der Funktion. Im folgenden Beispiel können Sie sich die Ergebnisse dieser Funktion ansehen.

```
postgres=>select * from pg_ls_tmpdir();
```

```
      name       |    size    |      modification
-----------------+------------+------------------------
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=>select query from pg_stat_activity where pid = 8355;

query
-------------------------------------------------------------------------------------------
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

Der Dateiname enthält die Verarbeitungs-ID (PID) der Sitzung, die die temporäre Datei generiert hat. Eine komplexere Abfrage, wie im folgenden Beispiel, führt eine Summenberechnung der temporären Dateien für jede PID durch.

```
postgres=>select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```
 pid  | count |   sum
------+-------+------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
(4 rows)
```

- **pg_stat_statements** – Wenn Sie den Parameter pg_stat_statements aktivieren, können Sie die durchschnittliche Nutzung temporärer Dateien pro Aufruf einsehen. Sie können die query_id der

Abfrage identifizieren und verwenden, um die Nutzung temporärer Dateien zu untersuchen, wie im folgenden Beispiel gezeigt.

```
postgres=>select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
(1 row)
```

```
postgres=>select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+---------------------------
 -7170349228837045701 | select a.aid from pgbench |    50 |                  239226 |
                     388678
(1 row)
```

- **Performance Insights** – Im Performance-Insights-Dashboard können Sie die Nutzung temporärer Dateien einsehen, indem Sie die Metriken temp_bytes und temp_files aktivieren. Anschließend können Sie den Durchschnitt dieser beiden Metriken sehen und feststellen, wie sie dem Abfrage-Workload entsprechen. In der Ansicht in Performance Insights werden nicht speziell die Abfragen angezeigt, die die temporären Dateien generieren. Wenn Sie jedoch Performance Insights mit der für `pg_ls_tmpdir` angezeigten Abfrage kombinieren, können Sie Fehler in Ihrem Abfrage-Workload beheben, analysieren und die Änderungen ermitteln.

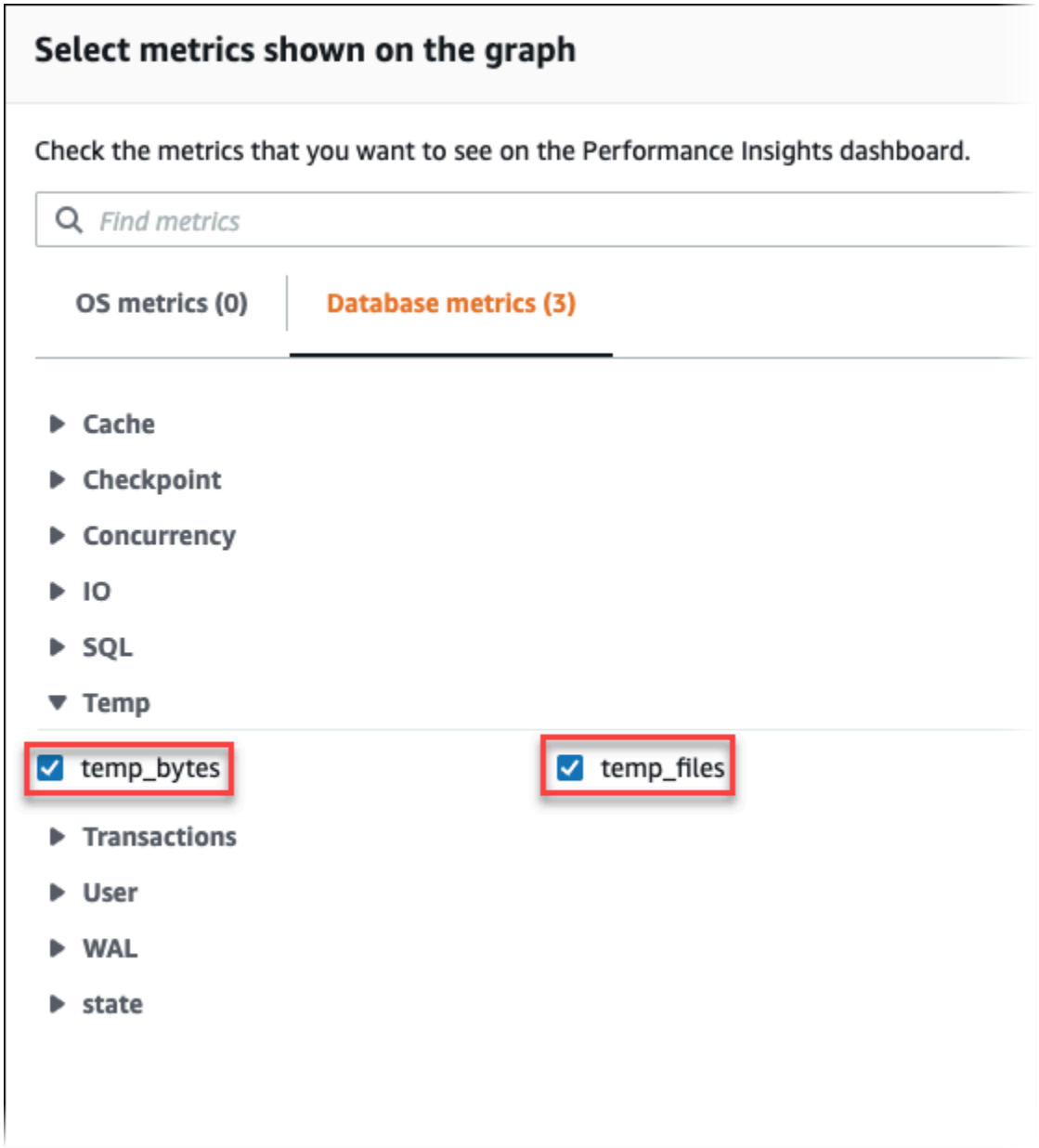
  Weitere Informationen zur Analyse von Metriken und Abfragen mit Performance Insights finden Sie unter Analyse der Metriken mit dem Performance-Insights-Dashboard.

Ein Beispiel für die Anzeige der Nutzung temporärer Dateien mit Performance Insights finden Sie unter Anzeigen der Nutzung temporärer Dateien mit Performance Insights.

Anzeigen der Nutzung temporärer Dateien mit Performance Insights

Sie können Performance Insights verwenden, um die Nutzung temporärer Dateien anzuzeigen, indem Sie die Metriken temp_bytes und temp_files aktivieren. Die Ansicht in Performance Insights zeigt nicht die spezifischen Abfragen an, die temporäre Dateien generieren. Wenn Sie jedoch Performance Insights mit der für `pg_ls_tmpdir` angezeigten Abfrage kombinieren, können Sie Fehler beheben, analysieren und die Änderungen in Ihrem Abfrage-Workload ermitteln.

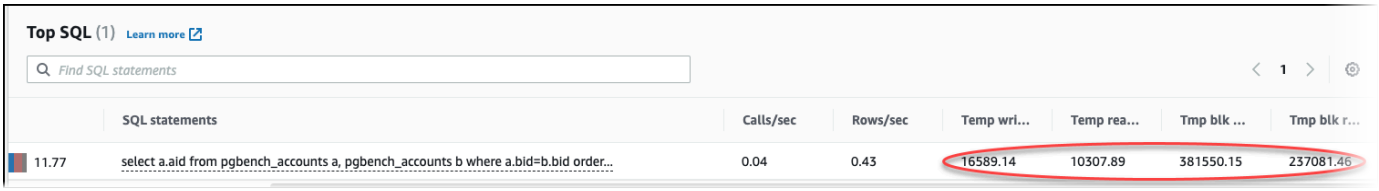
1. Wählen Sie im Performance-Insights-Dashboard Metriken verwalten aus.
2. Wählen Sie Datenbankmetriken und die Metriken temp_bytes und temp_files aus, wie im folgenden Screenshot gezeigt.



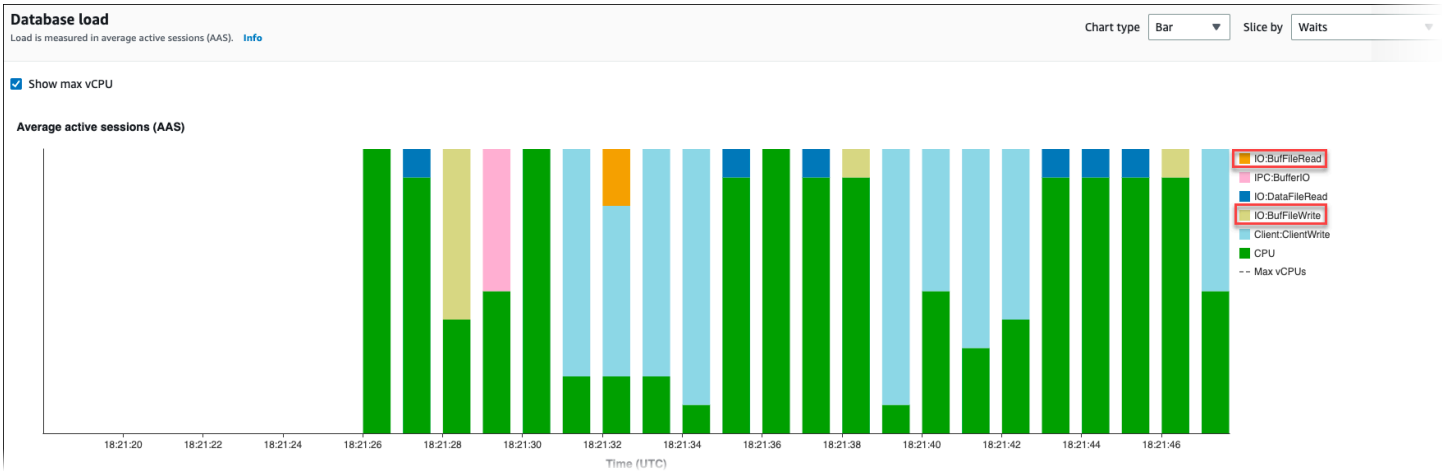
3. Wählen Sie auf der Registerkarte Top SQL das Symbol Einstellungen aus.
4. Schalten Sie im Fenster Einstellungen die folgenden Statistiken ein, damit sie auf der Registerkarte Top SQL angezeigt werden, und wählen Sie Weiter aus.

   - Temporäre Schreibvorgänge pro Sekunde
   - Temporäre Lesevorgänge pro Sekunde
   - Temporäre Massenschreibvorgänge/Aufruf
   - Temporäre Massenlesevorgänge/Aufruf

5. Die temporäre Datei wird aufgegliedert, wenn sie mit der für `pg_ls_tmpdir` gezeigten Abfrage kombiniert wird, wie im folgenden Beispiel gezeigt.



Die `IO:BufFileRead`- und `IO:BufFileWrite`-Ereignisse treten auf, wenn die häufigsten Abfragen in Ihrem Workload häufig temporäre Dateien erstellen. Mit Performance Insights können Sie die häufigsten Abfragen identifizieren, die auf `IO:BufFileRead` und `IO:BufFileWrite` warten, indem Sie die Abschnitte Durchschnittliche aktive Vorträge (AAS) in Datenbanklast und Top SQL überprüfen.



Weitere Informationen zur Analyse von Top-Abfragen und Last nach Warteereignis mit Performance Insights finden Sie unter Überblick über die Registerkarte „Top SQL“. Sie sollten die Abfragen identifizieren und optimieren, die zu einer erhöhten Nutzung temporärer Dateien und damit verbundenen Warteereignissen führen. Weitere Informationen zu diesen Warteereignissen und deren Behebung finden Sie unter IO:BufFileRead und IO:BufFileWrite.

> **Note**
>
> Der Parameter `work_mem` steuert, wann der Sortiervorgang nicht mehr genügend Speicherplatz hat und die Ergebnisse in temporäre Dateien geschrieben werden. Wir empfehlen, die Einstellung dieses Parameters nicht höher als auf den Standardwert festzulegen, da dadurch jede Datenbanksitzung mehr Speicher belegen würde. Außerdem kann eine einzelne Sitzung, die komplexe Verknüpfungen und Sortierungen durchführt, parallele Operationen ausführen, bei denen jeder Vorgang Speicherplatz belegt.

Wenn Sie einen umfangreichen Bericht mit mehreren Verknüpfungen und Sortierungen haben, empfiehlt es sich, diesen Parameter mit dem Befehl `SET work_mem` auf Sitzungsebene festzulegen. Dann wird die Änderung nur auf die aktuelle Sitzung angewendet und der Wert nicht global geändert.

Optimierung mit Warteereignissen für Aurora PostgreSQL

Warteereignisse sind ein wichtiges Optimierungs-Tool für Aurora PostgreSQL. Wenn Sie herausfinden können, warum Sitzungen auf Ressourcen warten und was sie tun, können Sie Engpässe besser reduzieren. Anhand der Informationen in diesem Abschnitt können Sie mögliche Ursachen und Abhilfemaßnahmen ermitteln. Bevor Sie sich mit diesem Abschnitt befassen, empfehlen wir Ihnen dringend, die grundlegenden Aurora-Konzepte zu verstehen, insbesondere die folgenden Themen:

- Amazon-Aurora-Speicher
- Verwalten von Performance und Skalierung für einen Aurora-DB-Cluster

Important

Die Warteereignisse in diesem Abschnitt gelten speziell für Aurora PostgreSQL. Verwenden Sie die Informationen in diesem Abschnitt, um nur Amazon Aurora zu optimieren, nicht RDS für PostgreSQL.
Einige Warteereignisse in diesem Abschnitt haben keine Entsprechungen in den Open-Source-Versionen dieser Datenbank-Engines. Andere Warteereignisse haben dieselben Namen wie Ereignisse in Open-Source-Engines, verhalten sich jedoch anders. Beispielsweise funktioniert Amazon-Aurora-Speicher anders als Open-Source-Speicher, sodass speicherbezogene Warteereignisse auf unterschiedliche Ressourcenbedingungen hinweisen.

Themen

- Grundlegende Konzepte für Aurora PostgreSQL-Optimierung
- Aurora PostgreSQL-Warteereignisse
- Client:ClientRead
- Client:ClientWrite

- CPU
- IO: BufFileRead und IO: BufFileWrite
- IO:DataFileRead
- IO:XactSync
- IPC:DamRecordTxAck
- IPC:parallel-Warteereignisse
- IPC: ProcArrayGroupUpdate
- Lock:advisory
- Lock:extend
- Lock:Relation
- Lock:transactionid
- Lock:tuple
- LWLock:buffer_content (BufferContent)
- LWLock:buffer_mapping
- LWLock:BufferIO (IPC:BufferIO)
- LWLock:lock_manager
- LWLock:MultiXact
- LWLock:pg_stat_statements
- Timeout:PgSleep

Grundlegende Konzepte für Aurora PostgreSQL-Optimierung

Bevor Sie Ihre Aurora PostgreSQL-Datenbank optimieren, sollten Sie wissen, was Warteereignisse sind und warum sie auftreten. Sehen Sie sich auch die grundlegende Speicher- und Festplattenarchitektur von Aurora PostgreSQL an. Ein hilfreiches Architekturdiagramm finden Sie im PostgreSQL-Wikibook.

Themen

- Aurora PostgreSQL-Warteereignisse
- Aurora PostgreSQL-Speicher
- Aurora PostgreSQL-Prozesse

Aurora PostgreSQL-Warteereignisse

Ein Warteereignis zeigt eine Ressource an, auf die eine Sitzung wartet. Das Warteereignis `Client:ClientRead` tritt beispielsweise ein, wenn Aurora PostgreSQL darauf wartet, Daten von der client.emory- und Festplattenarchitektur von Aurora PostgreSQL zu empfangen. Zu den typischen Ressourcen, auf die eine Sitzung wartet, gehören die folgenden:

- Singlethread-Zugriff auf einen Puffer, beispielsweise wenn eine Sitzung versucht, einen Puffer zu ändern
- Eine Zeile, die derzeit von einer anderen Sitzung gesperrt ist
- Eine gelesene Datendatei
- Eine geschriebene Protokolldatei

Um beispielsweise eine Abfrage zu erfüllen, kann die Sitzung einen vollständigen Tabellenscan durchführen. Wenn sich die Daten noch nicht im Arbeitsspeicher befinden, wartet die Sitzung, bis die Datenträger-I/O abgeschlossen ist. Wenn die Puffer in den Speicher gelesen werden, muss die Sitzung möglicherweise warten, da andere Sitzungen auf dieselben Puffer zugreifen. Die Datenbank zeichnet die Wartezeiten unter Verwendung eines vordefinierten Warteereignisses auf. Diese Ereignisse sind in Kategorien eingeteilt.

Ein Wait-Ereignis allein zeigt kein Leistungsproblem an. Wenn sich beispielsweise die angeforderten Daten nicht im Speicher befinden, müssen die Daten von der Festplatte gelesen werden. Wenn eine Sitzung eine Zeile für eine Aktualisierung sperrt, wartet eine andere Sitzung darauf, dass die Zeile entsperrt wird, damit sie sie aktualisieren kann. Bei einem Commit muss gewartet werden, bis der Schreibvorgang in eine Protokolldatei abgeschlossen ist. Wartezeiten sind ein wesentlicher Bestandteil des normalen Funktionierens einer Datenbank.

Eine große Anzahl von Warteereignissen weist normalerweise auf ein Leistungsproblem hin. In solchen Fällen können Sie Warteereignisdaten verwenden, um zu bestimmen, wo die Sitzungen Zeit verbringen. Wenn beispielsweise ein Bericht, der normalerweise in Minuten ausgeführt wird, jetzt stundenlang ausgeführt wird, können Sie die Warteereignisse identifizieren, die am meisten zur Gesamtwartezeit beitragen. Wenn Sie die Ursachen für die häufigsten Warteereignisse ermitteln können, können Sie manchmal Änderungen vornehmen, die die Leistung verbessern. Wenn Ihre Sitzung beispielsweise auf eine Zeile wartet, die von einer anderen Sitzung gesperrt wurde, können Sie die Sperrsitzung beenden.

Aurora PostgreSQL-Speicher

Aurora PostgreSQL-Speicher ist in freigegeben und lokal unterteilt.

Themen

- Gemeinsamer Speicher in Aurora PostgreSQL
- Lokaler Speicher in Aurora PostgreSQL

Gemeinsamer Speicher in Aurora PostgreSQL

Aurora PostgreSQL weist beim Start der Instance gemeinsam genutzten Speicher zu. Shared Memory ist in mehrere Teilbereiche unterteilt. Nachfolgend finden Sie eine Beschreibung der wichtigsten.

Themen

- Freigegebene Puffer
- Write-Ahead-Protokoll (WAL)-Puffer

Freigegebene Puffer

Der freigegebene Pufferpool ist ein Aurora PostgreSQL-Speicherbereich, der alle Seiten enthält, die von Anwendungsverbindungen verwendet werden oder verwendet wurden. Eine Seite ist die Speicherversion eines Plattenblocks. Der gemeinsam genutzte Pufferpool zwischenspeichert die von der Platte gelesenen Datenblöcke. Der Pool reduziert die Notwendigkeit, Daten erneut von der Festplatte zu lesen, wodurch die Datenbank effizienter arbeitet.

Jede Tabelle und jeder Index wird als Array von Seiten einer festen Größe gespeichert. Jeder Block enthält mehrere Tupel, die Zeilen entsprechen. Ein Tupel kann auf jeglicher Seite gespeichert werden.

Der gemeinsam genutzte Pufferpool hat endlichen Speicher. Wenn eine neue Anforderung eine Seite erfordert, die sich nicht im Speicher befindet und kein Speicher mehr vorhanden ist, entfernt Aurora PostgreSQL eine weniger häufig verwendete Seite, um die Anforderung aufzunehmen. Die Räumungsrichtlinie wird durch einen Takt-Sweep-Algorithmus implementiert.

Der Parameter `shared_buffers` bestimmt, wie viel Speicher der Server für das Caching von Daten bereitstellt.

Write-Ahead-Protokoll (WAL)-Puffer

Ein Write-Ahead-Protokoll(WAL)-Puffer enthält Transaktionsdaten, die Aurora PostgreSQL später in den persistenten Speicher schreibt. Mithilfe des WAL-Mechanismus kann Aurora PostgreSQL Folgendes tun:

- Daten nach einem Fehler wiederherstellen
- Reduzieren Sie die Festplatten–I/O indem Sie häufige Schreibvorgänge auf die Festplatte vermeiden

Wenn ein Client Daten ändert, schreibt Aurora PostgreSQL die Änderungen in den WAL-Puffer. Wenn der Client einen `COMMIT` ausgibt, schreibt der WAL-Writerprozess Transaktionsdaten in die WAL-Datei.

Der Parameter `wal_level` bestimmt, wie viele Informationen in das WAL geschrieben werden.

Lokaler Speicher in Aurora PostgreSQL

Jeder Backend-Prozess weist lokalen Speicher für die Abfrageverarbeitung zu.

Themen

- Arbeitsspeicherbereich
- Wartungs-Arbeitsspeicherbereich
- Temporärer Pufferbereich

Arbeitsspeicherbereich

Der Arbeitsspeicherbereich enthält temporäre Daten für Abfragen, die Sortierungen und Hashes durchführen. Beispielsweise führt eine Abfrage mit einer `ORDER BY`-Klausel eine Sortierung durch. Abfragen verwenden Hash-Tabellen in Hash-Joins und Aggregationen.

Der `work_mem`-Parameter die Speichermenge, die von internen Sortiervorgängen und Hash-Tabellen verwendet werden soll, bevor in temporäre Plattendateien geschrieben wird. Der Standardwert lautet 4 MB. Mehrere Sitzungen können gleichzeitig ausgeführt werden, und jede Sitzung kann Wartungsvorgänge parallel ausführen. Aus diesem Grund kann der gesamte verwendete Arbeitsspeicher ein Vielfaches der Einstellung `work_mem` betragen.

Wartungs-Arbeitsspeicherbereich

Der Wartungsarbeitsspeicherbereich speichert Daten für Wartungsvorgänge zwischen. Zu diesen Vorgängen gehören das Vakuumieren, das Erstellen eines Index und das Hinzufügen von Fremdschlüsseln.

Der Parameter `maintenance_work_mem` gibt die maximale Speichermenge an, die von Wartungsvorgängen verwendet werden soll. Der Standardwert lautet 64 MB. In einer Datenbanksitzung kann jeweils nur ein Wartungsvorgang ausgeführt werden.

Temporärer Pufferbereich

Der temporäre Pufferbereich speichert temporäre Tabellen für jede Datenbanksitzung zwischen.

Jede Sitzung weist temporäre Puffer nach Bedarf bis zu dem von Ihnen angegebenen Limit zu. Wenn die Sitzung endet, löscht der Server die Puffer.

Der Parameter `temp_buffers` legt die maximale Anzahl temporärer Puffer fest, die von jeder Sitzung verwendet werden. Vor der ersten Verwendung temporärer Tabellen innerhalb einer Sitzung können Sie den `temp_buffers`-Wert ändern.

Aurora PostgreSQL-Prozesse

Aurora PostgreSQL verwendet mehrere Prozesse.

Themen

- Postmaster-Prozess
- Backend-Prozesse
- Hintergrundprozesse

Postmaster-Prozess

Der Postmaster-Prozess ist der erste Prozess, der beim Starten von Aurora PostgreSQL gestartet wird. Der Postmaster-Prozess hat die folgenden Hauptaufgaben:

- Hintergrundprozesse teilen und überwachen
- Empfangen Sie Authentifizierungsanfragen von Clientprozessen und authentifizieren Sie sie, bevor Sie der Datenbank erlauben, Anfragen zu bearbeiten

Backend-Prozesse

Wenn der Postmaster eine Client-Anfrage authentifiziert, forkisiert der Postmaster einen neuen Backend-Prozess, auch Postgres-Prozess genannt. Ein Client-Prozess verbindet sich mit genau einem Backend-Prozess. Der Client-Prozess und der Backend-Prozess kommunizieren direkt ohne Eingriff des Postmaster-Prozesses.

Hintergrundprozesse

Der Postmaster-Prozess teilt mehrere Prozesse, die unterschiedliche Backend-Aufgaben ausführen. Einige der wichtigeren sind die folgenden:

- WAL-Writer

  Aurora PostgreSQL schreibt Daten im WAL-Puffer (Write Ahead Logging) in die Protokolldateien. Das Prinzip der Write-Ahead-Protokollierung besteht darin, dass die Datenbank keine Änderungen in die Datendateien schreiben kann, bis die Datenbank Protokolldatensätze geschrieben hat, die diese Änderungen auf die Festplatte beschreiben. Der WAL-Mechanismus reduziert Festplatten-I/O und ermöglicht Aurora PostgreSQL, die Protokolle zu verwenden, um die Datenbank nach einem Fehler wiederherzustellen.
- Hintergrund-Autor

  Dieser Prozess schreibt regelmäßig schmutzige (modifizierte) Seiten aus den Speicherpuffern in die Datendateien. Eine Seite wird schmutzig, wenn ein Backend-Prozess sie im Speicher ändert.
- Autovacuum-Daemon

  Der Daemon besteht aus Folgendem:

  - Der Autovacuum-Launcher
  - Die Autovacuum-Worker-Prozesse

  Wenn Autovacuum aktiviert ist, sucht es nach Tabellen mit einer großen Anzahl eingefügter, aktualisierter oder gelöschter Tupel. Der Daemon hat folgende Aufgaben:

  - Wiederherstellen oder Wiederverwenden von Speicherplatz, der von aktualisierten oder gelöschten Zeilen belegt ist
  - Vom Planer verwendete Statistiken aktualisieren
  - Schutz vor Verlust alter Daten durch Transaktions-ID-Wraparound

Die Autovacuum-Funktion automatisiert die Ausführung von `VACUUM`- und `ANALYZE`-Befehlen. `VACUUM` hat folgende Varianten: Standard und Voll. Standardvakuum läuft parallel zu anderen Datenbankvorgängen. `VACUUM FULL` erfordert eine exklusive Sperre für die Tabelle, an der es arbeitet. Daher kann es nicht parallel zu Vorgänge ausgeführt werden, die auf dieselbe Tabelle zugreifen. `VACUUM` erzeugt eine beträchtliche Menge an I/O-Datenverkehr, was zu einer schlechten Leistung anderer aktiver Sitzungen führen kann.

Aurora PostgreSQL-Warteereignisse

Die folgende Tabelle listet die Warteereignisse für Aurora PostgreSQL auf, die am häufigsten auf Leistungsprobleme hinweisen, und fasst die häufigsten Ursachen und Korrekturmaßnahmen zusammen. Die folgenden Warteereignisse sind eine Teilmenge der Liste in Amazon-Aurora-PostgreSQL-Warteereignisse.

Warteereignis	Definition
Client:ClientRead	Dieses Ereignis tritt ein, wenn Aurora PostgreSQL darauf wartet, Daten vom Client zu empfangen.
Client:ClientWrite	Dieses Ereignis tritt ein, wenn Aurora PostgreSQL darauf wartet, Daten an den Client zu schreiben.
CPU	Dieses Ereignis tritt auf, wenn ein Thread in der CPU aktiv ist oder auf die CPU wartet.
IO: BufFileRead und IO: BufFileWrite	Diese Ereignisse treten auf, wenn Aurora PostgreSQL temporäre Dateien erstellt.
IO:DataFileRead	Dieses Ereignis tritt auf, wenn eine Verbindung darauf wartet, dass ein Backend-Prozess eine erforderliche Seite aus dem Speicher liest, da die Seite nicht im gemeinsam genutzten Speicher verfügbar ist.
IO:XactSync	Dieses Ereignis tritt ein, wenn die Datenbank darauf wartet, dass das Aurora-Storage-Subsystem den Commit einer regulären Transaktion bzw. den

Warteereignis	Definition
	Commit oder Rollback einer vorbereiteten Transakti on ausführt.
IPC:DamRecordTxAck	Dieses Ereignis tritt auf, wenn Aurora PostgreSQL in einer Sitzung, die Datenbank-Aktivitätsstreams verwendet, ein Aktivitätsstream-Ereignis erzeugt und dann wartet, bis dieses Ereignis dauerhaft wird
Lock:advisory	Dieses Ereignis tritt auf, wenn eine PostgreSQL-Anwendung eine Sperre verwendet, um Aktivitäten über mehrere Sitzungen hinweg zu koordinieren.
Lock:extend	Dieses Ereignis tritt ein, wenn ein Backend-Prozess darauf wartet, eine Beziehung zu sperren, um sie zu erweitern, während ein anderer Prozess diese Beziehung für denselben Zweck gesperrt hat.
Lock:Relation	Dieses Ereignis tritt ein, wenn eine Abfrage darauf wartet, eine Sperre für eine Tabelle oder Ansicht zu erhalten, die derzeit von einer anderen Transaktion gesperrt ist.
Lock:transactionid	Dieses Ereignis tritt ein, wenn eine Transaktion auf eine Sperre auf Zeilenebene wartet.
Lock:tuple	Dieses Ereignis tritt ein, wenn ein Backend-P rozess darauf wartet, eine Sperre für ein Tupel zu erlangen.
LWLock:buffer_content (BufferContent)	Dieses Ereignis tritt ein, wenn eine Sitzung darauf wartet, eine Datenseite im Speicher zu lesen oder zu schreiben, während eine andere Sitzung diese Seite zum Schreiben gesperrt hat.
LWLock:buffer_mapping	Dieses Ereignis tritt ein, wenn eine Sitzung darauf wartet, einen Datenblock einem Puffer im gemeinsam genutzten Pufferpool zuzuordnen.

Warteereignis	Definition
LWLock:BufferIO (IPC:BufferIO)	Dieses Ereignis tritt auf, wenn Aurora PostgreSQL oder RDS for PostgreSQL darauf wartet, dass andere Prozesse ihre Eingabe-/Ausgabe-(I/O)-Vorgänge beenden, wenn sie gleichzeitig versuchen, auf eine Seite zuzugreifen.
LWLock:lock_manager	Dieses Ereignis tritt ein, wenn die Aurora PostgreSQL-Engine den Speicherbereich der gemeinsam genutzten Sperre verwaltet, um eine Sperre zuzuweisen, zu überprüfen und aufzuheben, wenn eine Fast-Path-Sperre nicht möglich ist.
LWLock:MultiXact	Diese Art von Ereignis tritt auf, wenn Aurora PostgreSQL eine Sitzung offen hält, um mehrere Transaktionen abzuschließen, die dieselbe Zeile in einer Tabelle betreffen. Das Warteereignis gibt an, welcher Aspekt der Verarbeitung mehrerer Transaktionen das Warteereignis generiert, d. h. LWLock:MultiXactOffsetSLRU, LWLock:MultiXactOffsetBuffer, LWLock:MultiXactMemberSLRU oder LWLock:MultiXactMemberBuffer.
Timeout:PgSleep	Dieses Ereignis tritt ein, wenn ein Serverprozess die Funktion `pg_sleep` aufgerufen hat und darauf wartet, dass das Sleep-Timeout abläuft.

Client:ClientRead

Das Ereignis `Client:ClientRead` tritt ein, wenn Aurora PostgreSQL darauf wartet, Daten vom Client zu empfangen.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten

- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für Aurora PostgreSQL Version 10 und höher unterstützt.

Kontext

Ein Aurora PostgreSQL DB-Cluster wartet darauf, Daten vom Client zu empfangen. Der Aurora PostgreSQL-DB-Cluster muss die Daten vom Client empfangen, bevor er weitere Daten an den Client senden kann. Die Zeit, die der Cluster wartet, bevor er Daten vom Client empfängt, ist ein `Client:ClientRead`-Ereignis.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Häufige Gründe dafür, dass das `Client:ClientRead`-Ereignis in den Top-Wartezeiten angezeigt wird, sind die folgenden:

Erhöhte Netzwerklatenz

Es kann zu einer erhöhten Netzwerklatenz zwischen dem Aurora PostgreSQL-DB-Cluster und dem Client kommen. Eine höhere Netzwerklatenz erhöht die Zeit, die der DB-Cluster benötigt, um Daten vom Client zu empfangen.

Erhöhte Belastung des Clients

Auf dem Client kann es zu CPU-Druck oder Netzwerksättigung kommen. Eine Zunahme der Last auf dem Client kann die Übertragung von Daten vom Client zum Aurora PostgreSQL-DB-Cluster verzögern.

Übermäßige Netzwerkrundfahrten

Eine große Anzahl von Netzwerk-Roundtrips zwischen dem Aurora PostgreSQL-DB-Cluster und dem Client kann die Datenübertragung vom Client zum Aurora PostgreSQL-DB-Cluster verzögern.

Großer Kopiervorgang

Während eines Kopiervorgangs werden die Daten vom Dateisystem des Clients in den Aurora PostgreSQL DB-Cluster übertragen. Das Senden einer großen Datenmenge an den DB-Cluster kann die Übertragung von Daten vom Client zum DB-Cluster verzögern.

Verbindung von Clients im Leerlauf

Eine Verbindung mit einer DB-Instance von Aurora PostgreSQL befindet sich im Transaktionsstatus im Leerlauf und wartet darauf, dass ein Client weitere Daten sendet oder einen Befehl ausgibt. Dieser Status kann zu einer Zunahme von `Client:ClientRead`-Ereignissen führen.

pgBouncer wird für das Verbindungspooling verwendet

PgBouncer verfügt über eine Low-Level-Netzwerkkonfigurationseinstellung namens `pkt_buf`, die standardmäßig auf 4.096 eingestellt ist. Wenn die Workload Abfragepakete mit mehr als 4.096 Byte über PgBouncer sendet, empfehlen wir, die `pkt_buf`-Einstellung auf 8.192 zu erhöhen. Wenn die neue Einstellung die Anzahl der `Client:ClientRead`-Ereignisse nicht verringert, empfehlen wir, die `pkt_buf`-Einstellung auf höhere Werte zu erhöhen, z. B. 16.384 oder 32.768. Wenn der Abfragetext groß ist, kann die größere Einstellung besonders hilfreich sein.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Platzieren Sie die Clients im selben Availability Zone- und VPC-Subnetz wie der Cluster
- Skalieren Sie Ihren -C
- Instances der aktuellen Generation verwenden
- Erhöhung der Netzwerkbandbreite
- Überwachen Sie Maximen für die Netzwerkleistung
- Überwachen Sie auf Transaktionen im Status „Leerlauf in Transaktion“

Platzieren Sie die Clients im selben Availability Zone- und VPC-Subnetz wie der Cluster

Um die Netzwerklatenz zu reduzieren und den Netzwerkdurchsatz zu erhöhen, platzieren Sie Clients in dieselbe Availability Zone und Virtual Private Cloud (VPC) -Subnetz wie der Aurora PostgreSQL DB-Cluster. Stellen Sie sicher, dass sich die Clients so geografisch wie möglich am DB-Cluster befinden.

Skalieren Sie Ihren -C

Bestimmen Sie mithilfe von Amazon CloudWatch oder anderen Host-Metriken, ob Ihr Client derzeit durch CPU- oder Netzwerkbandbreite oder beides eingeschränkt ist. Wenn der Kunde eingeschränkt ist, skalieren Sie Ihren Kunden entsprechend.

Instances der aktuellen Generation verwenden

In einigen Fällen verwenden Sie möglicherweise keine DB-Instance-Klasse, die Jumbo-Frames unterstützt. Wenn Sie Ihre Anwendung auf Amazon EC2 ausführen, sollten Sie eine Instance der aktuellen Generation für den Client verwenden. Konfigurieren Sie außerdem die maximale Übertragungseinheit (MTU) im Kundenvorgangssystem Diese Technik könnte die Anzahl der Netzläufe reduzieren und den Netzwerkdurchsatz erhöhen. Weitere Informationen finden Sie unter Jumbo-Frames (9.001 MTU) im Amazon-EC2-Benutzerhandbuch.

Weitere Informationen zu DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora. Um die DB-Instance-Klasse zu bestimmen, die einem Amazon EC2-Instance-Typ entspricht, platzieren Sie `db.` vor dem Namen des Amazon EC2-Instance-Typs. Beispielsweise entspricht die `r5.8xlarge`-Amazon EC2-Instance der `db.r5.8xlarge`-DB-Instance-Klasse.

Erhöhung der Netzwerkbandbreite

Verwenden Sie `NetworkReceiveThroughput`- und `NetworkTransmitThroughput`-Amazon CloudWatch-Metriken, um den ein- und ausgehenden Netzwerkverkehr auf dem DB-Cluster zu überwachen. Diese Metriken können Ihnen helfen festzustellen, ob die Netzwerkbandbreite für Ihre Workload ausreicht.

Wenn Ihre Netzwerkbandbreite nicht ausreicht, erhöhen Sie sie. Wenn der AWS-Client oder Ihre DB-Instance die Grenzen der Netzwerkbandbreite erreicht, können Sie die Bandbreite nur erhöhen, indem Sie die Größe Ihrer DB-Instance erhöhen.

Weitere Informationen zu CloudWatch-Metriken finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora.

Überwachen Sie Maximen für die Netzwerkleistung

Wenn Sie Amazon EC2-Clients verwenden, bietet Amazon EC2 Maximum für Netzwerkleistungsmetriken, einschließlich der aggregierten eingehenden und ausgehenden Netzwerkbandbreite. Es bietet auch Verbindungsverfolgung, um sicherzustellen, dass Pakete wie erwartet zurückgegeben werden, und Zugriff auf Link-lokale Dienste für Dienste wie das Domain

Name System (DNS). Um diese Maximen zu überwachen, verwenden Sie einen aktuell erweiterten Netzwerktreiber und überwachen Sie die Netzwerkleistung für Ihren Client.

Weitere Informationen finden Sie unter Überwachen der Netzwerkleistung für Ihre Amazon-EC2-Instance im Amazon-EC2-Benutzerhandbuch und unter Überwachen der Netzwerkleistung für Ihre Amazon-EC2-Instance im Amazon-EC2-Benutzerhandbuch.

Überwachen Sie auf Transaktionen im Status „Leerlauf in Transaktion“

Überprüfen Sie, ob Sie eine steigende Anzahl von `idle in transaction`-Verbindungen haben. Beobachten Sie dazu die `state`-Spalte in der `pg_stat_activity`-Tabelle. Möglicherweise können Sie die Verbindungsquelle identifizieren, indem Sie eine Abfrage ähnlich der folgenden ausführen.

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

Client:ClientWrite

Das `Client:ClientWrite`-Ereignis tritt auf, wenn Aurora PostgreSQL darauf wartet, Daten an den Client zu schreiben.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für Aurora PostgreSQL Version 10 und höher unterstützt.

Kontext

Ein Clientprozess muss alle Daten lesen, die von einem Aurora PostgreSQL DB-Cluster empfangen wurden, bevor der Cluster weitere Daten senden kann. Die Zeit, die der Cluster wartet, bevor weitere Daten an den Client gesendet werden, ist ein `Client:ClientWrite`-Ereignis.

Ein reduzierter Netzwerkdurchsatz zwischen dem Aurora PostgreSQL DB-Cluster und dem Client kann dieses Ereignis verursachen. CPU-Druck und Netzwerksättigung auf dem Client können dieses Ereignis ebenfalls verursachen. CPU-Druck liegt vor, wenn die CPU voll ausgelastet ist und Aufgaben auf CPU-Zeit warten. Eine Netzwerksättigung liegt vor, wenn das Netzwerk zwischen Datenbank und Client mehr Daten überträgt, als es verarbeiten kann.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Häufige Gründe dafür, dass das `Client:ClientWrite`-Ereignis in den Top-Wartezeiten angezeigt wird, sind die folgenden:

Erhöhte Netzwerklatenz

Es kann zu einer erhöhten Netzwerklatenz zwischen dem Aurora PostgreSQL-DB-Cluster und dem Client kommen. Eine höhere Netzwerklatenz erhöht die Zeit, die der Client benötigt, um die Daten zu empfangen.

Erhöhte Belastung des Clients

Auf dem Client kann es zu CPU-Druck oder Netzwerksättigung kommen. Eine Erhöhung der Belastung des Clients verzögert den Empfang von Daten aus dem Aurora PostgreSQL DB-Cluster.

Große Datenmenge, die an den Kunden gesendet werden

Der Aurora PostgreSQL DB-Cluster sendet möglicherweise eine große Datenmenge an den Client. Ein Client kann die Daten möglicherweise nicht so schnell empfangen, wie der Cluster sie sendet. Aktivitäten wie das Kopieren einer großen Tabelle können zu einer Zunahme von `Client:ClientWrite`-Ereignissen führen.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Platzieren Sie die Clients im selben Availability Zone- und VPC-Subnetz wie der Cluster
- Instances der aktuellen Generation verwenden
- Reduzieren Sie die an den Kunden gesendeten Daten
- Skalieren Sie Ihren -C

Platzieren Sie die Clients im selben Availability Zone- und VPC-Subnetz wie der Cluster

Um die Netzwerklatenz zu reduzieren und den Netzwerkdurchsatz zu erhöhen, platzieren Sie Clients in dieselbe Availability Zone und Virtual Private Cloud (VPC) -Subnetz wie der Aurora PostgreSQL DB-Cluster.

Instances der aktuellen Generation verwenden

In einigen Fällen verwenden Sie möglicherweise keine DB-Instance-Klasse, die Jumbo-Frames unterstützt. Wenn Sie Ihre Anwendung auf Amazon EC2 ausführen, sollten Sie eine Instance der aktuellen Generation für den Client verwenden. Konfigurieren Sie außerdem die maximale Übertragungseinheit (MTU) im Kundenvorgangssystem Diese Technik könnte die Anzahl der Netzläufe reduzieren und den Netzwerkdurchsatz erhöhen. Weitere Informationen finden Sie unter Jumbo-Frames (9.001 MTU) im Amazon-EC2-Benutzerhandbuch.

Weitere Informationen zu DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora. Um die DB-Instance-Klasse zu bestimmen, die einem Amazon EC2-Instance-Typ entspricht, platzieren Sie `db.` vor dem Namen des Amazon EC2-Instance-Typs. Beispielsweise entspricht die `r5.8xlarge`-Amazon EC2-Instance der `db.r5.8xlarge`-DB-Instance-Klasse.

Reduzieren Sie die an den Kunden gesendeten Daten

Passen Sie Ihre Anwendung nach Möglichkeit an, um die Datenmenge zu reduzieren, die der Aurora PostgreSQL DB-Cluster an den Client sendet. Solche Anpassungen entlasten die CPU- und Netzwerkkonflikte auf dem Client.

Skalieren Sie Ihren -C

Bestimmen Sie mithilfe von Amazon CloudWatch oder anderen Host-Metriken, ob Ihr Client derzeit durch CPU- oder Netzwerkbandbreite oder beides eingeschränkt ist. Wenn der Client eingeschränkt ist, skalieren Sie Ihren Client entsprechend.

CPU

Dieses Ereignis tritt auf, wenn ein Thread in der CPU aktiv ist oder auf die CPU wartet.

Themen

- Unterstützte Engine-Versionen
- Kontext

- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen sind für Aurora PostgreSQL Version 9.6 und höher relevant.

Kontext

Die Zentrale Verarbeitungseinheit (CPU) ist die Komponente eines Computers, die Anweisungen ausführt. Beispielsweise führen CPU-Anweisungen arithmetische Vorgänge aus und tauschen Daten im Speicher aus. Wenn eine Abfrage die Anzahl der Anweisungen erhöht, die sie über das Datenbank-Engine ausführt, erhöht sich der Zeitaufwand für die Ausführung der Abfrage. CPU-Scheduling gibt einem Prozess CPU-Zeit. Die Planung wird vom Kernel des Betriebssystems orchestriert.

Themen

- Wie kann man sagen, wann diese Wartezeit stattfindet
- DbloadCPU-Metrik
- Metriken für Os.cPuUtilization
- Wahrscheinliche Ursache für CPU-Planung

Wie kann man sagen, wann diese Wartezeit stattfindet

Dieses CPU-Warteereignis zeigt an, dass ein Backend-Prozess in der CPU aktiv ist oder auf die CPU wartet. Sie wissen, dass es passiert, wenn eine Abfrage die folgenden Informationen anzeigt:

- Die Spalte `pg_stat_activity.state` hat den Wert `active`.
- Die Spalten `wait_event_type` und `wait_event` in `pg_stat_activity` sind beide `null`.

Führen Sie die folgende Abfrage aus, um die Backend-Prozesse anzuzeigen, die auf der CPU verwenden oder auf dieser warten.

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
```

```
AND    wait_event IS NULL;
```

DbloadCPU-Metrik

Die Performance Insights-Metrik für CPU ist `DBLoadCPU`. Der Wert für `DBLoadCPU` kann vom Wert für die Amazon CloudWatch-Metrik `CPUUtilization` abweichen. Die letztere Metrik wird vom HyperVisor für eine Datenbank-Instance gesammelt.

Metriken für Os.cPuUtilization

Performance Insights Betriebssystem-Metriken liefern detaillierte Informationen zur CPU-Auslastung. Sie können beispielsweise die folgenden Metriken anzeigen:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

Performance Insights meldet die CPU-Auslastung durch die Datenbank-Engine als `os.cpuUtilization.nice.avg`.

Wahrscheinliche Ursache für CPU-Planung

Aus Sicht des Betriebssystems ist die CPU aktiv, wenn der Leerlauf-Thread nicht ausgeführt wird. Die CPU ist aktiv, während sie eine Berechnung durchführt, aber sie ist auch aktiv, wenn sie auf Speicher-I/O wartet. Diese Art von I/O dominiert eine typische Datenbank-Workload.

Wenn die folgenden Bedingungen erfüllt sind, werden Prozesse wahrscheinlich warten, bis die folgenden Bedingungen erfüllt sind:

- Die CloudWatch `CPUUtilization`-Metrik liegt nahe bei 100 Prozent.
- Die durchschnittliche Belastung ist größer als die Anzahl der vCPUs, was auf eine hohe Last hinweist. Sie finden die `loadAverageMinute`-Metrik im Abschnitt Betriebssystemmetriken in Performance Insights.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das CPU-Warteereignis mehr als normal auftritt und möglicherweise auf ein Leistungsproblem hinweist, sind typische Ursachen die folgenden.

Themen

- Wahrscheinliche Ursachen für plötzliche Stacheln
- Wahrscheinliche Ursachen für langfristige Hochfrequenz
- Corner Cases

Wahrscheinliche Ursachen für plötzliche Stacheln

Die wahrscheinlichsten Ursachen für plötzliche Stacheln sind wie folgt:

- Ihre Anwendung hat zu viele gleichzeitige Verbindungen zur Datenbank geöffnet. Dieses Szenario wird als „Verbindungssturm“ bezeichnet.
- Ihre Anwendungs-Workload hat sich auf eine der folgenden Weisen geändert:
  - Neue Anfragen
  - Eine Zunahme der Größe Ihres Datensatzes
  - Indexpflege oder -erstellung
  - Neue Funktionen
  - Neue Betreiber
  - Eine Zunahme der parallelen Abfrageausführung
- Ihre Abfrageausführungspläne haben sich geändert. In einigen Fällen kann eine Änderung zu einem Anstieg der Puffer führen. Beispielsweise verwendet die Abfrage jetzt einen sequentiellen Scan, als sie zuvor einen Index verwendet hat. In diesem Fall benötigen die Abfragen mehr CPU, um dasselbe Ziel zu erreichen.

Wahrscheinliche Ursachen für langfristige Hochfrequenz

Die wahrscheinlichsten Ursachen für Ereignisse, die über einen langen Zeitraum auftreten:

- Zu viele Backend-Prozesse laufen gleichzeitig auf der CPU. Diese Prozesse können parallele Arbeiter sein.
- Abfragen funktionieren suboptimal, da sie eine große Anzahl von Puffern benötigen.

Corner Cases

Wenn sich herausstellt, dass keine der wahrscheinlichen Ursachen tatsächliche Ursachen ist, können folgende Situationen auftreten:

- Die CPU tauscht Prozesse ein- und aus.
- Der CPU-Kontextwechsel hat zugenommen.
- Aurora PostgreSQL-Code fehlen Warteereignisse.

Aktionen

Wenn das CPU-Wait-Ereignis die Datenbankaktivität dominiert, weist dies nicht unbedingt auf ein Leistungsproblem hin. Reagieren Sie auf dieses Ereignis nur, wenn sich die Leistung verschlechtert.

Themen

- Untersuchen Sie, ob die Datenbank den CPU-Anstieg verursacht
- Bestimmen Sie, ob die Anzahl der Verbindungen gestiegen ist
- Reagieren auf Workload-Änderungen

Untersuchen Sie, ob die Datenbank den CPU-Anstieg verursacht

Untersuchen Sie die `os.cpuUtilization.nice.avg`-Metrik in Performance Insights. Wenn dieser Wert weit unter der CPU-Auslastung liegt, tragen Nicht-Datenbankprozesse den Hauptbeitrag zur CPU bei.

Bestimmen Sie, ob die Anzahl der Verbindungen gestiegen ist

Untersuchen Sie die Metrik `DatabaseConnections` in Amazon CloudWatch. Ihre Aktion hängt davon ab, ob die Zahl während des Zeitraums erhöhter CPU-Warteereignisse erhöht oder gesunken ist.

Die Verbindungen nahmen zu

Wenn die Anzahl der Verbindungen gestiegen ist, vergleichen Sie die Anzahl der Backend-Prozesse, die CPU verbrauchen, mit der Anzahl der vCPUs. Die folgenden Szenarien sind möglich:

- Die Anzahl der Backend-Prozesse, die CPU verbrauchen, ist geringer als die Anzahl der vCPUs.

  In diesem Fall ist die Anzahl der Verbindungen kein Problem. Möglicherweise versuchen Sie jedoch weiterhin, die CPU-Auslastung zu reduzieren.
- Die Anzahl der Backend-Prozesse, die CPU verbrauchen, ist größer als die Anzahl der vCPUs.

  Ziehen Sie in diesem Fall die folgenden Optionen in Betracht:

- Verringern Sie die Anzahl der Backend-Prozesse, die mit Ihrer Datenbank verbunden sind. Implementieren Sie beispielsweise eine Verbindungs-Pooling-Lösung wie RDS Proxy. Weitere Informationen hierzu finden Sie unter Amazon-RDS-Proxy für Aurora.
- Aktualisieren Sie Ihre Instance-Größe, um eine höhere Anzahl von vCPUs zu erhalten.
- Leiten Sie ggf. einige schreibgeschützte Workloads auf Reader-Knoten um.

Die Verbindungen haben nicht zugenommen

Untersuchen Sie die `blks_hit`-Metriken in Performance Insights. Suchen Sie nach einer Korrelation zwischen einem Anstieg von `blks_hit` und der CPU-Auslastung. Die folgenden Szenarien sind möglich:

- CPU-Auslastung und `blks_hit` sind korreliert.

  Suchen Sie in diesem Fall die wichtigsten SQL-Anweisungen, die mit der CPU-Auslastung verknüpft sind, und suchen Sie nach Planänderungen. Sie können eine der folgenden Techniken verwenden:

  - Erklären Sie die Pläne manuell und vergleichen Sie sie mit dem erwarteten Ausführungsplan.
  - Achten Sie auf eine Zunahme der Blocktreffer pro Sekunde und lokalen Blocktreffern pro Sekunde. Wählen Sie im Abschnitt Top-SQL des Performance Insights-Dashboards Einstellungen aus.
- CPU-Auslastung und `blks_hit` sind nicht korreliert.

  Stellen Sie in diesem Fall fest, ob einer der folgenden Fälle auftritt:

  - Die Anwendung stellt schnell eine Verbindung zur Datenbank her und trennt sie von dieser.

    Diagnostizieren Sie dieses Verhalten, indem Sie `log_connections` und `log_disconnections` aktivieren und dann die PostgreSQL-Protokolle analysieren. Ziehen Sie in Erwägung, den `pgbadger`-Protokoll-Analyzer zu verwenden. Weitere Informationen finden Sie unter https://github.com/darold/pgbadger.
  - Das Betriebssystem ist überlastet.

    In diesem Fall zeigt Performance Insights, dass Backend-Prozesse länger CPU verbrauchen als gewöhnlich. Suchen Sie in den Metriken von Performance Insights `os.cpuUtilization` oder der Metrik CloudWatch `CPUUtilization` nach Beweisen. Wenn das Betriebssystem überlastet ist, sehen Sie sich Enhanced Monitoring-Metriken an, um eine weitere Diagnose zu erhalten.

Schauen Sie sich insbesondere die Prozessliste und den Prozentsatz der CPU an, die von jedem Prozess verbraucht wird.

- Top SQL-Anweisungen verbrauchen zu viel CPU.

  Untersuchen Sie Anweisungen, die mit der CPU-Auslastung verknüpft sind, um festzustellen, ob sie weniger CPU verbrauchen können. Führen Sie einen `EXPLAIN`-Befehl aus und konzentrieren Sie sich auf die Planknoten, die die größte Auswirkung haben. Erwägen Sie, einen Visualizer für PostgreSQL-Ausführungspläne zu verwenden. Um dieses Tool auszuprobieren, siehe http://explain.dalibo.com/.

Reagieren auf Workload-Änderungen

Wenn sich Ihre Workload geändert hat, suchen Sie nach folgenden Arten von Änderungen:

Neue Anfragen

Überprüfen Sie, ob die neuen Abfragen erwartet werden. Stellen Sie in diesem Fall sicher, dass ihre Ausführungspläne und die Anzahl der Ausführungen pro Sekunde erwartet werden.

Eine Zunahme der Größe des Datensatzes

Bestimmen Sie, ob die Partitionierung, falls sie noch nicht implementiert ist, helfen könnte. Diese Strategie könnte die Anzahl der Seiten verringern, die eine Abfrage abrufen muss.

Indexpflege oder -erstellung

Überprüfen Sie, ob der Zeitplan für die Wartung erwartet wird. Eine bewährte Methode besteht darin, Wartungsarbeiten außerhalb der Hauptverkehrszeiten zu planen.

Neue Funktionen

Überprüfen Sie, ob diese Funktionen während des Tests erwartungsgemäß funktionieren. Überprüfen Sie insbesondere, ob die Anzahl der Ausführungen pro Sekunde erwartet wird.

Neue Betreiber

Überprüfen Sie, ob sie während des Tests erwartungsgemäß funktionieren.

Eine Zunahme der laufenden parallelen Abfragen

Stellen Sie fest, ob eine der folgenden Situationen aufgetreten ist:

- Die beteiligten Relationen oder Indizes sind plötzlich so groß geworden, dass sie sich deutlich von `min_parallel_table_scan_size` oder `min_parallel_index_scan_size` unterscheiden.

- Die letzten Änderungen wurden an `parallel_setup_cost` oder `parallel_tuple_cost` vorgenommen.
- Die letzten Änderungen wurden an `max_parallel_workers` oder `max_parallel_workers_per_gather` vorgenommen.

IO: BufFileRead und IO: BufFileWrite

Die Ereignisse `IO:BufFileRead` und `IO:BufFileWrite` treten auf, wenn Aurora PostgreSQL temporäre Dateien erstellt. Wenn Vorgänge mehr Arbeitsspeicher benötigen, als die derzeit definierten Arbeitsspeicherparameter definieren, schreiben sie temporäre Daten in persistenten Speicher. Dieser Vorgang wird manchmal als Übertragung auf die Festplatte bezeichnet. Weitere Informationen zu den temporären Dateien und ihrer Verwendung finden Sie unter Verwalten temporärer Dateien mit PostgreSQL.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

`IO:BufFileRead` und `IO:BufFileWrite` beziehen sich auf den Arbeitsspeicherbereich und den Wartungsarbeitsspeicherbereich. Weitere Informationen zu diesen lokalen Speicherbereichen finden Sie unter Arbeitsspeicherbereich und Wartungs-Arbeitsspeicherbereich.

Der Standardwert für `work_mem` ist 4 MB. Wenn eine Sitzung parallel Vorgänge ausführt, verwendet jeder Worker, der die Parallelität bearbeitet, 4 MB Speicher. Stellen Sie `work_mem` daher sorgfältig ein. Wenn Sie den Wert zu stark erhöhen, verbraucht eine Datenbank mit vielen Sitzungen möglicherweise zu viel Speicher. Wenn Sie den Wert zu niedrig festlegen, erstellt Aurora PostgreSQL temporäre Dateien im lokalen Speicher. Die Festplatte I/O für diese temporären Dateien kann die Leistung beeinträchtigen.

Wenn Sie die folgende Ereignisfolge beobachten, generiert Ihre Datenbank möglicherweise temporäre Dateien:

1. Plötzlicher und starker Rückgang der Verfügbarkeit
2. Schnelle Erholung für den freien Speicherplatz

Möglicherweise sehen Sie auch ein „Kettensäge“ -Muster. Dieses Muster kann darauf hinweisen, dass Ihre Datenbank ständig kleine Dateien erstellt.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Im Allgemeinen werden diese Warteereignisse durch Vorgänge verursacht, die mehr Speicher verbrauchen, als die Parameter `work_mem` oder `maintenance_work_mem` zuweisen. Um dies zu kompensieren, schreiben die Vorgänge in temporäre Dateien. Häufige Ursachen für die `IO:BufFileRead`- und `IO:BufFileWrite`-Ereignisse sind die folgenden:

Abfragen, die mehr Speicher benötigen als im Arbeitsspeicherbereich vorhanden

Abfragen mit den folgenden Merkmalen verwenden den Arbeitsspeicherbereich:

- Hash-Verknüpfungen
- `ORDER BY`-Klausel
- `GROUP BY`-Klausel
- `DISTINCT`
- Fensterfunktionen
- `CREATE TABLE AS SELECT`
- Aktualisierung der materialisierten Ansicht

Anweisungen, die mehr Speicher benötigen als im Arbeitsspeicherbereich für Wartungsarbeiten vorhanden

Die folgenden Anweisungen verwenden den Arbeitsspeicherbereich für Wartungsarbeiten:

- `CREATE INDEX`
- `CLUSTER`

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Identifizieren Sie das Problem
- Untersuchen Sie Ihre Join-Anfragen
- Überprüfen Sie Ihre ORDER BY- und GROUP BY Anfragen
- Verwenden Sie den DISTINCT-Vorgang nicht
- Erwägen Sie, Fensterfunktionen anstelle von GROUP-BY-Funktionen zu verwenden
- Untersuchen Sie materialisierte Ansichten und CTAS-Aussagen
- Verwenden Sie pg_repack, wenn Sie Indizes erstellen
- Erhöhen Sie maintenance work_mem, wenn Sie Tabellen clustern
- Passen Sie den Speicher an, um IO: BufFileRead und IO: zu verhindern BufFileWrite

Identifizieren Sie das Problem

Sie können die Nutzung temporärer Dateien direkt in Performance Insights anzeigen. Weitere Informationen finden Sie unter Anzeigen der Nutzung temporärer Dateien mit Performance Insights. Wenn Performance Insights deaktiviert ist, stellen Sie möglicherweise eine Zunahme der `IO:BufFileRead` `IO:BufFileWrite` Betriebsabläufe fest. Gehen Sie wie folgt vor, um Fehler zu beheben:

1. Untersuchen Sie die `FreeLocalStorage` Metrik in Amazon CloudWatch.
2. Suchen Sie nach einem Kettensägenmuster, bei dem es sich um eine Reihe von gezackten Stacheln handelt.

Ein Kettensägenmuster weist auf einen schnellen Verbrauch und die Freigabe von Speicher hin, die häufig mit temporären Dateien verbunden sind. Wenn Sie dieses Muster bemerken, aktivieren Sie Performance Insights. Wenn Sie Performance Insights verwenden, können Sie feststellen, wann die Wait-Ereignisse auftreten und welche Abfragen mit ihnen verknüpft sind. Ihre Lösung hängt von der spezifischen Abfrage ab, die die Ereignisse verursacht.

Oder stellen Sie den Parameter `log_temp_files` ein. Dieser Parameter protokolliert alle Abfragen, die mehr als Schwellenwert für temporäre Dateien erzeugen. Wenn der Wert `0` ist, protokolliert

Aurora PostgreSQL alle temporären Dateien. Wenn der Wert `1024` ist, protokolliert Aurora PostgreSQL alle Abfragen, die temporäre Dateien erzeugen, die größer als 1 MB sind. Weitere Informationen zu `log_temp_files` finden Sie unter Fehlerberichte und -protokollierung in der PostgreSQL-Dokumentation.

Untersuchen Sie Ihre Join-Anfragen

Ihre Anwendung verwendet wahrscheinlich Joins. Die folgende Abfrage verbindet beispielsweise vier Tabellen.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

Eine mögliche Ursache für Spitzen bei der Verwendung temporärer Dateien ist ein Problem in der Abfrage selbst. Beispielsweise filtert eine defekte Klausel die Joins möglicherweise nicht richtig. Betrachten Sie den zweiten inneren Join im folgenden Beispiel.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

Die obige Abfrage verknüpft fälschlicherweise `customer.id` mit `customer.id`, wodurch zwischen jedem Kunden und jeder Bestellung ein kartesisches Produkt generiert wird. Diese Art des versehentlichen Joins generiert große temporäre Dateien. Abhängig von der Größe der Tabellen kann eine kartesische Abfrage sogar Speicher füllen. Wenn die folgenden Bedingungen erfüllt sind, hat Ihre Anwendung möglicherweise kartesische Joins:

- Sie sehen einen großen, starken Rückgang der Speicherverfügbarkeit, gefolgt von einer schnellen Wiederherstellung.
- Es werden keine Indizes erstellt.
- Es werden keine `CREATE TABLE FROM SELECT`-Anweisungen ausgegeben.
- Es werden keine materialisierten Ansichten aktualisiert.

Um festzustellen, ob die Tabellen mit den richtigen Schlüsseln verbunden werden, überprüfen Sie Ihre Abfrage- und objektrelationalen Mapping-Anweisungen. Denken Sie daran, dass bestimmte Abfragen Ihrer Anwendung nicht ständig aufgerufen werden und einige Abfragen dynamisch generiert werden.

Überprüfen Sie Ihre ORDER BY- und GROUP BY Anfragen

In einigen Fällen kann eine `ORDER BY`-Klausel zu übermäßig vielen temporären Dateien führen. Berücksichtigen Sie die folgenden Hinweise:

- Schließen Sie Spalten nur dann in eine `ORDER BY`-Klausel ein, wenn sie sortiert werden müssen. Diese Richtlinie ist besonders wichtig für Abfragen, die Tausende von Zeilen zurückgeben und viele Spalten in der `ORDER BY`-Klausel angeben.
- Ziehen Sie in Betracht, Indizes zu erstellen, um `ORDER BY`-Klauseln zu beschleunigen, wenn sie mit Spalten übereinstimmen, die dieselbe aufsteigende oder absteigende Reihenfolge aufweisen. Partielle Indizes sind vorzuziehen, da sie kleiner sind. Kleinere Indizes werden schneller gelesen und durchquert.
- Wenn Sie Indizes für Spalten erstellen, die Nullwerte akzeptieren können, überlegen Sie, ob die Nullwerte am Ende oder am Anfang der Indizes gespeichert werden sollen.

  Reduzieren Sie nach Möglichkeit die Anzahl der Zeilen, die sortiert werden müssen, indem Sie die Ergebnismenge filtern. Wenn Sie `WITH`-Klausel-Anweisungen oder Unterabfragen verwenden, denken Sie daran, dass eine innere Abfrage eine Ergebnismenge generiert und an die äußere Abfrage übergibt. Je mehr Zeilen eine Abfrage herausfiltern kann, desto weniger muss die Abfrage erledigen.
- Wenn Sie nicht die vollständige Ergebnismenge abrufen müssen, verwenden Sie die `LIMIT`-Klausel. Wenn Sie beispielsweise nur die obersten fünf Zeilen benötigen, generiert eine Abfrage mit der `LIMIT`-Klausel keine Ergebnisse. Auf diese Weise benötigt die Abfrage weniger Speicher und temporäre Dateien.

Eine Abfrage, die eine GROUP BY-Klausel verwendet, kann auch temporäre Dateien erfordern. GROUP BY-Abfragen fassen Werte mithilfe von Funktionen wie den folgenden zusammen:

- COUNT
- AVG
- MIN
- MAX
- SUM
- STDDEV

Befolgen Sie zum Optimieren von GROUP BY-Abfragen die Empfehlungen für ORDER BY-Abfragen.

Verwenden Sie den DISTINCT-Vorgang nicht

Vermeiden Sie nach Möglichkeit die Verwendung des DISTINCT-Vorgangs, um doppelte Zeilen zu entfernen. Je mehr unnötige und doppelte Zeilen Ihre Abfrage zurückgibt, desto teurer wird der DISTINCT-Vorgang. Fügen Sie nach Möglichkeit Filter in der WHERE-Klausel hinzu, auch wenn Sie dieselben Filter für verschiedene Tabellen verwenden. Das Filtern der Abfrage und der korrekte Beitritt verbessern Ihre Leistung und reduziert den Ressourcennutzung. Es verhindert auch falsche Berichte und Ergebnisse.

Wenn Sie DISTINCT für mehrere Zeilen derselben Tabelle verwenden müssen, sollten Sie einen zusammengesetzten Index erstellen. Das Gruppieren mehrerer Spalten in einem Index kann die Zeit zum Auswerten verschiedener Zeilen verbessern. Wenn Sie Amazon-Aurora-PostgreSQL-Version 10 oder höher verwenden, können Sie außerdem mithilfe des CREATE STATISTICS-Befehls Statistiken zwischen mehreren Spalten korrelieren.

Erwägen Sie, Fensterfunktionen anstelle von GROUP-BY-Funktionen zu verwenden

Mit GROUP BY ändern Sie die Ergebnismenge und rufen dann das aggregierte Ergebnis ab. Mithilfe von Fensterfunktionen aggregieren Sie Daten, ohne die Ergebnismenge zu ändern. Eine Fensterfunktion verwendet die OVER-Klausel, um Berechnungen über die von der Abfrage definierten Mengen durchzuführen und eine Zeile mit einer anderen zu korrelieren. Sie können alle GROUP BY-Funktionen in Fensterfunktionen verwenden, aber auch Funktionen wie die folgenden verwenden:

- RANK
- ARRAY_AGG
- ROW_NUMBER

- LAG
- LEAD

Um die Anzahl der temporären Dateien zu minimieren, die von einer Fensterfunktion generiert werden, entfernen Sie Duplikationen für dieselbe Ergebnismenge, wenn Sie zwei verschiedene Aggregationen benötigen. Betrachten Sie folgende Abfrage.

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
  FROM empsalary;
```

Sie können die Abfrage mit der WINDOW-Klausel wie folgt umschreiben.

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

Standardmäßig konsolidiert der Ausführungsplaner von Aurora PostgreSQL ähnliche Knoten, sodass keine Vorgänge dupliziert werden. Durch die Verwendung einer expliziten Deklaration für den Fensterblock können Sie die Abfrage jedoch einfacher pflegen. Sie können die Leistung auch verbessern, indem Sie Doppelarbeit verhindern.

Untersuchen Sie materialisierte Ansichten und CTAS-Aussagen

Wenn eine materialisierte Ansicht aktualisiert wird, wird eine Abfrage ausgeführt. Diese Abfrage kann einen Vorgang wie GROUP BY, ORDER BY oder DISTINCT enthalten. Während einer Aktualisierung können Sie eine große Anzahl temporärer Dateien und die Warteereignisse IO:BufFileWrite und IO:BufFileRead beobachten. Wenn Sie eine Tabelle basierend auf einer SELECT-Anweisung erstellen, führt die CREATE TABLE-Anweisung eine Abfrage aus. Um die benötigten temporären Dateien zu reduzieren, optimieren Sie die Abfrage.

Verwenden Sie pg_repack, wenn Sie Indizes erstellen

Wenn Sie einen Index erstellen, ordnet die Engine die Ergebnismenge an. Wenn Tabellen größer werden und die Werte in der indizierten Spalte vielfältiger werden, benötigen die temporären Dateien mehr Speicherplatz. In den meisten Fällen können Sie die Erstellung temporärer Dateien für große Tabellen nicht verhindern, ohne den Speicherbereich für Wartungsarbeiten zu ändern. Weitere Informationen finden Sie unter Wartungs-Arbeitsspeicherbereich.

Eine mögliche Problemumgehung beim Neuerstellen eines großen Index besteht darin, das pg_repack-Tool zu verwenden. Weitere Informationen finden Sie unter Reorganisieren von Tabellen in PostgreSQL-Datenbanken mit minimalen Sperren in der pg_repack-Dokumentation.

Erhöhen Sie maintenance work_mem, wenn Sie Tabellen clustern

Der Befehl `CLUSTER` gruppiert die durch table_name angegebene Tabelle basierend auf einem vorhandenen Index, der durch index_name angegeben wird. Aurora PostgreSQL erstellt die Tabelle physisch so neu, dass sie der Reihenfolge eines bestimmten Indexes entspricht.

Als magnetische Speicherung vorherrschend war, war Clustering üblich, da der Speicherdurchsatz begrenzt war. Jetzt, da SSD-basierter Speicher üblich ist, ist Clustering weniger beliebt. Wenn Sie jedoch Tabellen clustern, können Sie die Leistung je nach Tabellengröße, Index, Abfrage usw. immer noch geringfügig steigern.

Wenn Sie den Befehl `CLUSTER` ausführen und die Warteereignisse `IO:BufFileWrite` und `IO:BufFileRead` beobachten, stimmen Sie `maintenance_work_mem` ab. Erhöhen Sie die Speichergröße auf einen ziemlich großen Betrag. Ein hoher Wert bedeutet, dass die Engine mehr Speicher für den Clustering-Vorgang verwenden kann.

Passen Sie den Speicher an, um IO: BufFileRead und IO: zu verhindern BufFileWrite

In irgendeiner Situation müssen Sie den Speicher abstimmen. Ihr Ziel ist es, die folgenden Anforderungen auszugleichen:

- Der `work_mem`-Wert (siehe Arbeitsspeicherbereich)
- Der Speicher, der nach dem Diskontieren des`shared_buffers`Wert (siehePuffer Pool)
- Die maximale Anzahl geöffneter und verwendeter Verbindungen, die durch `max_connections` begrenzt ist

Erhöhen Sie die Größe des Arbeitsspeicherbereichs

In einigen Situationen besteht Ihre einzige Möglichkeit darin, den von Ihrer Sitzung verwendeten Speicher zu erhöhen. Wenn Ihre Abfragen richtig geschrieben sind und die richtigen Schlüssel für Joins verwenden, sollten Sie den `work_mem`-Wert erhöhen. Weitere Informationen finden Sie unter Arbeitsspeicherbereich.

Um herauszufinden, wie viele temporäre Dateien eine Abfrage generiert, setzen Sie `log_temp_files` auf `0`. Wenn Sie den `work_mem`-Wert auf den in den Protokollen angegebenen

Höchstwert erhöhen, verhindern Sie, dass die Abfrage temporäre Dateien generiert. `work_mem` legt jedoch das Maximum pro Planknoten für jede Verbindung oder jeden parallelen Worker fest. Wenn die Datenbank über 5.000 Verbindungen verfügt und jede 256 MiB-Speicher verwendet, benötigt die Engine 1,2 TiB RAM. Daher kann es sein, dass Ihrer Instance der Speicher knapp wird.

Reservieren Sie ausreichend Speicher für den freigegebenen Pufferpool

Ihre Datenbank verwendet Speicherbereiche wie den freigegebenen Pufferpool, nicht nur den Arbeitsspeicherbereich. Berücksichtigen Sie die Anforderungen dieser zusätzlichen Speicherbereiche, bevor Sie `work_mem` erhöhen. Weitere Informationen zum Pufferpool finden Sie unter Puffer Pool.

Angenommen, Ihre Aurora PostgreSQL-Instance-Klasse ist db.r5.2xlarge. Diese Klasse hat 64 GiB Speicher. Standardmäßig sind 75 Prozent des Arbeitsspeichers für den freigegebenen Pufferpool reserviert. Nachdem Sie den dem Shared Memory-Bereich zugewiesenen Betrag abgezogen haben, bleiben 16.384 MB übrig. Weisen Sie den verbleibenden Speicher nicht ausschließlich dem Arbeitsspeicherbereich zu, da das Betriebssystem und die Engine ebenfalls Speicher benötigen.

Der Speicher, den Sie `work_mem` zuordnen können, hängt von der Instance-Klasse ab. Wenn Sie eine größere Instance-Klasse verwenden, ist mehr Speicher verfügbar. Im vorhergehenden Beispiel können Sie jedoch nicht mehr als 16 GiB verwenden. Andernfalls ist Ihre Instance nicht verfügbar, wenn ihr der Speicher ausgeht. Um die Instance aus dem nicht verfügbaren Status wiederherzustellen, werden die Aurora PostgreSQL-Automatisierungsdienste automatisch neu gestartet.

Verwalten der Anzahl der Verbindungen

Angenommen, Ihre Datenbank-Instance hat 5.000 gleichzeitige Verbindungen. Jede Verbindung verwendet mindestens 4 MB `work_mem`. Der hohe Speicherverbrauch der Verbindungen dürfte die Leistung beeinträchtigen. Als Reaktion darauf haben Sie die folgenden Optionen:

- Aktualisieren Sie auf eine größere Instance-Klasse.
- Verringern Sie die Anzahl gleichzeitiger Datenbankverbindungen mit einem Verbindungsproxy oder Pooler.

Berücksichtigen Sie bei Proxys Amazon RDS Proxy, PGBouncer oder einen auf Ihrer Anwendung basierenden Verbindungspooler. Diese Lösung lindert die CPU-Last. Es reduziert auch das Risiko, wenn alle Verbindungen den Arbeitsspeicherbereich benötigen. Wenn weniger Datenbankverbindungen vorhanden sind, können Sie den Wert von `work_mem` erhöhen. Auf

diese Weise reduzieren Sie das Auftreten der `IO:BufFileRead`- und `IO:BufFileWrite`-Warteereignisse. Auch die Abfragen, die auf den Arbeitsspeicherbereich warten, beschleunigen sich erheblich.

IO:DataFileRead

Das `IO:DataFileRead`-Ereignis tritt auf, wenn eine Verbindung darauf wartet, dass ein Backend-Prozess eine erforderliche Seite aus dem Speicher liest, da die Seite nicht im gemeinsam genutzten Speicher verfügbar ist.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Alle Abfragen und Datenmanipulationsvorgänge (DML) greifen auf Seiten im Pufferpool zu. Zu den Anweisungen, die Lesevorgänge auslösen können, gehören `SELECT`, `UPDATE` und `DELETE`. Ein `UPDATE` kann beispielsweise Seiten aus Tabellen oder Indizes lesen. Wenn sich die angeforderte oder aktualisierte Seite nicht im gemeinsam genutzten Pufferpool befindet, kann dieser Lesevorgang zum `IO:DataFileRead`-Ereignis führen.

Da der gemeinsame Pufferpool endlich ist, kann er sich füllen. In diesem Fall zwingen Anfragen nach Seiten, die sich nicht im Speicher befinden, die Datenbank dazu, Blöcke von der Festplatte zu lesen. Wenn das `IO:DataFileRead`-Ereignis häufig auftritt, ist Ihr gemeinsam genutzter Pufferpool möglicherweise zu klein für Ihre Workload. Dieses Problem ist bei `SELECT`-Abfragen akut, die eine große Anzahl von Zeilen lesen, die nicht in den Pufferpool passen. Weitere Informationen zum Pufferpool finden Sie unter Puffer Pool.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Häufige Ursachen für das Ereignis `IO:DataFileRead` sind die folgenden:

Verbindungsspitzen

Möglicherweise finden Sie mehrere Verbindungen, die die gleiche Anzahl von io:DataFileRead-Wait-Ereignissen erzeugen. In diesem Fall kann eine Spitze (plötzlicher und starker Anstieg) bei `IO:DataFileRead`-Ereignissen auftreten.

SELECT- und DML-Anweisungen, die sequentielle Scans durchführen

Ihre Anwendung führt möglicherweise einen neuen Vorgang aus. Oder ein vorhandener Vorgang könnte sich aufgrund eines neuen Ausführungsplans ändern. Suchen Sie in solchen Fällen nach Tabellen (insbesondere großen Tabellen), die einen größeren `seq_scan`-Wert haben. Finden Sie sie, indem Sie `pg_stat_user_tables` abfragen. Verwenden Sie die Erweiterung `pg_stat_statements`, um Abfragen zu verfolgen, die mehr Lesevorgänge generieren.

CTAS und CREATE INDEX für große Datensätze

Ein CTAS ist eine `CREATE TABLE AS SELECT`-Anweisung. Wenn Sie ein CTAS mit einem großen Datensatz als Quelle ausführen oder einen Index für eine große Tabelle erstellen, kann das `IO:DataFileRead`-Ereignis auftreten. Wenn Sie einen Index erstellen, muss die Datenbank möglicherweise das gesamte Objekt mithilfe eines sequentiellen Scans lesen. Ein CTAS generiert `IO:DataFile`-Reads, wenn Seiten nicht im Speicher sind.

Mehrere Vakuumarbeiter laufen gleichzeitig

Vakuumarbeiter können manuell oder automatisch ausgelöst werden. Wir empfehlen, eine aggressive Vakuumstrategie zu verabschieden. Wenn eine Tabelle jedoch viele aktualisierte oder gelöschte Zeilen enthält, erhöhen sich die `IO:DataFileRead`-Wartezeiten. Nachdem der Raum zurückgewonnen wurde, nimmt die für `IO:DataFileRead` aufgewendete Vakuumzeit ab.

Aufnahme großer Datenmengen

Wenn Ihre Anwendung große Datenmengen aufnimmt, können `ANALYZE`-Vorgänge häufiger auftreten. Der `ANALYZE`-Prozess kann durch einen Autovacuum Launcher ausgelöst oder manuell aufgerufen werden.

Der `ANALYZE`-Vorgang liest eine Teilmenge der Tabelle. Die Anzahl der zu scannenden Seiten wird berechnet, indem 30 mit dem `default_statistics_target`-Wert multipliziert wird. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation. Der Parameter `default_statistics_target` akzeptiert Werte zwischen 1 und 10.000, wobei der Standardwert 100 ist.

Hungertod

Wenn Netzwerkbandbreite oder CPU der Instance verbraucht werden, kann das `IO:DataFileRead`-Ereignis häufiger auftreten.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Überprüfen Sie Prädikatfilter auf Abfragen, die Wartezeiten generieren
- Minimieren Sie die Auswirkungen von Wartungsvorgängen
- Reagieren Sie auf eine hohe Anzahl von Verbindungen

Überprüfen Sie Prädikatfilter auf Abfragen, die Wartezeiten generieren

Angenommen, Sie identifizieren bestimmte Abfragen, die `IO:DataFileRead`-Warteereignisse generieren. Sie können sie mit den folgenden Techniken identifizieren:

- Performance Insights
- Katalogansichten wie die der Erweiterung `pg_stat_statements`
- Die Katalogansicht `pg_stat_all_tables`, wenn sie periodisch eine erhöhte Anzahl von physischen Lesevorgängen anzeigt
- Die `pg_statio_all_tables`-Ansicht, wenn sie zeigt, dass `_read`-Zähler steigen

Wir empfehlen Ihnen, zu bestimmen, welche Filter im Prädikat (`WHERE`-Klausel) dieser Abfragen verwendet werden. Befolgen Sie diese Richtlinien:

- Führen Sie den Befehl `EXPLAIN` aus. Geben Sie in der Ausgabe an, welche Arten von Scans verwendet werden. Ein sequentieller Scan weist nicht zwangsläufig ein Problem an. Abfragen, die sequenzielle Scans verwenden, erzeugen im Vergleich zu Abfragen, die Filter verwenden, natürlich mehr `IO:DataFileRead`-Ereignisse.

  Finden Sie heraus, ob die in der `WHERE`-Klausel aufgeführte Spalte indiziert ist. Wenn nicht, erwägen Sie, einen Index für diese Spalte zu erstellen. Dieser Ansatz vermeidet die sequentiellen Scans und reduziert die `IO:DataFileRead`-Ereignisse. Wenn eine Abfrage restriktive Filter

enthält und immer noch sequenzielle Scans erzeugt, bewerten Sie, ob die richtigen Indizes verwendet werden.

- Finden Sie heraus, ob die Abfrage auf eine sehr große Tabelle zugreift. In einigen Fällen kann das Partitionieren einer Tabelle die Leistung verbessern, sodass die Abfrage nur notwendige Partitionen lesen kann.
- Untersuchen Sie die Kardinalität (Gesamtzahl der Zeilen) Ihrer Join-Vorgänge. Beachten Sie, wie restriktiv die Werte sind, die Sie den Filtern für Ihre `WHERE`-Klausel übergeben. Wenn möglich, stimmen Sie Ihre Abfrage ein, um die Anzahl der Zeilen zu reduzieren, die in jedem Schritt des Plans übergeben werden.

Minimieren Sie die Auswirkungen von Wartungsvorgängen

Wartungsvorgänge wie `VACUUM` und `ANALYZE` sind wichtig. Wir empfehlen, sie nicht zu deaktivieren, da Sie `IO:DataFileRead`-Warteereignisse im Zusammenhang mit diesen Wartungsvorgängen finden. Die folgenden Ansätze können die Auswirkungen dieser Vorgänge minimieren:

- Führen Sie Wartungsvorgänge während außerhalb der Hauptverkehrszeiten manuell aus. Diese Technik verhindert, dass die Datenbank den Schwellenwert für automatische Vorgänge erreicht.
- Erwägen Sie bei sehr großen Tabellen, die Tabelle zu partitionieren. Diese Technik reduziert den Overhead von Wartungsvorgängen. Die Datenbank greift nur auf die Partitionen zu, die gewartet werden müssen.
- Wenn Sie große Datenmengen aufnehmen, sollten Sie die Funktion zur automatischen Analyse deaktivieren.

Die Auto-Vakuum-Funktion wird automatisch für eine Tabelle ausgelöst, wenn die folgende Formel zutrifft.

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

Die Ansicht `pg_stat_user_tables` und der Katalog `pg_class` haben mehrere Zeilen. Eine Zeile kann einer Zeile in Ihrer Tabelle entsprechen. Diese Formel geht davon aus, dass die `reltuples` für eine bestimmte Tabelle stehen. Die Parameter `autovacuum_vacuum_scale_factor` (standardmäßig 0.20) und `autovacuum_vacuum_threshold` (standardmäßig 50 Tupel) werden normalerweise global für die gesamte Instance gesetzt. Sie können jedoch verschiedene Werte für eine bestimmte Tabelle festlegen.

Themen

- Suchen Sie nach Tabellen, die unnötigerweise Speicherplatz belegen.
- Finden Sie Indizes mit unnötigem Speicherplatz
- Finden Sie Tabellen, die für die automatische Vakuumierung berechtigt sind

Suchen Sie nach Tabellen, die unnötigerweise Speicherplatz belegen.

Um Tabellen zu finden, die mehr Speicherplatz belegen als nötig, führen Sie die folgende Abfrage aus. Wenn diese Abfrage von einer Datenbankbenutzerrolle ausgeführt wird, die nicht über die `rds_superuser`-Rolle verfügt, gibt sie nur Informationen zu den Tabellen zurück, für die die Benutzerrolle Leserechte besitzt. Diese Abfrage wird von PostgreSQL Version 12 und späteren Versionen unterstützt.

```
WITH report AS (
   SELECT   schemaname
           ,tblname
           ,n_dead_tup
           ,n_live_tup
           ,block_size*tblpages AS real_size
           ,(tblpages-est_tblpages)*block_size AS extra_size
           ,CASE WHEN tblpages - est_tblpages > 0
              THEN 100 * (tblpages - est_tblpages)/tblpages::float
              ELSE 0
            END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*block_size AS
 bloat_size
           ,CASE WHEN tblpages - est_tblpages_ff > 0
              THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
              ELSE 0
            END AS bloat_ratio
           ,is_na
    FROM (
          SELECT  ceil( reltuples / ( (block_size-page_hdr)/tpl_size ) ) +
 ceil( toasttuples / 4 ) AS est_tblpages
                 ,ceil( reltuples / ( (block_size-page_hdr)*fillfactor/
(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff
                 ,tblpages
                 ,fillfactor
                 ,block_size
                 ,tblid
                 ,schemaname
                 ,tblname
```

```
                  ,n_dead_tup
                  ,n_live_tup
                  ,heappages
                  ,toastpages
                  ,is_na
              FROM (
                    SELECT ( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
                               - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE
tpl_hdr_size%ma END
                               - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
ceil(tpl_data_size)::int%ma END
                           ) AS tpl_size
                           ,block_size - page_hdr AS size_per_block
                           ,(heappages + toastpages) AS tblpages
                           ,heappages
                           ,toastpages
                           ,reltuples
                           ,toasttuples
                           ,block_size
                           ,page_hdr
                           ,tblid
                           ,schemaname
                           ,tblname
                           ,fillfactor
                           ,is_na
                           ,n_dead_tup
                           ,n_live_tup
                         FROM (
                              SELECT  tbl.oid                       AS tblid
                                     ,ns.nspname                    AS schemaname
                                     ,tbl.relname                   AS tblname
                                     ,tbl.reltuples                 AS reltuples
                                     ,tbl.relpages                  AS heappages
                                     ,coalesce(toast.relpages, 0)   AS toastpages
                                     ,coalesce(toast.reltuples, 0)  AS toasttuples
                                     ,psat.n_dead_tup               AS n_dead_tup
                                     ,psat.n_live_tup               AS n_live_tup
                                     ,24                            AS page_hdr
                                     ,current_setting('block_size')::numeric AS
block_size

,coalesce(substring( array_to_string(tbl.reloptions, ' ') FROM
'fillfactor=([0-9]+)')::smallint, 100) AS fillfactor
```

```
                                        ,CASE WHEN version()~'mingw32' OR version()~'64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END        AS ma
                                        ,23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0
 THEN ( 7 + count(*) ) / 8 ELSE 0::int END              AS tpl_hdr_size
                                        ,sum( (1-coalesce(s.null_frac, 0)) *
 coalesce(s.avg_width, 1024) )                                AS tpl_data_size
                                        ,bool_or(att.atttypid =
 'pg_catalog.name'::regtype) OR count(att.attname) <> count(s.attname)         AS is_na
                                    FROM  pg_attribute       AS att
                                    JOIN  pg_class           AS tbl    ON (att.attrelid =
 tbl.oid)
                                    JOIN  pg_stat_all_tables AS psat   ON (tbl.oid =
 psat.relid)
                                    JOIN  pg_namespace       AS ns     ON (ns.oid =
 tbl.relnamespace)
                                LEFT JOIN  pg_stats          AS s      ON
 (s.schemaname=ns.nspname AND s.tablename = tbl.relname AND s.inherited=false AND
 s.attname=att.attname)
                                LEFT JOIN  pg_class          AS toast  ON
 (tbl.reltoastrelid = toast.oid)
                                    WHERE  att.attnum > 0
                                      AND  NOT att.attisdropped
                                      AND  tbl.relkind = 'r'
                                 GROUP BY  tbl.oid, ns.nspname, tbl.relname,
 tbl.reltuples, tbl.relpages, toastpages, toasttuples, fillfactor, block_size, ma,
 n_dead_tup, n_live_tup
                                 ORDER BY  schemaname, tblname
                               ) AS s
                  ) AS s2
       ) AS s3
 ORDER BY bloat_size DESC
)
  SELECT *
    FROM report
   WHERE bloat_ratio != 0
 -- AND schemaname = 'public'
 -- AND tblname = 'pgbench_accounts'
;

-- WHERE NOT is_na
--   AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1
```

Sie können in Ihrer Anwendung nach einer Überlastung von Tabellen und Indizes suchen. Weitere Informationen finden Sie unter Diagnostizieren einer Überlastung von Tabellen und Indizes.

Finden Sie Indizes mit unnötigem Speicherplatz

Um Indizes zu finden, die unnötigen Speicherplatz benötigen, führen Sie die folgende Abfrage aus.

```
-- WARNING: run with a nonsuperuser role, the query inspects
-- only indexes on tables you have permissions to read.
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.

SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
```

```
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
      )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
      -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
    FROM (
      SELECT
        i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
AS table_oid,
        current_setting('block_size')::numeric AS bs, fillfactor,
        CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
          WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
          ELSE 4
        END AS maxalign,
        /* per page header, fixed size: 20 for 7.X, 24 for others */
        24 AS pagehdr,
        /* per page btree opaque data */
        16 AS pageopqdata,
        /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
        CASE WHEN max(coalesce(s.null_frac,0)) = 0
          THEN 2 -- IndexTupleData size
          ELSE 2 + (( 32 + 8 - 1 ) / 8)
          -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
index + 8 - 1 /8)
        END AS index_tuple_hdr_bm,
        /* data len: we remove null values save space using it fractionnal part from
stats */
        sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
        max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
AS is_na
      FROM pg_attribute AS a
        JOIN (
          SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
            idx.reltuples, idx.relpages, idx.relam,
            indrelid, indexrelid, indkey::smallint[] AS attnum,
            coalesce(substring(
              array_to_string(idx.reloptions, ' ')
```

```
               from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
          FROM pg_index
            JOIN pg_class idx ON idx.oid=pg_index.indexrelid
            JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
            JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
          WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
        ) AS i ON a.attrelid = i.indexrelid
        JOIN pg_stats AS s ON s.schemaname = i.nspname
          AND ((s.tablename = i.tblname AND s.attname =
 pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
          -- stats from tbl
          OR  (s.tablename = i.idxname AND s.attname = a.attname))
          -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

Finden Sie Tabellen, die für die automatische Vakuumierung berechtigt sind

Führen Sie die folgende Abfrage aus, um Tabellen zu finden, die für die automatische Vakuumierung berechtigt sind.

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
```

```
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
         coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
         as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

Reagieren Sie auf eine hohe Anzahl von Verbindungen

Wenn Sie Amazon CloudWatch überwachen, stellen Sie möglicherweise fest, dass die `DatabaseConnections`-Metrik ansteigt. Dieser Anstieg deutet auf eine erhöhte Anzahl von Verbindungen zu Ihrer Datenbank hin. Wir empfehlen folgende Vorgehensweise:

- Beschränken Sie die Anzahl der Verbindungen, die die Anwendung mit jeder Instance öffnen kann. Wenn Ihre Anwendung über eine eingebettete Verbindungspool-Funktion verfügt, legen Sie eine angemessene Anzahl von Verbindungen fest. Basieren Sie die Zahl darauf, was die vCPUs in Ihrer Instance effektiv parallelisieren können.

Wenn Ihre Anwendung keine Verbindungspool-Funktion verwendet, sollten Sie den Amazon RDS Proxy oder eine Alternative in Betracht ziehen. Mit diesem Ansatz können Ihre Anwendung mehrere Verbindungen mit dem Load Balancer öffnen. Der Balancer kann dann eine begrenzte Anzahl von Verbindungen mit der Datenbank öffnen. Da weniger Verbindungen parallel laufen, führt Ihre DB-Instance weniger Kontextwechsel im Kernel durch. Abfragen sollten schneller voranschreiten und zu weniger Warteereignissen führen. Weitere Informationen finden Sie unter Amazon-RDS-Proxy für Aurora.

- Nutzen Sie nach Möglichkeit die Reader-Knoten für Aurora PostgreSQL und lesen Sie Replikate für RDS für PostgreSQL. Wenn Ihre Anwendung einen schreibgeschützten Vorgang ausführt, senden Sie diese Anfragen an den reader-Endpunkt. Diese Technik verbreitet Anwendungsanfragen auf alle Reader-Knoten, wodurch der I/O-Druck auf den Writer-Knoten reduziert wird.
- Ziehen Sie, Ihre DB-Instance zu skalieren. Eine Instance-Klasse mit höherer Kapazität bietet mehr Speicher, was Aurora PostgreSQL einen größeren freigegebenen Pufferpool zum Speichern von Seiten gibt. Die größere Größe gibt der DB-Instance auch mehr vCPUs für die Handhabung von Verbindungen. Mehr vCPUs sind besonders hilfreich, wenn die Vorgänge, die `IO:DataFileRead`-Warteereignisse generieren, Schreibvorgänge sind.

IO:XactSync

Das `IO:XactSync` Ereignis tritt ein, wenn die Datenbank darauf wartet, dass das Aurora-Storage-Subsystem den Commit einer regulären Transaktion bzw. den Commit oder Rollback einer vorbereiteten Transaktion ausführt. Eine vorbereitete Transaktion ist Teil des Supports von PostgreSQL für einen zweiphasigen Commit. Dieses Ereignis kann auch auftreten, wenn eine Abfrage darauf wartet, dass eine andere Transaktion übergeben wird, insbesondere in Fällen, in denen Auto-Commit deaktiviert ist. In solchen Szenarien kann es vorkommen, dass Updates scheinbar auf XactSync warten, obwohl sie noch nicht übergeben wurden.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Das Ereignis `IO:XactSync` zeigt an, dass die Instance-Zeit damit verbringt, darauf zu warten, dass das Aurora-Speichersubsystem bestätigt, dass Transaktionsdaten verarbeitet wurden.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `IO:XactSync`-Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

Netzwerksättigung

Der Datenverkehr zwischen Clients und der DB-Instance oder der Datenverkehr zum Speicher-Subsystem ist möglicherweise zu hoch für die Netzwerkbandbreite.

CPU-Druck

Eine hohe Workload könnte verhindern, dass der Aurora-Speicher-Daemon genügend CPU-Zeit erhält.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Überwachen Sie Ihre Ressourcen
- Hochskalieren Sie die CPU
- Erhöhung der Netzwerkbandbreite
- Reduzieren Sie die Anzahl der Commits

Überwachen Sie Ihre Ressourcen

Um die Ursache der erhöhten `IO:XactSync`-Ereignisse zu ermitteln, überprüfen Sie die folgenden Metriken:

- `WriteThroughput` und `CommitThroughput` – Änderungen des Schreibdurchsatzes oder des Commit-Durchsatzes können eine Zunahme der Workload anzeigen.
- `WriteLatency` und `CommitLatency` – Änderungen der Schreib- oder Commit-Latenz können darauf hinweisen, dass das Speichersubsystem zu mehr Arbeit aufgefordert wird.
- `CPUUtilization` – Wenn die CPU-Auslastung der Instance über 90 Prozent liegt, erhält der Aurora-Speicher-Daemon möglicherweise nicht genügend Zeit auf der CPU. In diesem Fall verschlechtert sich die I/O-Leistung.

Informationen zu diesen Metriken finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

Hochskalieren Sie die CPU

Um Probleme mit CPU-Hunger zu beheben, sollten Sie in Betracht ziehen, zu einem Instance-Typ mit mehr CPU-Kapazität zu wechseln. Informationen zur CPU-Kapazität für eine DB-Instance-Klasse finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

Erhöhung der Netzwerkbandbreite

Um festzustellen, ob die Instance ihre Netzwerkbandbreitenlimits erreicht, suchen Sie nach den folgenden anderen Warteereignissen:

- `IO:DataFileRead`, `IO:BufferRead`, `IO:BufferWrite` und `IO:XactWrite` – Abfragen, die große Mengen an I/O verwenden, können mehr dieser Warteereignisse generieren.
- `Client:ClientRead` und`Client:ClientWrite` – Abfragen mit großen Mengen an Client-Kommunikation können mehr dieser Warteereignisse generieren.

Wenn die Netzwerkbandbreite ein Problem darstellt, sollten Sie erwägen, zu einem Instance-Typ mit mehr Netzwerkbandbreite zu wechseln. Informationen zur Netzwerkleistung für eine DB-Instance-Klasse finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

Reduzieren Sie die Anzahl der Commits

Um die Anzahl der Commits zu reduzieren, kombinieren Sie Anweisungen in Transaktionsblöcken.

IPC:DamRecordTxAck

Das `IPC:DamRecordTxAck`-Ereignis tritt auf, wenn Aurora PostgreSQL in einer Sitzung, die Datenbank-Aktivitätsstreams verwendet, ein Aktivitätsstream-Ereignis generiert und dann wartet, bis dieses Ereignis dauerhaft wird.

Themen

- Relevante Engine-Versionen
- Kontext
- Ursachen
- Aktionen

Relevante Engine-Versionen

Diese Warteereignisinformationen sind für alle Aurora PostgreSQL 10.7 und höher 10 Versionen, 11.4 und höher 11 Versionen sowie alle 12 und 13 Versionen relevant.

Kontext

Im synchronen Modus wird die Dauerhaftigkeit von Aktivitätsstream-Ereignissen gegenüber der Datenbankleistung bevorzugt. Während auf ein dauerhaftes Schreiben des Ereignisses gewartet wird, blockiert die Sitzung andere Datenbankaktivitäten, was das `IPC:DamRecordTxAck`-Warteereignis verursacht.

Ursachen

Die häufigste Ursache für das Auftreten des `IPC:DamRecordTxAck`-Ereignisses in Top-Waits ist, dass die Funktion Database Activity Streams (DAS) ein ganzheitliches Audit ist. Eine höhere SQL-Aktivität generiert Aktivitätsstream-Ereignisse, die aufgezeichnet werden müssen.

Aktionen

Abhängig von den Ursachen Ihres Wait-Ereignisses empfehlen wir verschiedene Aktionen:

- Reduzieren Sie die Anzahl der SQL-Anweisungen oder deaktivieren Sie Datenbank-Aktivitätsströme. Dadurch wird die Anzahl der Ereignisse reduziert, die dauerhafte Schreibvorgänge erfordern.
- Wechseln Sie in den asynchronen Modus. Dadurch können Konflikte beim `IPC:DamRecordTxAck`-Warteereignis reduziert werden.

  Die DAS-Funktion kann jedoch nicht die Haltbarkeit jedes Ereignisses im asynchronen Modus garantieren.

IPC:parallel-Warteereignisse

Die folgenden `IPC:parallel wait events` weisen darauf hin, dass eine Sitzung auf die Interprozesskommunikation in Bezug auf parallele Abfrageausführungen wartet.

- `IPC:BgWorkerStartup` - Ein Prozess wartet darauf, dass ein paralleler Worker-Prozess seine Startsequenz abgeschlossen hat. Dies geschieht, wenn Worker für eine parallele Abfrageausführung initialisiert werden.
- `IPC:BgWorkerShutdown` - Ein Prozess wartet darauf, dass ein paralleler Worker-Prozess seine Shutdown-Sequenz abgeschlossen hat. Dies geschieht während der Bereinigungsphase der parallelen Abfrageausführung.
- `IPC:ExecuteGather` - Ein Prozess wartet darauf, während der Abfrageausführung Daten von parallelen Worker-Prozessen zu empfangen. Dies ist der Fall, wenn der Leader-Prozess Ergebnisse von seinen Workern erfassen muss.
- `IPC:ParallelFinish` - Ein Prozess wartet darauf, dass parallele Worker ihre Ausführung beenden und ihre Endergebnisse melden. Dies geschieht während der Abschlussphase der parallelen Abfrageausführung.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Bei der parallelen Abfrageausführung in PostgreSQL arbeiten mehrere Prozesse zusammen, um eine einzelne Abfrage zu verarbeiten. Wenn festgestellt wird, dass eine Abfrage für die Parallelisierung geeignet ist, koordiniert ein Leader-Prozess die Arbeit mit einem oder mehreren parallelen Worker-Prozessen je nach `max_parallel_workers_per_gather`-Parametereinstellung. Der Leader-Prozess teilt die Arbeit auf die einzelnen Worker auf und jeder Worker verarbeitet unabhängig seinen Teil der Daten. Die Ergebnisse werden dann an den Leader-Prozess zurückgemeldet.

> **Note**
>
> Jeder parallele Worker fungiert als separater Prozess mit Ressourcenanforderungen, die einer vollständigen Benutzersitzung ähneln. Das bedeutet, dass eine parallele Abfrage mit 4 Workern bis zu fünfmal so viele Ressourcen (CPU, Arbeitsspeicher, I/O-Bandbreite) verbrauchen kann wie eine nicht parallele Abfrage, da sowohl der Leader-Prozess als auch die einzelnen Worker-Prozesse ihre eigenen Ressourcenzuweisungen verwalten. So werden beispielsweise Einstellungen wie `work_mem` individuell auf jeden Worker angewendet, wodurch sich die gesamte Speicherbelegung aller Prozesse vervielfachen kann.

Die Architektur einer parallelen Abfrage besteht aus drei Hauptkomponenten:

- Leader-Prozess: Dies ist der Hauptprozess, der den Parallelbetrieb einleitet, den Workload aufteilt und die Abstimmung mit den Worker-Prozessen vornimmt.
- Worker-Prozesse: Dies sind Hintergrundprozesse, die Teile der Abfrage parallel ausführen.
- Gather/Gather merge: Dies sind Operationen, die Ergebnisse mehrerer Worker-Prozesse kombinieren und zurück an den Leader melden.

Während der parallelen Ausführung müssen Prozesse über Mechanismen der Interprozesskommunikation (IPC) miteinander kommunizieren. Diese IPC-Warteereignisse treten in verschiedenen Phasen auf:

- Worker-Start: wenn parallele Worker initialisiert werden
- Datenaustausch: wenn Worker Daten verarbeiten und Ergebnisse an den Leader senden
- Worker-Shutdown: wenn die parallele Ausführung abgeschlossen ist und Worker beendet werden
- Synchronisierungspunkte: wenn Prozesse koordiniert werden oder darauf warten müssen, dass andere Prozesse ihre Aufgaben abschließen

Das Kennen dieser Warteereignisse ist entscheidend für die Diagnose von Leistungsproblemen im Zusammenhang mit der parallelen Abfrageausführung, insbesondere in Umgebungen mit hoher Nichtsequentialität, in denen mehrere parallele Abfragen gleichzeitig ausgeführt werden.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Verschiedene Faktoren können zu einer Zunahme parallel-bezogener IPC-Warteereignisse beitragen:

Hohe Nichtsequentialität paralleler Abfragen

Wenn viele parallele Abfragen gleichzeitig ausgeführt werden, kann dies zu Ressourcenkonflikten und längeren Wartezeiten für IPC-Operationen führen. Dies ist besonders häufig bei Systemen mit hohem Transaktionsvolumen oder analytischen Workloads der Fall.

Suboptimale Pläne für parallele Abfragen

Wenn der Abfrageplaner ineffiziente Parallelisierungspläne auswählt, kann dies zu unnötiger Parallelisierung oder schlechter Arbeitsverteilung unter den Workern führen. Dies kann längere IPC-Wartezeiten zur Folge haben, insbesondere bei den Ereignissen IPC:ExecuteGather und IPC:ParallelFinish. Diese Planungsprobleme sind häufig auf veraltete Statistiken und „aufgeblähte“ Tabellen und Indizes zurückzuführen.

Häufiges Starten und Herunterfahren paralleler Worker

Kurzlebige Abfragen, die häufig parallele Worker initiieren und beenden, können zu einer Zunahme von `IPC:BgWorkerStartup`- und `IPC:BgWorkerShutdown`-Ereignissen führen. Dies kommt oftmals bei OLTP-Workloads mit vielen kleinen, parallelisierbaren Abfragen vor.

Einschränkungen für Ressourcen

Eine begrenzte CPU-, Arbeitsspeicher- oder I/O-Kapazität kann zu Engpässen bei der parallelen Ausführung führen, was zu längeren Wartezeiten bei den IPC-Ereignissen führt. Wenn beispielsweise die CPU ausgelastet ist, kann es länger dauern, bis Worker-Prozesse starten oder ihren Teil der Arbeit erledigen.

Komplexe Abfragestrukturen

Abfragen mit mehreren Parallelitätsebenen (z. B. parallele Joins, auf die parallele Aggregationen folgen) können zu komplexeren IPC-Mustern und potenziell längeren Wartezeiten führen, insbesondere bei `IPC:ExecuteGather`-Ereignissen.

Große Ergebnismengen

Abfragen, die große Ergebnismengen liefern, können zu längeren `IPC:ExecuteGather`-Wartezeiten führen, da der Leader-Prozess mehr Zeit damit verbringt, Ergebnisse von Worker-Prozessen zu erfassen und zu verarbeiten.

Das Kennen dieser Faktoren kann bei der Diagnose und Behebung von Leistungsproblemen im Zusammenhang mit der parallelen Abfrageausführung in Aurora PostgreSQL hilfreich sein.

Aktionen

Wenn Sie Wartezeiten im Zusammenhang mit parallelen Abfragen bemerken, bedeutet dies in der Regel, dass ein Backend-Prozess parallele Worker-Prozesse koordiniert oder auf diese wartet. Diese Wartezeiten treten häufig bei der Ausführung von Parallelisierungsplänen auf. Sie können die Auswirkungen dieser Wartezeiten untersuchen und abmildern, indem Sie die Verwendung paralleler Worker überwachen, die Parametereinstellungen überprüfen und die Abfrageausführung und Ressourcenzuweisung optimieren.

Themen

- Analysieren von Abfrageplänen in Bezug auf ineffiziente Parallelität
- Überwachen der Verwendung paralleler Abfragen
- Überprüfen und Anpassen von Einstellungen für parallele Abfragen
- Optimieren der Ressourcenzuweisung
- Untersuchen der Verbindungsverwaltung
- Überprüfen und Optimieren von Wartungsvorgängen
- Verwenden der Abfrageplanverwaltung (QPM)

Analysieren von Abfrageplänen in Bezug auf ineffiziente Parallelität

Die parallele Ausführung von Abfragen kann häufig zu Systeminstabilität, CPU-Spitzen und unvorhersehbaren Leistungsschwankungen bei Abfragen führen. Es ist wichtig, gründlich zu analysieren, ob eine Parallelität Ihrem spezifischen Workload tatsächlich zugutekommt. Verwenden Sie EXPLAIN ANALYZE, um Pläne für die parallele Ausführung von Abfragen zu überprüfen.

Deaktivieren Sie zur Ermittlung der Planeffizienz vorübergehend die Parallelität auf Sitzungsebene:

```
SET max_parallel_workers_per_gather = 0;
EXPLAIN ANALYZE <your_query>;
```

Aktivieren Sie zwecks Vergleich wieder die Parallelität:

```
RESET max_parallel_workers_per_gather;
EXPLAIN ANALYZE <your_query>;
```

Wenn die Deaktivierung der Parallelität zu besseren oder konsistenteren Ergebnissen führt, sollten Sie erwägen, sie für bestimmte Abfragen auf Sitzungsebene mithilfe von SET-Befehlen zu deaktivieren. Um eine umfassendere Wirkung zu erzielen, können Sie die Parallelität auf Instance-Ebene deaktivieren, indem Sie die entsprechenden Parameter in Ihrem Cluster oder Ihrer Instance-Parametergruppe anpassen. Weitere Informationen finden Sie unter Amazon-Aurora-PostgreSQL-Parameter.

Überwachen der Verwendung paralleler Abfragen

Verwenden Sie die folgenden Abfragen, um einen Einblick in die Aktivität und Kapazität paralleler Abfragen zu gewinnen:

Überprüfen Sie die aktiven, parallelen Worker-Prozesse:

```
SELECT
    COUNT(*)
FROM
    pg_stat_activity
WHERE
    backend_type = 'parallel worker';
```

Diese Abfrage zeigt die Anzahl der aktiven, parallelen Worker-Prozesse. Ein hoher Wert kann darauf hinweisen, dass Ihr „max_parallel_workers“ mit einem hohen Wert konfiguriert ist. Erwägen Sie, diesen zu verringern.

Überprüfen Sie gleichzeitige parallele Abfragen:

```
SELECT
    COUNT(DISTINCT leader_pid)
FROM
    pg_stat_activity
WHERE
```

```
    leader_pid IS NOT NULL;
```

Diese Abfrage gibt die Anzahl der verschiedenen Leader-Prozesse zurück, durch die parallele Abfragen gestartet wurden. Eine hohe Zahl weist hier darauf hin, dass mehrere Sitzungen gleichzeitig parallele Abfragen ausführen, was die CPU- und Speicherauslastung erhöhen kann.

Überprüfen und Anpassen von Einstellungen für parallele Abfragen

Überprüfen Sie die folgenden Parameter, um sicherzustellen, dass sie Ihrem Workload entsprechen:

- `max_parallel_workers`: Gesamtzahl der parallelen Worker in allen Sitzungen
- `max_parallel_workers_per_gather`: Max. Worker-Anzahl pro Abfrage

Bei OLAP-Workloads kann eine Erhöhung dieser Werte die Leistung verbessern. Für OLTP-Workloads werden im Allgemeinen niedrigere Werte bevorzugt.

```
SHOW max_parallel_workers;
SHOW max_parallel_workers_per_gather;
```

Optimieren der Ressourcenzuweisung

Überwachen Sie die CPU-Auslastung und ziehen Sie eine Anpassung der Anzahl der vCPUs in Betracht, wenn diese konstant hoch ist und Ihre Anwendung von parallelen Abfragen profitiert. Stellen Sie sicher, dass ausreichend Speicher für parallele Operationen verfügbar ist.

- Ermitteln Sie anhand von Performance Insights, ob das System CPU-gebunden ist.
- Jeder parallele Worker verwendet seinen eigenen `work_mem`. Stellen Sie sicher, dass die Gesamtspeichernutzung innerhalb der Instance-Grenzen liegt.

Parallele Abfragen können erheblich mehr Ressourcen verbrauchen als nicht parallele, da jeder Worker-Prozess ein vollständig separater Prozess ist, der ungefähr die gleichen Auswirkungen auf das System hat wie eine zusätzliche Benutzersitzung. Dies sollte bei der Auswahl eines Werts für diese Einstellung sowie bei der Konfiguration anderer Einstellungen zur Steuerung der Ressourcennutzung berücksichtigt werden, zum Beispiel `work_mem`. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation. Ressourcenbegrenzungen, wie `work_mem` werden individuell

auf jeden Worker angewendet, das bedeutet, dass die Gesamtauslastung über alle Prozesse hinweg viel höher sein kann, als dies normalerweise für einen einzelnen Prozess der Fall wäre.

Erwägen Sie, die vCPUs zu erhöhen oder die Speicherparameter zu optimieren, wenn Ihr Workload stark parallelisiert ist.

Untersuchen der Verbindungsverwaltung

Wenn die Verbindung überlastet ist, prüfen Sie Strategien für ein Verbindungspooling der Anwendung. Erwägen Sie die Implementierung von Verbindungspooling auf Anwendungsebene, sofern es nicht bereits verwendet wird.

Überprüfen und Optimieren von Wartungsvorgängen

Koordinieren Sie die Indexerstellung und andere Wartungsaufgaben, um Ressourcenkonflikte zu vermeiden. Erwägen Sie, diese Vorgänge für Zeiten außerhalb der Spitzenzeiten einzuplanen. Vermeiden Sie es, umfangreiche Wartungsvorgänge (z. B. parallele Indexerstellungen) für Zeiten mit hoher Benutzerabfragelast einzuplanen. Diese Operationen können parallele Worker beanspruchen und die Leistung bei regulären Abfragen negativ beeinträchtigen.

Verwenden der Abfrageplanverwaltung (QPM)

In Amazon Aurora PostgreSQL ist das Abfrageplanverwaltung (QPM)-Feature so ausgelegt, dass die Anpassungsfähigkeit und Stabilität des Plans unabhängig von Änderungen der Datenbankumgebung, die zu einer Regression des Abfrageplans führen könnten, sichergestellt ist. Weitere Informationen finden Sie unter Übersicht über die Abfrageplanverwaltung in Aurora PostgreSQL. QPM bietet eine gewisse Kontrolle über den Optimierer. Überprüfen Sie die genehmigten Pläne in QPM, um sicherzustellen, dass sie den aktuellen Parallelitätseinstellungen entsprechen. Aktualisieren oder entfernen Sie veraltete Pläne, die möglicherweise eine suboptimale parallele Ausführung erzwingen.

Sie können die Pläne auch mit pg_hint_plan korrigieren. Weitere Informationen finden Sie unter Reparieren von Plänen mit pg_hint_plan. Sie können mit dem `Parallel`-Hinweis eine parallele Ausführung erzwingen. Weitere Informationen finden Sie in unter Hints for parallel plans.

IPC: ProcArrayGroupUpdate

Das `IPC:ProcArrayGroupUpdate` Ereignis tritt ein, wenn eine Sitzung darauf wartet, dass der Gruppenleiter den Transaktionsstatus am Ende des Vorgangs aktualisiert. Während PostgreSQL im Allgemeinen Warteereignisse vom Typ IPC mit parallel Abfrageoperationen verknüpft, ist dieses spezielle Wartungsereignis nicht spezifisch für parallel Abfragen.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Das Prozessarray verstehen — Das Prozessarray (proc) ist eine gemeinsam genutzte Speicherstruktur in PostgreSQL. Es enthält Informationen über alle laufenden Prozesse, einschließlich Transaktionsdetails. Während des Abschlusses der Transaktion (`COMMIT`oder`ROLLBACK`) ProcArray muss die Datei aktualisiert werden, um die Änderung widerzuspiegeln und die TransactionID aus dem Array zu löschen. Die Sitzung, die versucht, ihre Transaktion abzuschließen, muss eine exklusive Sperre für erhalten. ProcArray Dadurch wird verhindert, dass andere Prozesse gemeinsame oder exklusive Sperren für die Datei erhalten.

Mechanismus zur Gruppenaktualisierung — Wenn ein Back-End-Prozess bei der Ausführung eines COMMIT- oder ROLLBACK-Vorgangs ProcArrayLock im exklusiven Modus keinen Wert abrufen kann, aktualisiert er ein spezielles Feld mit dem Namen. ProcArrayGroupMember Dadurch wird die Transaktion der Liste der Sitzungen hinzugefügt, die beendet werden sollen. Dieser Backend-Prozess schläft dann, und die Zeit, in der er sich befindet, wird als ProcArrayGroupUpdate Warteereignis instrumentiert. Der erste Prozess im ProcArray procArrayGroup With-Member, der als Leader-Prozess bezeichnet wird, erfolgt ProcArrayLock im exklusiven Modus. Anschließend wird die Liste der Prozesse gelöscht, die auf das Löschen der GruppentransactionID warten. Sobald dieser Vorgang abgeschlossen ist, gibt der Leader alle Prozesse in dieser Liste frei ProcArrayLock und aktiviert sie, sodass sie darüber informiert werden, dass ihre Transaktion abgeschlossen ist.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Je mehr Prozesse ausgeführt werden, desto länger hält ein Leader an einem procArrayLock im exklusiven Modus fest. Je mehr Schreibtransaktionen in einem Gruppenaktualisierungsszenario enden, was zu einer potenziellen Anhäufung von Prozessen führt, die auf das Warteereignis `ProcArrayGroupUpdate` warten. In der Top-SQL-Ansicht von Database Insights werden Sie

sehen, dass COMMIT die Anweisung ist, bei der der Großteil dieses Warteereignisses auftritt. Dies ist zu erwarten, erfordert jedoch eine eingehendere Untersuchung der spezifischen Schreib-SQL, die gerade ausgeführt wird, um festzustellen, welche geeigneten Maßnahmen zu ergreifen sind.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen. Identifizieren Sie `IPC:ProcArrayGroupUpdate` Ereignisse mithilfe von Amazon RDS Performance Insights oder durch Abfragen der PostgreSQL-Systemansicht. `pg_stat_activity`

Themen

- Überwachung von Transaktions-, Commit- und Rollback-Vorgängen
- Reduzierung der Parallelität
- Implementierung von Verbindungspooling

Überwachung von Transaktions-, Commit- und Rollback-Vorgängen

Überwachen Sie Commits und Rollbacks — Eine erhöhte Anzahl von Commits und Rollbacks kann zu einem erhöhten Druck auf die führen. ProcArray Wenn beispielsweise eine SQL-Anweisung aufgrund einer erhöhten Anzahl von Verletzungen doppelter Schlüssel zu scheitern beginnt, kann es zu einer Zunahme von Rollbacks kommen, was wiederum zu Konflikten und zu einer Überlastung der Tabellen führen kann. ProcArray

Amazon RDS Database Insights stellt die PostgreSQL-Metriken `xact_rollback` bereit `xact_commit` und berichtet über die Anzahl der Commits und Rollbacks pro Sekunde.

Reduzierung der Parallelität

Batch-Transaktionen — Wenn möglich, sollten Operationen in Einzeltransaktionen gebündelt werden, um die Anzahl der Operationen zu reduzieren commit/rollback .

Parallelität einschränken — Reduzieren Sie die Anzahl der gleichzeitig aktiven Transaktionen, um Sperrkonflikte bei der zu vermeiden. ProcArray Es sind zwar einige Tests erforderlich, aber eine Reduzierung der Gesamtzahl gleichzeitiger Verbindungen kann Konflikte reduzieren und den Durchsatz aufrechterhalten.

Implementierung von Verbindungspooling

Verbindungspooling-Lösungen — Verwenden Sie Verbindungspooling, um Datenbankverbindungen effizient zu verwalten und so die Gesamtzahl der Backends und damit die Arbeitslast auf dem zu

reduzieren. ProcArray Es sind zwar einige Tests erforderlich, aber die Reduzierung der Gesamtzahl gleichzeitiger Verbindungen kann Konflikte reduzieren und den Durchsatz aufrechterhalten.

Weitere Informationen finden Sie unter Verbindungspooling für Aurora PostgreSQL.

Reduzieren Sie Verbindungsstürme — In ähnlicher Weise führt ein Muster, bei dem häufig Verbindungen hergestellt und beendet werden, zu zusätzlichem Druck auf die. ProcArray Durch die Reduzierung dieses Musters werden Konflikte insgesamt reduziert.

Lock:advisory

Das `Lock:advisory`-Ereignis tritt auf, wenn eine PostgreSQL-Anwendung eine Sperre verwendet, um Aktivitäten über mehrere Sitzungen hinweg zu koordinieren.

Themen

- Relevante Engine-Versionen
- Kontext
- Ursachen
- Aktionen

Relevante Engine-Versionen

Diese Warteereignisinformationen sind für Aurora PostgreSQL Versionen 9.6 und höher relevant.

Kontext

PostgreSQL-Beratungssperren sind Anwendungsebene, kooperative Sperren werden explizit durch den Anwendungscode des Benutzers gesperrt und freigeschaltet. Eine Anwendung kann PostgreSQL-Beratungssperren verwenden, um Aktivitäten über mehrere Sitzungen hinweg zu koordinieren. Im Gegensatz zu normalen Sperren auf Objekt- oder Zeilenebene hat die Anwendung die volle Kontrolle über die Lebensdauer des Schlosses. Weitere Informationen finden Sie unter Empfohlene Sperren in der PostgreSQL-Dokumentation.

Beratungssperren können vor dem Ende einer Transaktion freigegeben werden oder von einer Sitzung über Transaktionen hinweg gehalten werden. Dies gilt nicht für implizite, vom System erzwungene Sperren, wie z. B. eine zugriffsexklusive Sperre für eine Tabelle, die von einer `CREATE INDEX`-Anweisung abgerufen wird.

Eine Beschreibung der Funktionen zum Erlangen (Sperren) und Freigeben (Entsperren) von Advisory Locks finden Sie unter Advisory Lock Functions in der PostgreSQL-Dokumentation.

Advisory Locks werden zusätzlich zum regulären PostgreSQL-Locking-System implementiert und sind in der `pg_locks`-Systemansicht sichtbar.

Ursachen

Dieser Schlosstyp wird ausschließlich von einer Anwendung gesteuert, die ihn explizit verwendet. Beratungssperren, die für jede Zeile als Teil einer Abfrage erworben werden, können zu einem Anstieg der Sperren oder zu einem langfristigen Aufbau führen.

Diese Effekte treten auf, wenn die Abfrage so ausgeführt wird, dass Sperren für mehr Zeilen erwirbt, als von der Abfrage zurückgegeben werden. Die Anwendung muss schließlich jede Sperre freigeben, aber wenn Sperren für Zeilen erworben werden, die nicht zurückgegeben werden, kann die Anwendung nicht alle Sperren finden.

Das folgende Beispiel stammt aus Empfohlene Sperren in der PostgreSQL-Dokumentation.

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

In diesem Beispiel kann die `LIMIT`-Klausel die Ausgabe der Abfrage nur stoppen, nachdem die Zeilen bereits intern ausgewählt und ihre ID-Werte gesperrt wurden. Dies kann plötzlich passieren, wenn ein wachsendes Datenvolumen dazu führt, dass der Planer einen anderen Ausführungsplan auswählt, der während der Entwicklung nicht getestet wurde. Der Aufbau erfolgt in diesem Fall, weil die Anwendung `pg_advisory_unlock` explizit für jeden gesperrten ID-Wert aufruft. In diesem Fall kann es jedoch nicht die Sperren finden, die für Zeilen erworben wurden, die nicht zurückgegeben wurden. Da die Sperren auf Sitzungsebene erworben werden, werden sie am Ende der Transaktion nicht automatisch freigegeben.

Eine weitere mögliche Ursache für Spikes bei blockierten Sperrversuchen sind unbeabsichtigte Konflikte. In diesen Konflikten teilen sich nicht verwandte Teile der Anwendung versehentlich denselben Sperren-ID-Raum.

Aktionen

Überprüfen Sie die Anwendungsnutzung von Beratungssperren und Details, wo und wann im Anwendungsablauf jede Art von Beratungssperre erworben und freigegeben wird.

Stellen Sie fest, ob eine Sitzung zu viele Sperren erwirbt oder eine lang andauernde Sitzung keine Sperren früh genug freigibt, was zu einem langsamen Aufbau von Sperren führt. Sie können

einen langsamen Aufbau von Sperren auf Sitzungsebene korrigieren, indem Sie die Sitzung mit `pg_terminate_backend(pid)` beenden.

Ein Client, der auf eine Beratungssperre wartet, erscheint in `pg_stat_activity` mit `wait_event_type=Lock` und `wait_event=advisory`. Sie können bestimmte Sperrwerte erhalten, indem Sie die `pg_locks`-Systemansicht nach demselben `pid` abfragen und nach `locktype=advisory` und `granted=f` suchen.

Sie können dann die blockierende Sitzung identifizieren, indem Sie `pg_locks` nach derselben beratenden Sperre mit `granted=t` abfragen, wie im folgenden Beispiel gezeigt.

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
         concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
 blocked_wait_event,
         concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
 blocking_wait_event,
         blocked_activity.state AS blocked_state,
         blocking_activity.state AS blocking_state,
         blocked_locks.locktype AS blocked_locktype,
         blocking_locks.locktype AS blocking_locktype,
         blocked_activity.query AS blocked_statement,
         blocking_activity.query AS blocking_statement
    FROM pg_catalog.pg_locks blocked_locks
    JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
 blocked_locks.pid
    JOIN pg_catalog.pg_locks blocking_locks
        ON blocking_locks.locktype = blocked_locks.locktype
        AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
        AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
        AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
        AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
        AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
        AND blocking_locks.transactionid IS NOT DISTINCT FROM
 blocked_locks.transactionid
        AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
        AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
        AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
        AND blocking_locks.pid != blocked_locks.pid
```

```
    JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
    WHERE NOT blocked_locks.GRANTED;
```

Alle API-Funktionen für beratende Sperren haben zwei Sätze von Argumenten, entweder ein `bigint`-Argument oder zwei `integer`-Argumente:

- Bei den API-Funktionen mit einem `bigint`-Argument befinden sich die oberen 32 Bit in `pg_locks.classid` und die unteren 32 Bit in `pg_locks.objid`.
- Bei den API-Funktionen mit zwei `integer`-Argumenten ist das erste Argument `pg_locks.classid` und das zweite Argument ist `pg_locks.objid`.

Der `pg_locks.objsubid`-Wert gibt an, welches API-Formular verwendet wurde: 1 bedeutet ein `bigint`-Argument; 2 bedeutet zwei `integer`-Argumente.

Lock:extend

Das `Lock:extend`-Ereignis tritt ein, wenn ein Backend-Prozess darauf wartet, eine Beziehung zu sperren, um sie zu erweitern, während ein anderer Prozess diese Beziehung für denselben Zweck gesperrt hat.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Das Ereignis `Lock:extend` zeigt an, dass ein Backend-Prozess darauf wartet, eine Beziehung zu erweitern, für die ein anderer Backend-Prozess eine Sperre hält, während er diese Beziehung erweitert. Da jeweils nur ein Prozess eine Beziehung erweitern kann, generiert das System ein

`Lock:extend`-Warteereignis. `INSERT`-, `COPY`- und `UPDATE`-Vorgänge können dieses Ereignis erzeugen.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `Lock:extend`-Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

Anstieg der gleichzeitigen Einfügungen oder Aktualisierungen derselben Tabelle

Es kann zu einer Zunahme der Anzahl gleichzeitiger Sitzungen mit Abfragen kommen, die in dieselbe Tabelle einfügen oder aktualisieren.

Unzureichende Netzwerkbandbreite

Die Netzwerkbandbreite auf der DB-Instance reicht möglicherweise nicht aus, um die Speicherkommunikationsanforderungen der aktuellen Workload zu gewährleisten. Dies kann zu einer Speicherlatenz führen, die zu einem Anstieg der `Lock:extend`-Ereignisse führt.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Reduzieren Sie gleichzeitige Einfügungen und Aktualisierungen auf dieselbe Beziehung
- Erhöhung der Netzwerkbandbreite

Reduzieren Sie gleichzeitige Einfügungen und Aktualisierungen auf dieselbe Beziehung

Stellen Sie zunächst fest, ob die `tup_inserted`- und `tup_updated`-Metriken und damit auch dieses Warteereignis gestiegen ist. Überprüfen Sie in diesem Fall, welche Beziehungen für Einfüge- und Aktualisierungsvorgänge in hohem Streit stehen. Um dies zu ermitteln, fragen Sie die `pg_stat_all_tables`-Ansicht nach den Werten in den `n_tup_ins`- und `n_tup_upd`-Feldern ab. Informationen zur Ansicht `pg_stat_all_tables` finden Sie unter pg_stat_all_tables in der PostgreSQL-Dokumentation.

Um weitere Informationen über das Blockieren und blockierte Abfragen zu erhalten, fragen Sie `pg_stat_activity` wie im folgenden Beispiel ab:

```
SELECT
```

```
    blocked.pid,
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |            query             | blocking_id |
           blocking_query                          | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+-----------------------------------------------------------------
+---------------------+-------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

Nachdem Sie Beziehungen identifiziert haben, die zur Erhöhung von `Lock:extend`-Ereignissen beitragen, verwenden Sie die folgenden Techniken, um die Konflikte zu reduzieren:

- Finden Sie heraus, ob Sie Partitionierung verwenden können, um die Konflikte für dieselbe Tabelle zu reduzieren. Das Trennen eingefügter oder aktualisierter Tupel in verschiedene Partitionen kann die Konflikte verringern. Weitere Informationen zur Partitionierung finden Sie unter Verwalten von PostgreSQL-Partitionen mit der Erweiterung pg_partman.
- Wenn das Warteereignis hauptsächlich auf Aktualisierungsaktivitäten zurückzuführen ist, sollten Sie erwägen, den Füllfaktor-Wert der Beziehung zu reduzieren. Dies kann Anfragen nach neuen Blöcken während des Updates reduzieren. Der Füllfaktor ist ein Speicherparameter für eine Tabelle, der den maximalen Speicherplatz zum Packen einer Tabellenseite bestimmt. Es wird als Prozentsatz des gesamten Speicherplatzes für eine Seite ausgedrückt. Weitere Informationen zum Parameter fillfactor finden Sie unter CREATE TABLE in der PostgreSQL-Dokumentation.

> **Important**
>
> Wir empfehlen dringend, Ihr System zu testen, wenn Sie den Füllfaktor ändern, da sich die Änderung dieses Wertes je nach Workload negativ auf die Leistung auswirken kann.

Erhöhung der Netzwerkbandbreite

Um zu sehen, ob die Schreiblatenz zunimmt, überprüfen Sie die `WriteLatency`-Metrik in CloudWatch. Wenn ja, verwenden Sie die Amazon CloudWatch-Metriken von `WriteThroughput` und `ReadThroughput`, um den speicherbezogenen Datenverkehr auf dem DB-Cluster zu überwachen. Diese Metriken können Ihnen helfen festzustellen, ob die Netzwerkbandbreite für die Speicheraktivität Ihrer Workload ausreicht.

Wenn Ihre Netzwerkbandbreite nicht ausreicht, erhöhen Sie sie. Wenn Ihre DB-Instance die Grenzen der Netzwerkbandbreite erreicht, besteht die einzige Möglichkeit, die Bandbreite zu erhöhen, darin, die Größe Ihrer DB-Instance zu erhöhen.

Weitere Informationen zu CloudWatch-Metriken finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora. Informationen zur Netzwerkleistung für jede DB-Instance-Klasse finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

Lock:Relation

Das `Lock:Relation`-Ereignis tritt ein, wenn eine Abfrage darauf wartet, eine Sperre für eine Tabelle oder Sicht (Relation) zu erhalten, die derzeit von einer anderen Transaktion gesperrt ist.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Die meisten PostgreSQL-Befehle verwenden implizit Sperren, um den gleichzeitigen Zugriff auf Daten in Tabellen zu steuern. Sie können diese Sperren auch explizit mit dem `LOCK`-Befehl in Ihrem Anwendungscode verwenden. Viele Sperrmodi sind nicht miteinander kompatibel und können Transaktionen blockieren, wenn sie versuchen, auf dasselbe Objekt zuzugreifen. In diesem Fall

generiert Aurora PostgreSQL ein `Lock:Relation`-Ereignis. Einige gängige Beispiele sind die folgenden:

- Exklusive Sperren wie `ACCESS EXCLUSIVE` können alle gleichzeitigen Zugriffe blockieren. Vorgänge in der Datendefinitionssprache (DDL) wie `DROP TABLE`, `TRUNCATE`, `VACUUM FULL` und `CLUSTER` erwerben implizit `ACCESS EXCLUSIVE`-Sperren. `ACCESS EXCLUSIVE` ist auch der Standardsperrmodus für `LOCK TABLE`-Anweisungen, die keinen Modus explizit angeben.
- Die Verwendung von `CREATE INDEX (without CONCURRENT)` für eine Tabelle steht in Konflikt mit den DML-Anweisungen `UPDATE`, `DELETE` und `INSERT`, die `ROW EXCLUSIVE`-Sperren anfordern.

Weitere Informationen zu Sperren auf Tabellenebene und widersprüchlichen Sperrmodi finden Sie unter Explizite Sperren in der PostgreSQL-Dokumentation.

Blockieren von Abfragen und Transaktionen entsperren in der Regel auf eine der folgenden Arten:

- Blockierende Abfrage — Die Anwendung kann die Abfrage abbrechen oder der Benutzer kann den Prozess beenden. Die Engine kann die Abfrage auch aufgrund des Statement-Timeouts einer Sitzung oder eines Deadlock-Erkennungsmechanismus zum Ende zwingen.
- Blockieren einer Transaktion – Eine Transaktion stoppt die Blockierung, wenn sie eine `ROLLBACK`- oder `COMMIT`-Anweisung ausführt. Rollbacks erfolgen auch automatisch, wenn Sitzungen von einem Client oder durch Netzwerkprobleme getrennt oder beendet werden. Sitzungen können beendet werden, wenn das Datenbank-Engine heruntergefahren wird, wenn das System keinen Arbeitsspeicher mehr hat usw.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `Lock:Relation`-Ereignis häufiger als normal auftritt, kann dies auf ein Leistungsproblem hinweisen. Zu den typischen Ursachen zählen auch die Folgenden:

Gleichzeitige Sitzungen mit widersprüchlichen Tabellensperren erhöht

Es kann zu einer Zunahme der Anzahl gleichzeitiger Sitzungen mit Abfragen kommen, die dieselbe Tabelle mit widersprüchlichen Sperrmodi sperren.

Wartungsvorgänge

Zustandswartungsvorgänge wie `VACUUM` und `ANALYZE` können die Anzahl widersprüchlicher Sperren erheblich erhöhen. `VACUUM FULL` erhält eine `ACCESS EXCLUSIVE`-Sperre und

ANALYZE erhält eine SHARE UPDATE EXCLUSIVE-Sperre. Beide Arten von Sperren können ein Lock:Relation-Wait-Ereignis verursachen. Wartungsvorgänge für Anwendungsdaten wie das Aktualisieren einer materialisierten Ansicht können auch blockierte Abfragen und Transaktionen erhöhen.

Sperrt bei Reader-In

Es könnte ein Konflikt zwischen den Beziehungssperren bestehen, die vom Writer und den Readern gehalten werden. Derzeit werden nur ACCESS EXCLUSIVE-Beziehungssperren auf Reader-Instances repliziert. Allerdings gerät die ACCESS EXCLUSIVE-Beziehungssperre in Konflikt mit jeder ACCESS SHARE-Beziehungssperre, die vom Reader gehalten wird. Dies kann zu einer Erhöhung der Warteereignisse für die Sperrbeziehung des Readers führen.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Reduzieren Sie die Auswirkungen der Blockierung von SQL-Anweisungen
- Minimieren Sie die Auswirkungen von Wartungsvorgängen
- Auf Lesersperren überprüfen

Reduzieren Sie die Auswirkungen der Blockierung von SQL-Anweisungen

Um die Auswirkungen des Blockierens von SQL-Anweisungen zu reduzieren, ändern Sie Ihren Anwendungscode nach Möglichkeit. Es folgen zwei gängige Techniken zum Reduzieren von Blöcken:

- Verwenden Sie die Option NOWAIT – Einige SQL-Befehle, wie z. B. SELECT- und LOCK-Anweisungen, unterstützen diese Option. Die NOWAIT-Direktive bricht die Sperre anfordernde Abfrage ab, wenn die Sperre nicht sofort erworben werden kann. Diese Technik kann dazu beitragen, zu verhindern, dass eine Blockiersitzung eine Anhäufung blockierter Sitzungen dahinter verursacht.

  Beispiel: Angenommen, Transaktion A wartet auf eine Sperre, die von Transaktion B gehalten wird. Wenn B nun eine Sperre für eine Tabelle anfordert, die durch Transaktion C gesperrt ist, könnte Transaktion A blockiert werden, bis die Transaktion C abgeschlossen ist. Wenn Transaktion B jedoch ein NOWAIT verwendet, wenn sie die Sperre für C anfordert, kann sie schnell fehlschlagen und sicherstellen, dass Transaktion A nicht unbegrenzt warten muss.

- Verwenden Sie `SET lock_timeout` – Legen Sie einen `lock_timeout`-Wert fest, um die Zeit zu begrenzen, die eine SQL-Anweisung wartet, um eine Sperre für eine Beziehung zu erhalten. Wenn die Sperre nicht innerhalb des angegebenen Timeouts erworben wird, wird die Transaktion, die die Sperre anfordert, abgebrochen. Stellen Sie diesen Wert auf Sitzungsebene ein.

Minimieren Sie die Auswirkungen von Wartungsvorgängen

Wartungsvorgänge wie `VACUUM` und `ANALYZE` sind wichtig. Wir empfehlen, sie nicht zu deaktivieren, da Sie `Lock:Relation`-Warteereignisse im Zusammenhang mit diesen Wartungsvorgängen finden. Die folgenden Ansätze können die Auswirkungen dieser Vorgänge minimieren:

- Führen Sie Wartungsvorgänge während außerhalb der Hauptverkehrszeiten manuell aus.
- Um `Lock:Relation`-Wartezeiten zu reduzieren, die durch Autovacuum-Aufgaben verursacht werden, führen Sie alle erforderlichen Autovacuum-Optimierungen durch. Informationen zum Optimieren von Autovacuum finden Sie unter Arbeiten mit PostgreSQL Autovacuum auf Amazon RDS im Amazon-RDS-Benutzerhandbuch.

Auf Lesersperren überprüfen

Sie können sehen, wie gleichzeitige Sitzungen bei einem Autor und Lesern Sperren halten, die sich gegenseitig blockieren. Eine Möglichkeit dazu besteht darin, Abfragen auszuführen, die den Sperrtyp und die Beziehung zurückgeben. Die Tabelle enthält eine Abfolge von Abfragen für zwei dieser gleichzeitigen Sitzungen, eine Writer-Sitzung und eine Reader-Sitzung.

Der Wiedergabeprozess wartet die Dauer von `max_standby_streaming_delay` ab, bevor die Reader-Abfrage abgebrochen wird. Wie im Beispiel gezeigt, liegt das Sperr-Timeout von 100 ms deutlich unter dem standardmäßigen `max_standby_streaming_delay`-Wert von 30 Sekunden. Für die Sperre tritt ein Timeout auf, bevor ein Problem entsteht.

Sequenzereignis	Sitzung	Befehl oder Ausgabe
Dieses legt eine Umgebungsvariable namens READER mit dem angegebenen Wert fest und versucht, mit diesem Endpunkt eine Verbindung zur DB-Instance herzustellen.	Lesersitzung	CLI-Befehl: `export READER=aurorapg2.1234567891 0.us-west-1.rds.amazonaws.com`

Sequenzereignis	Sitzung	Befehl oder Ausgabe

		`psql -h $READER`
		
		Ausgabe:
		
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

Dieses legt eine Umgebungsvariable namens WRITER fest und versucht, mit diesem Endpunkt eine Verbindung zur DB-Instance herzustellen.	Writer session	CLI-Befehl:
		
		`export WRITER=aurorapg1.12345678910.us-west-1.rds.amazonaws.com`
		`psql -h $WRITER`
		
		Ausgabe:
		
		`psql (15devel, server 10.14)`
		`Type "help" for help.`

Die Schreibsitzung erstellt die Tabelle t1 auf der Writer-Instance.	Writer session	PostgreSQL-Abfrage:
		
		`postgres=> CREATE TABLE t1(b integer);`
		`CREATE TABLE`

Wenn es keine widersprüchlichen Abfragen für den Writer gibt, wird sofort die ACCESS EXCLUSIVE-Sperre für den Writer erworben.	Writer session	`ACCESS EXCLUSIVE` -Sperre aktiviert

Sequenzereignis	Sitzung	Befehl oder Ausgabe

Die Lesesitzung legt ein Sperr-Timeout-Intervall von 100 Millisekunden fest.	Lesersitzung	PostgreSQL-Abfrage:
		`postgres=> SET lock_timeout=100;`
		`SET`

Die Lesesitzung versucht, Daten aus der Tabelle t1 auf der Reader-Instance zu lesen.	Lesersitzung	PostgreSQL-Abfrage:
		`postgres=> SELECT * FROM t1;`
		Beispielausgabe:
		`b`
		`---`
		`(0 rows)`

Die Schreibsitzung verwirft t1.	Writer session	PostgreSQL-Abfrage:
		`postgres=> BEGIN;`
		`BEGIN`
		`postgres=> DROP TABLE t1;`
		`DROP TABLE`
		`postgres=>`

Sequenzereignis	Sitzung	Befehl oder Ausgabe

Die Abfrage ist abgelaufen und wird im Reader abgebrochen.	Lesersitzung	PostgreSQL-Abfrage:
		`postgres=>  SELECT * FROM t1;`
		Beispielausgabe:
		`ERROR:  canceling statement due to lock timeout`
		`LINE 1: SELECT * FROM t1;`
		`^`

Die Lesesitzung fragt `pg_locks` und `pg_stat_activity` ab, um die Fehlerursache zu ermitteln.	Lesersitzung	PostgreSQL-Abfrage:
		`postgres=>  SELECT locktype, relation, mode, backend_type`
		`postgres=>  FROM pg_locks l, pg_stat_activity t1`
		`postgres=>  WHERE l.pid=t1.pid AND relation = 't1'::regclass::oid;`

Sequenzereignis	Sitzung	Befehl oder Ausgabe

Das Ergebnis zeigt an, dass der `aurora wal replay`-Prozess eine `ACCESS EXCLUSIVE` -Sperre für Tabelle t1 hält.	Lesersitzung	Abfrageergebnis:
		`locktype | relation | mode | backend_type`
		`----------+----------+-------------------+---------------`
		`relation | 68628525 | AccessExclusiveLock | aurora wal replay`
		`(1 row)`

Lock:transactionid

Das `Lock:transactionid`-Ereignis tritt ein, wenn eine Transaktion auf eine Sperre auf Zeilenebene wartet.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Das Ereignis `Lock:transactionid` tritt ein, wenn eine Transaktion versucht, eine Sperre auf Zeilenebene zu erlangen, die bereits einer gleichzeitig laufenden Transaktion gewährt wurde. Die Sitzung, die das `Lock:transactionid`-Wait-Ereignis anzeigt, ist aufgrund dieser Sperre blockiert.

Nachdem die blockierende Transaktion entweder in einer COMMIT- oder ROLLBACK-Anweisung endet, kann die blockierte Transaktion fortgesetzt werden.

Die Semantik der Multiversions-Parallelität von Aurora PostgreSQL garantiert, dass Leser keine Autoren blockieren und Autoren Leser nicht blockieren. Damit Konflikte auf Zeilenebene auftreten können, müssen blockierende und blockierte Transaktionen widersprüchliche Anweisungen der folgenden Typen ausgeben:

- UPDATE
- SELECT … FOR UPDATE
- SELECT … FOR KEY SHARE

Die Anweisung SELECT … FOR KEY SHARE ist ein Sonderfall. Die Datenbank verwendet die Klausel FOR KEY SHARE, um die Leistung der referenziellen Integrität zu optimieren. Eine Sperre auf Zeilenebene für eine Zeile kann INSERT-, UPDATE- und DELETE-Befehle für andere Tabellen blockieren, die auf die Zeile verweisen.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn dieses Ereignis mehr als normal auftritt, sind die Ursache normalerweise UPDATE-, SELECT … FOR UPDATE-, SELECT … FOR KEY SHARE-Anweisungen in Kombination mit den folgenden Bedingungen.

Themen

- Hohe Gleichzeitigkeit
- Leerlauf in Transaktion
- Lang laufende Transaktionen

Hohe Gleichzeitigkeit

Aurora PostgreSQL kann eine körnige Sperrsemantik auf Zeilenebene verwenden. Die Wahrscheinlichkeit von Konflikten auf Zeilenebene steigt, wenn die folgenden Bedingungen erfüllt sind:

- Eine sehr gleichzeitige Workload beansprucht dieselben Zeilen.
- Parallelbetrieb steigt.

Leerlauf in Transaktion

Manchmal zeigt die Spalte `pg_stat_activity.state` den Wert `idle in transaction` an. Dieser Wert wird für Sitzungen angezeigt, die eine Transaktion gestartet, aber noch kein `COMMIT` oder `ROLLBACK` ausgegeben haben. Wenn der `pg_stat_activity.state`-Wert nicht `active` ist, ist die in `pg_stat_activity` angezeigte Abfrage die letzte, die ausgeführt wurde. Die blockierende Sitzung verarbeitet eine Abfrage nicht aktiv, da eine offene Transaktion eine Sperre hält.

Wenn eine Leerlauf-Transaktion eine Sperre auf Zeilenebene erworben hat, kann dies verhindern, dass andere Sitzungen sie erwerben. Diese Bedingung führt zu einem häufigen Auftreten des Warteereignisses `Lock:transactionid`. Um das Problem zu diagnostizieren, überprüfen Sie die Ausgabe von `pg_stat_activity` und `pg_locks`.

Lang laufende Transaktionen

Transaktionen, die lange laufen, erhalten lange Zeit Sperren. Diese langjährigen Sperren können die Ausführung anderer Transaktionen verhindern.

Aktionen

Zeilensperre ist ein Konflikt zwischen `UPDATE`-, `SELECT … FOR UPDATE`- oder `SELECT … FOR KEY SHARE`-Anweisungen. Bevor Sie eine Lösung versuchen, sollten Sie herausfinden, wann diese Anweisungen in derselben Zeile ausgeführt werden. Wählen Sie mit diesen Informationen eine in den folgenden Abschnitten beschriebene Strategie aus.

Themen

- Reagieren auf hohe Parallelbetrieb
- Reagieren Sie auf ungenutzte Transaktionen
- Reagieren Sie auf lang andauernde Transaktionen

Reagieren auf hohe Parallelbetrieb

Wenn Parallelität das Problem darstellt, versuchen Sie eine der folgenden Techniken:

- Senken Sie die Parallelität in der Anwendung. Verringern Sie beispielsweise die Anzahl der aktiven Sitzungen.
- Implementieren Sie einen Verbindungspool. Informationen zum Poolen von Verbindungen mit RDS-Proxy finden Sie unter Amazon-RDS-Proxy für Aurora.

- Entwerfen Sie die Anwendung oder das Datenmodell, um konkurrierende `UPDATE`- und `SELECT … FOR UPDATE`-Anweisungen zu vermeiden. Sie können auch die Anzahl der Fremdschlüssel verringern, auf die von `SELECT … FOR KEY SHARE`-Anweisungen zugegriffen wird.

Reagieren Sie auf ungenutzte Transaktionen

Wenn `pg_stat_activity.state idle in transaction` anzeigt, verwenden Sie die folgenden Strategien:

- Schalten Sie nach Möglichkeit Autocommit ein. Dieser Ansatz verhindert, dass Transaktionen andere Transaktionen blockieren, während sie auf ein `COMMIT` oder `ROLLBACK` warten.
- Suchen Sie nach Codepfaden, denen `COMMIT`, `ROLLBACK` oder `END` fehlt.
- Stellen Sie sicher, dass die Ausnahmebehandlungslogik in Ihrer Anwendung immer einen Pfad zu einem gültigen `end of transaction` hat.
- Stellen Sie sicher, dass Ihre Anwendung Abfrageergebnisse verarbeitet, nachdem die Transaktion mit `COMMIT` oder `ROLLBACK` beendet wurde.

Reagieren Sie auf lang andauernde Transaktionen

Wenn Transaktionen mit langer Laufzeit das häufige Auftreten von `Lock:transactionid` verursachen, versuchen Sie die folgenden Strategien:

- Halten Sie Zeilensperren von lang andauernden Transaktionen fern.
- Beschränken Sie die Länge von Abfragen, indem Sie nach Möglichkeit Autocommit implementieren.

Lock:tuple

Das `Lock:tuple`-Ereignis tritt ein, wenn ein Backend-Prozess darauf wartet, eine Sperre für ein Tupel zu erlangen.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Das Ereignis `Lock:tuple` zeigt an, dass ein Back-End darauf wartet, eine Sperre für ein Tupel zu erlangen, während ein anderes Back-End eine widersprüchliche Sperre für dasselbe Tupel hält. Die folgende Tabelle veranschaulicht ein Szenario, in dem Sitzungen das `Lock:tuple`-Ereignis generieren.

Zeit	1. Sitzung	2. Sitzung	3. Sitzung
t1	Startet eine Transaktion.		
t2	Aktualisiert Zeile 1.		
t3		Aktualisiert Zeile 1. Die Sitzung erwirbt eine exklusive Sperre für das Tupel und wartet dann darauf, dass Sitzung 1 die Sperre durch Commit oder Rollback freigibt.	
t4			Aktualisiert Zeile 1. Die Sitzung wartet darauf, dass Sitzung 2 die exklusive Sperre für das Tupel freigibt.

Oder Sie können dieses Warteereignis simulieren, indem Sie das Benchmarking-Tool `pgbench` verwenden. Konfigurieren Sie eine hohe Anzahl gleichzeitiger Sitzungen, um dieselbe Zeile in einer Tabelle mit einer benutzerdefinierten SQL-Datei zu aktualisieren.

Weitere Informationen zu widersprüchlichen Sperrmodi finden Sie unter Explizite Sperren in der PostgreSQL-Dokumentation. Weitere Informationen zu `pgbench` finden Sie unter pgbench in der PostgreSQL-Dokumentation.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn dieses Ereignis mehr als normal auftritt und möglicherweise auf ein Leistungsproblem hinweist, sind typische Ursachen:

- Eine große Anzahl gleichzeitiger Sitzungen versucht, eine widersprüchliche Sperre für dasselbe Tupel zu erlangen, indem sie `UPDATE`- oder `DELETE`-Anweisungen ausführen.
- In hochgradig gleichzeitigen Sitzungen wird eine `SELECT`-Anweisung ausgeführt, die den `FOR UPDATE`- oder `FOR NO KEY UPDATE`-Sperrmodus verwendet.
- Verschiedene Faktoren veranlassen Anwendungs- oder Verbindungspools dazu, weitere Sitzungen zu öffnen, um dieselben Vorgänge auszuführen. Wenn neue Sitzungen versuchen, dieselben Zeilen zu ändern, kann die DB-Last stark ansteigen und `Lock:tuple` kann erscheinen.

Weitere Informationen finden Sie unter Sperren auf Zeilenebene in der PostgreSQL-Dokumentation.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Untersuchen Sie Ihre Anwendungslogik
- Finde die Blocker-Sitzung
- Reduzieren Sie Parallelität, wenn es hoch ist
- Beheben von Engpässen

Untersuchen Sie Ihre Anwendungslogik

Finden Sie heraus, ob sich eine Blocker-Sitzung schon lange im `idle in transaction`-Zustand befindet. Wenn ja, erwägen Sie, die Blocker-Sitzung als kurzfristige Lösung zu beenden. Sie können die Funktion `pg_terminate_backend` verwenden. Weitere Informationen zu dieser Funktion finden Sie unter Server-Signalisierungsfunktionen in der PostgreSQL-Dokumentation.

Gehen Sie für eine langfristige Lösung wie folgt vor:

- Passen Sie die Anwendungslogik an.
- Verwenden Sie den Parameter `idle_in_transaction_session_timeout`. Dieser Parameter beendet jede Sitzung mit einer offenen Transaktion, die länger als die angegebene Zeitspanne im

Leerlauf ist. Weitere Informationen finden Sie unter Standardeinstellungen für Clientverbindungen in der PostgreSQL-Dokumentation.

- Verwenden Sie Autocommit so weit wie möglich. Weitere Informationen finden Sie unter SET AUTOCOMMIT in der PostgreSQL-Dokumentation.

Finde die Blocker-Sitzung

Identifizieren Sie während des `Lock:tuple`-Wait-Ereignisses den Blocker und die blockierte Sitzung, indem Sie herausfinden, welche Sperren voneinander abhängen. Weitere Informationen finden Sie unter Informationen zur Sperrabhängigkeit im PostgreSQL-Wiki. Um vergangene `Lock:tuple`-Ereignisse zu analysieren, verwenden Sie die Aurora-Funktion `aurora_stat_backend_waits`.

Im folgenden Beispiel werden alle Sitzungen abgefragt, nach `tuple` gefiltert und nach `wait_time` sortiert.

```
--AURORA_STAT_BACKEND_WAITS
      SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_query,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     left(query,80) as current_query,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we
```

```
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;

  pid  | usename | app_name |                 current_query                  |
 current_wait_type | current_wait_event | current_state | wait_type | wait_event |
 waits | wait_time
-------+---------+----------+------------------------------------------------
+-------------------+--------------------+--------------+-----------+------------
+-------+-----------
 32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000018
 11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000024
```

Reduzieren Sie Parallelität, wenn es hoch ist

Das `Lock:tuple`-Ereignis kann ständig auftreten, insbesondere in einer arbeitsreichen Zeit. Erwägen Sie in dieser Situation, die hohe Parallelität für sehr belegte Reihen zu reduzieren. Oft steuern nur wenige Zeilen eine Warteschlange oder die boolesche Logik, was diese Zeilen sehr ausgelastet macht.

Sie können die Parallelität reduzieren, indem Sie verschiedene Ansätze verwenden, die auf der Geschäftsanforderung, der Anwendungslogik und dem Workload-Typ basieren. Sie können z. B. Folgendes tun:

- Gestalten Sie Ihre Tabellen- und Datenlogik neu, um hohe Parallelität zu reduzieren.
- Ändern Sie die Anwendungslogik, um die hohe Parallelität auf Zeilenebene zu reduzieren.
- Nutzen und gestalten Sie Abfragen mit Sperren auf Zeilenebene.
- Verwenden Sie die `NOWAIT`-Klausel mit Wiederholungsvorgänge.
- Erwägen Sie, optimistische und hybridsperrende Logik-Parallelitätssteuerung zu nutzen.
- Überlegen Sie, die Isolationsstufe der Datenbank zu ändern.

Beheben von Engpässen

Das `Lock:tuple` kann bei Engpässen wie CPU-Aushungerungen oder maximaler Nutzung der Amazon EBS-Bandbreite auftreten. Um Engpässe zu verringern, sollten Sie die folgenden Ansätze berücksichtigen:

- Skalieren Sie Ihren Instance-Klassentyp hoch.
- Optimieren Sie ressourcenintensive Abfragen.
- Ändern Sie die Anwendungslogik.
- Archivieren Sie Daten, die selten zugegriffen wird.

LWLock:buffer_content (BufferContent)

Das Ereignis `LWLock:buffer_content` tritt ein, wenn eine Sitzung darauf wartet, eine Datenseite im Speicher zu lesen oder zu schreiben, während eine andere Sitzung diese Seite zum Schreiben gesperrt hat. In Aurora PostgreSQL 13 und höher heißt dieses Warteereignis `BufferContent`.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Um Daten zu lesen oder zu manipulieren, greift PostgreSQL über Shared Memory Puffer darauf zu. Um aus dem Puffer zu lesen, erhält ein Prozess eine leichte Sperre (LWLock) für den Pufferinhalt im freigegebenen Modus. Um in den Puffer zu schreiben, wird diese Sperre im exklusiven Modus angezeigt. Gemeinsame Sperren ermöglichen es anderen Prozessen, gleichzeitig gemeinsame Sperren für diesen Inhalt zu erwerben. Exklusive Sperren verhindern, dass andere Prozesse irgendeine Art von Sperre erhalten.

Das `LWLock:buffer_content` (`BufferContent`)-Ereignis zeigt an, dass mehrere Prozesse versuchen, einfache Sperren (LWLocks) auf Inhalte eines bestimmten Puffers anzuwenden.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `LWLock:buffer_content`(`BufferContent`)-Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die folgenden typischen Ursachen:

Die gleichzeitigen Aktualisierungen der gleichen Daten wurden erhöht

Es kann zu einer Zunahme der Anzahl gleichzeitiger Sitzungen mit Abfragen kommen, die denselben Pufferinhalt aktualisieren. Diese Behauptung kann bei Tabellen mit vielen Indizes ausgeprägter sein.

Workload-Daten befinden sich nicht im Speicher

Wenn sich Daten, die die aktive Workload verarbeitet, nicht im Speicher befinden, können diese Warteereignisse zunehmen. Dieser Effekt liegt daran, dass Prozesse, die Sperren halten, sie länger halten können, während sie Festplatten-I/O-Vorgänge ausführen.

Übermäßiger Einsatz von Fremdschlüsselbeschränkungen

Fremdschlüsseleinschränkungen können die Zeit erhöhen, die ein Prozess an einer Pufferinhaltssperre hält. Dieser Effekt liegt daran, dass Lesevorgänge eine gemeinsame Pufferinhaltssperre für den referenzierten Schlüssel erfordern, während dieser Schlüssel aktualisiert wird.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen. Sie können `LWLock:buffer_content`(`BufferContent`)-Ereignisse identifizieren, indem Sie Amazon RDS Performance Insights verwenden oder die Ansicht `pg_stat_activity` abfragen.

Themen

- Verbessern Sie die Effizienz im Speicher
- Reduzieren Sie die Verwendung von Fremdschlüsselbeschränkungen
- Entferne nicht verwendete Indizes
- Entfernen von doppelten Indizes
- Löschen oder NEU INDIZIEREN von ungültigen Indizes
- Verwenden von partiellen Indizes
- Entfernen von überflüssigen Daten aus Tabellen und Indizes (Bloat)

Verbessern Sie die Effizienz im Speicher

Um die Wahrscheinlichkeit zu erhöhen, dass sich aktive Workload-Daten im Speicher befinden, partitionieren Sie Tabellen oder skalieren Sie Ihre Instance-Klasse hoch. Weitere Informationen zu DB-Instance-Klassen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Überwachen Sie die `BufferCacheHitRatio`-Metrik. Diese misst den Prozentsatz der Anfragen, die vom Puffer-Cache einer DB-Instance in Ihrem DB-Cluster verarbeitet werden. Diese Metrik gibt Aufschluss über die Datenmenge, die vom Speicher bereitgestellt wird. Eine hohe Trefferrate bedeutet, dass Ihre DB-Instance über ausreichend Speicherplatz für Ihren Arbeitsdatensatz verfügt, während eine niedrige Trefferrate darauf hindeutet, dass Ihre Abfragen häufig auf Daten aus dem Speicher zugreifen.

Die Cache-Lesetreffer pro Tabelle und die Cache-Lesetreffer pro Index im Abschnitt zu den Speichereinstellungen des PG-Collector-Berichts können Aufschluss über die Cache-Trefferrate von Tabellen und Indizes geben.

Reduzieren Sie die Verwendung von Fremdschlüsselbeschränkungen

Untersuchen Sie Workloads, bei denen eine hohe Anzahl von `LWLock:buffer_content`(`BufferContent`)-Wait-Ereignissen auf die Verwendung von Fremdschlüsseleinschränkungen auftritt. Entfernen Sie unnötige Fremdschlüsselbeschränkungen.

Entferne nicht verwendete Indizes

Identifizieren Sie bei Workloads mit einer hohen Anzahl von `LWLock:buffer_content`(`BufferContent`)-Wait-Ereignissen nicht verwendete Indizes und entfernen Sie sie.

Der Abschnitt zu nicht genutzten Indizes im PG-Collector-Bericht kann Aufschluss über die nicht genutzten Indizes in der Datenbank geben.

Entfernen von doppelten Indizes

Identifizieren Sie doppelte Indizes und entfernen Sie sie.

Der Abschnitt zu doppelten Indizes im PG-Collector-Bericht kann Aufschluss über die doppelten Indizes in der Datenbank geben.

Löschen oder NEU INDIZIEREN von ungültigen Indizes

Ungültige Indizes kommen in der Regel vor, wenn `CREATE INDEX CONCURRENTLY` oder `REINDEX CONCURRENTLY` verwendet wird und der Befehl fehlschlägt oder abgebrochen wird.

Ungültige Indizes können nicht für Abfragen verwendet werden, werden aber trotzdem aktualisiert und beanspruchen Speicherplatz.

Der Abschnitt zu ungültigen Indizes im PG-Collector-Bericht kann Aufschluss über ungültige Indizes in der Datenbank geben.

Verwenden von partiellen Indizes

Partielle Indizes können verwendet werden, um die Abfrageleistung zu verbessern und die Indexgröße zu reduzieren. Ein partieller Index ist ein Index, der auf einer Teilmenge einer Tabelle basiert, wobei die Teilmenge durch einen bedingten Ausdruck definiert ist. Wie in der partial index-Dokumentation beschrieben, können partielle Indizes den Aufwand für die Verwaltung von Indizes reduzieren, da PostgreSQL den Index nicht in jedem Fall aktualisieren muss.

Entfernen von überflüssigen Daten aus Tabellen und Indizes (Bloat)

Ein übermäßiger Tabellen- und Index-Bloat kann sich negativ auf die Datenbankleistung auswirken. Tabellen und Indizes, die viel überflüssige Daten enthalten, erhöhen die Größe des aktiven Arbeitssatzes, wodurch die Effizienz im Arbeitsspeicher beeinträchtigt wird. Darüber hinaus erhöhen überflüssige Daten die Speicherkosten und verlangsamen die Ausführung von Abfragen. Informationen zur Bloat-Diagnose finden Sie unter Diagnostizieren einer Überlastung von Tabellen und Indizes. Darüber hinaus gibt der Abschnitt zur Fragmentierung (Bloat) im PG-Collector-Bericht Aufschluss über Tabellen und Indizes mit überflüssigen Daten.

Zur Behebung des Problems der überflüssigen Daten in Tabellen und Indizes gibt es einige Optionen:

VACUUM FULL

> `VACUUM FULL` erstellt eine neue Kopie der Tabelle, wobei nur die aktiven Tupel kopiert werden, und ersetzt dann die alte Tabelle durch die neue, wobei eine `ACCESS EXCLUSIVE`-Sperre beibehalten wird. Dadurch wird jegliches Lesen oder Schreiben in die Tabelle verhindert, was zu einem Ausfall führen kann. Außerdem dauert `VACUUM FULL` länger, wenn die Tabelle groß ist.

pg_repack

> `pg_repack` ist hilfreich in Situationen, in denen `VACUUM FULL` möglicherweise nicht geeignet ist. Es erstellt eine neue Tabelle, die die Daten der Tabelle mit den überflüssigen Daten enthält,

verfolgt die Änderungen gegenüber der Originaltabelle nach und ersetzt diese Tabelle dann durch die neue. Es sperrt die Originaltabelle nicht für Lese- oder Schreibvorgänge, während die neue Tabelle erstellt wird. Weitere Informationen zur Verwendung von `pg_repack` finden Sie unter Reduzieren von überflüssigen Daten mit pg_repack und unter pg_repack.

REINDEX

Der `REINDEX`-Befehl kann genutzt werden, um das Problem eines Index-Bloat zu beheben. `REINDEX` schreibt eine neue Version des Index ohne die „toten“ oder leeren bzw. fast leeren Seiten, was den Platzverbrauch des Index reduziert. Detaillierte Informationen zum `REINDEX`-Befehl finden Sie in der REINDEX-Dokumentation.

Nach dem Entfernen der überflüssigen Daten aus Tabellen und Indizes kann es notwendig sein, die Selbstbereinigungsfrequenzen für diese Tabellen zu erhöhen. Die Implementierung von Einstellungen für eine aggressive Selbstbereinigung auf Tabellenebene kann dazu beitragen, überflüssige Daten zu vermeiden. Weitere Informationen finden Sie in der Dokumentation über `Vacuuming and analyzing tables automatically`.

LWLock:buffer_mapping

Dieses Ereignis tritt ein, wenn eine Sitzung darauf wartet, einen Datenblock einem Puffer im gemeinsam genutzten Pufferpool zuzuordnen.

> **Note**
>
> Dieses Ereignis wird in Aurora PostgreSQL Version 12 und niedriger als `LWLock:buffer_mapping` und in Version 13 und höher als `LWLock:BufferMapping` angezeigt.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Ursachen
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen sind für Aurora PostgreSQL Version 9.6 und höher relevant.

Kontext

Der freigegebene Pufferpool ist ein Aurora PostgreSQL-Speicherbereich, der alle Seiten enthält, die von Prozessen verwendet werden oder wurden. Wenn ein Prozess eine Seite benötigt, liest er die Seite in den freigegebenen Pufferpool. Der Parameter `shared_buffers` legt die Größe des gemeinsam genutzten Puffers fest und reserviert einen Speicherbereich zum Speichern der Tabellen- und Indexseiten. Wenn Sie diesen Parameter ändern, stellen Sie sicher, dass Sie die Datenbank neu starten. Weitere Informationen finden Sie unter Freigegebene Puffer.

Das `LWLock:buffer_mapping`-Wait-Ereignis tritt in den folgenden Szenarien auf:

- Ein Prozess durchsucht die Puffertabelle nach einer Seite und erwirbt eine freigegebene Puffer-Mapping-Sperre.
- Ein Prozess lädt eine Seite in den Pufferpool und erwirbt eine exklusive Puffer-Mapping-Sperre.
- Ein Prozess entfernt eine Seite aus dem Pool und erwirbt eine exklusive Puffer-Mapping-Sperre.

Ursachen

Wenn dieses Ereignis mehr als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, greift die Datenbank in und aus dem freigegebenen Pufferpool aus. Zu den typischen Ursachen zählen auch die Folgenden:

- Große Abfragen
- Aufgeblähte Indizes und Tabellen
- Vollständige Tabellenscans
- Eine gemeinsame Poolgröße, die kleiner als der Arbeitssatz ist

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Überwachen Sie pufferbezogene Metriken
- Bewerten Sie Ihre Indexierungsstrategie

- Reduzieren Sie die Anzahl der Puffer, die schnell zugewiesen werden müssen

Überwachen Sie pufferbezogene Metriken

Wenn `LWLock:buffer_mapping` auf Spitze wartet, untersuchen Sie die Puffertrefferquote. Sie können diese Metriken verwenden, um ein besseres Verständnis dafür zu erhalten, was im Puffer-Cache passiert. Untersuchen Sie die folgenden Metriken:

`BufferCacheHitRatio`

Diese Amazon CloudWatch-Metrik misst den Prozentsatz der Anfragen, die vom Puffer-Cache einer DB-Instance in Ihrem DB-Cluster verarbeitet werden. Möglicherweise sehen Sie diese Metrikabnahme im Vorfeld des `LWLock:buffer_mapping`-Wait-Ereignisses.

`blks_hit`

Diese Zählermetrik für Performance Insights gibt die Anzahl der Blöcke an, die aus dem freigegebenen Pufferpool abgerufen wurden. Nachdem das Wait-Ereignis `LWLock:buffer_mapping` aufgetreten ist, können Sie eine Spitze in `blks_hit` beobachten.

`blks_read`

Diese Zählermetrik für Performance Insights gibt die Anzahl der Blöcke an, für die I/O in den freigegebenen Pufferpool eingelesen werden mussten. Sie können im Vorfeld des `LWLock:buffer_mapping`-Warteereignisses eine Spitze in `blks_read` beobachten.

Bewerten Sie Ihre Indexierungsstrategie

Überprüfen Sie Folgendes, um zu bestätigen, dass Ihre Indexierungsstrategie die Leistung nicht beeinträchtigt:

Indexblähung

Stellen Sie sicher, dass Index und Tabellenaufblähungen nicht dazu führen, dass unnötige Seiten in den freigegebenen Puffer gelesen werden. Wenn Ihre Tabellen nicht verwendete Zeilen enthalten, sollten Sie die Daten archivieren und die Zeilen aus den Tabellen entfernen. Sie können dann die Indizes für die skalierten Tabellen neu erstellen.

Indizes für häufig verwendete Abfragen

Um festzustellen, ob Sie über die optimalen Indizes verfügen, überwachen Sie die Metriken der DB-Engine in Performance Insights. Die `tup_returned`-Metrik zeigt die Anzahl der gelesenen

Zeilen an. Die `tup_fetched`-Metrik zeigt die Anzahl der an den Client zurückgegebenen Zeilen. Wenn `tup_returned` deutlich größer als `tup_fetched` ist, werden die Daten möglicherweise nicht richtig indiziert. Außerdem sind Ihre Tabellenstatistiken möglicherweise nicht aktuell.

Reduzieren Sie die Anzahl der Puffer, die schnell zugewiesen werden müssen

Um die `LWLock:buffer_mapping`-Warteereignisse zu reduzieren, versuchen Sie, die Anzahl der Puffer zu reduzieren, die schnell zugewiesen werden müssen. Eine Strategie besteht darin, kleinere Batch-Vorgänge durchzuführen. Möglicherweise können Sie kleinere Batches erreichen, indem Sie Ihre Tabellen partitionieren.

LWLock:BufferIO (IPC:BufferIO)

Das `LWLock:BufferIO`-Ereignis tritt ein, wenn Aurora PostgreSQL oder RDS for PostgreSQL darauf wartet, dass andere Prozesse ihre Eingabe-/Ausgabe-(I/O)-Vorgänge beenden, wenn sie gleichzeitig versuchen, auf eine Seite zuzugreifen. Sein Zweck besteht darin, dass dieselbe Seite in den freigegebenen Puffer eingelesen wird.

Themen

- Relevante Engine-Versionen
- Kontext
- Ursachen
- Aktionen

Relevante Engine-Versionen

Diese Warteereignisinformationen sind für alle Versionen von Aurora PostgreSQL. Für Aurora PostgreSQL 12 und frühere Versionen wird dieses Warteereignis als lwlock:buffer_io bezeichnet, während es in der Version Aurora PostgreSQL 13 den Namen lwlock:bufferio trägt. Aus der Version Aurora PostgreSQL 14 wurde das BufferIO-Warteereignis von `LWLock` zum Warteereignistyp `IPC` (ipc:bufferio) verschoben.

Kontext

Jeder gemeinsam genutzte Puffer hat eine I/O-Sperre, die mit dem `LWLock:BufferIO`-Warteereignis verbunden ist, jedes Mal, wenn ein Block (oder eine Seite) außerhalb des gemeinsam genutzten Pufferpools abgerufen werden muss.

Diese Sperre wird verwendet, um mehrere Sitzungen zu behandeln, die alle Zugriff auf denselben Block benötigen. Dieser Block muss von außerhalb des gemeinsam genutzten Pufferpools gelesen werden, der durch den `shared_buffers`-Parameter definiert wird.

Sobald die Seite innerhalb des Shared Buffer Pool gelesen wird, wird die `LWLock:BufferIO`-Sperre freigegeben.

> **Note**
>
> Das `LWLock:BufferIO`-Wait-Ereignis geht dem IO:DataFileRead-Warteereignis voraus. Das `IO:DataFileRead`-Wait-Ereignis tritt auf, während Daten aus dem Speicher gelesen werden.

Weitere Informationen zu leichten Sperren finden Sie unter Übersicht über Sperren.

Ursachen

Häufige Gründe dafür, dass das `LWLock:BufferIO`-Ereignis in den Top-Wartezeiten angezeigt wird, sind die folgenden:

- Mehrere Backends oder Verbindungen, die versuchen, auf dieselbe Seite zuzugreifen, für die auch ein I/O-Vorgang aussteht
- Das Verhältnis zwischen der Größe des gemeinsam genutzten Pufferpools (definiert durch den `shared_buffers`-Parameter) und der Anzahl der Puffer, die von der aktuellen Workload benötigt werden
- Die Größe des freigegebenen Pufferpools ist nicht gut mit der Anzahl der Seiten, die durch andere Vorgänge geräumt werden
- Große oder aufgeblähte Indizes, bei denen die Engine mehr Seiten als nötig in den freigegebenen Pufferpool lesen muss
- Mangel an Indizes, die die DB-Engine dazu zwingen, mehr Seiten aus den Tabellen als nötig zu lesen
- Plötzliche Spitzen für Datenbankverbindungen, die versuchen, Vorgänge auf derselben Seite auszuführen

Aktionen

Abhängig von den Ursachen Ihres Wait-Ereignisses empfehlen wir verschiedene Aktionen:

- Beobachten Sie die Amazon CloudWatch-Metriken auf Korrelation zwischen starken Abnahmen der `BufferCacheHitRatio`- und `LWLock:BufferIO`-Warteereignisse. Dieser Effekt kann bedeuten, dass Sie eine kleine Einstellung für gemeinsame Puffer haben. Möglicherweise müssen Sie es erhöhen oder Ihre DB-Instance-Klasse hochskalieren. Sie können Ihre Workload in mehr Reader-Knoten aufteilen.
- Überprüfen Sie, ob Sie nicht verwendete Indizes haben, und entfernen Sie sie dann.
- Verwenden Sie partitionierte Tabellen (die auch partitionierte Indizes haben). Dies hilft, die Neuordnung des Index niedrig zu halten und ihre Auswirkungen zu reduzieren.
- Vermeiden Sie, Spalten unnötig zu indizieren.
- Verhindern Sie plötzliche Spitzen der Datenbankverbindung, indem Sie einen Verbindungspool verwenden.
- Beschränken Sie die maximale Anzahl von Verbindungen zur Datenbank als bewährte Methode

LWLock:lock_manager

Dieses Ereignis tritt ein, wenn die Aurora PostgreSQL-Engine den Speicherbereich der gemeinsam genutzten Sperre verwaltet, um eine Sperre zuzuweisen, zu überprüfen und aufzuheben, wenn eine Fast-Path-Sperre nicht möglich ist.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen sind für Aurora PostgreSQL Version 9.6 und höher relevant.

Kontext

Wenn Sie eine SQL-Anweisung ausgeben, zeichnet Aurora PostgreSQL Sperren auf, um die Struktur, Daten und Integrität Ihrer Datenbank während gleichzeitiger Vorgänge zu schützen. Der Motor kann dieses Ziel mit einer schnellen Pfadsperre oder einer nicht schnellen Pfadsperre erreichen. Eine Pfadsperre, die nicht schnell ist, ist teurer und erzeugt mehr Overhead als eine schnelle Pfadsperre.

Schnelle Pfadsperre

Um den Overhead von Sperren zu reduzieren, die häufig genommen und freigegeben werden, aber selten in Konflikt geraten, können Backend-Prozesse eine schnelle Pfadsperrung verwenden. Die Datenbank verwendet diesen Mechanismus für Sperren, die die folgenden Kriterien erfüllen:

- Sie verwenden die STANDARD–Sperrmethode.
- Sie stellen eine Sperre für eine Datenbankbeziehung statt einer gemeinsamen Beziehung dar.
- Sie sind schwache Sperren, die wahrscheinlich nicht in Konflikt stehen.
- Die Engine kann schnell überprüfen, dass keine widersprüchlichen Sperren existieren können.

Die Engine kann keine schnelle Pfadsperre verwenden, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Sperre erfüllt nicht die vorhergehenden Kriterien.
- Für den Backend-Prozess sind keine Slots mehr verfügbar.

Weitere Informationen zum Sperren von Fast Path finden Sie unter Fast Path in der README-Datei des PostgreSQL-Sperrmanagers und unter pg-locks in der PostgreSQL-Dokumentation.

Beispiel für ein Skalierungsproblem für den Sperrmanager

In diesem Beispiel speichert eine Tabelle mit dem Namen `purchases` Daten aus fünf Jahren, aufgeteilt nach Tagen. Jede Partition hat zwei Indizes. Die folgende Abfolge von Ereignissen tritt auf:

1. Sie fragen Daten für viele Tage ab, wodurch die Datenbank viele Partitionen lesen muss.
2. Die Datenbank erstellt einen Sperreintrag für jede Partition. Wenn Partitionsindizes Teil des Optimizer-Zugriffspfads sind, erstellt die Datenbank auch für sie einen Sperreintrag.
3. Wenn die Anzahl der angeforderten Sperreneinträge für denselben Backend-Prozess höher als 16 ist, was dem Wert von `FP_LOCK_SLOTS_PER_BACKEND` entspricht, verwendet der Sperrenmanager die Sperrmethode ohne Fast Path.

Moderne Anwendungen haben möglicherweise Hunderte von Sitzungen. Wenn gleichzeitige Sitzungen das übergeordnete Element ohne ordnungsgemäßen Schnitt von Partitionen abfragen, erstellt die Datenbank möglicherweise Hunderte oder sogar Tausende von nicht schnellen Pfadsperren. Wenn diese Parallelität höher als die Anzahl der vCPUs ist, wird normalerweise das `LWLock:lock_manager`-Warteereignis angezeigt.

> **Note**
>
> Das Wait-Ereignis `LWLock:lock_manager` hat nichts mit der Anzahl der Partitionen oder Indizes in einem Datenbankschema zu tun. Stattdessen hängt es mit der Anzahl der nicht schnellen Pfadsperren zusammen, die die Datenbank steuern muss.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Wenn das `LWLock:lock_manager` häufiger als normal auftritt, was möglicherweise auf ein Leistungsproblem hinweist, sind die wahrscheinlichsten Ursachen für plötzliche Spitzen wie folgt:

- Gleichzeitige aktive Sitzungen führen Abfragen aus, die keine schnellen Pfadsperren verwenden. Diese Sitzungen überschreiten auch die maximale vCPU.
- Eine große Anzahl gleichzeitiger aktiver Sitzungen greift auf eine stark partitionierte Tabelle zu. Jede Partition hat mehrere Indizes.
- Die Datenbank erlebt einen Verbindungssturm. Standardmäßig erzeugen einige Anwendungen und Connection Pool-Software mehr Verbindungen, wenn die Datenbank langsam ist. Diese Praxis verschlimmert das Problem. Optimieren Sie Ihre Connection Pool-Software so, dass keine Verbindungsstürme auftreten.
- Eine große Anzahl von Sitzungen fragt eine übergeordnete Tabelle ab, ohne Partitionen zu beschneiden.
- Eine Datendefinitionssprache (DDL), Datenmanipulationssprache (DML) oder ein Wartungsbefehl sperrt ausschließlich eine Beleg-Beziehung oder Tupel, auf die häufig zugegriffen oder geändert werden.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

Themen

- Verwenden Sie das Beschneiden von Partitionen
- Entfernen unnötiger Indizes
- Optimieren Sie Ihre Abfragen für schnelles Pfadsperren
- Tune auf andere Warteereignisse
- Reduzieren von Hardware-Engpässen

- Verwenden eines Verbindungs-Poolers
- Aktualisieren Sie Ihre Aurora PostgreSQL Version

Verwenden Sie das Beschneiden von Partitionen

Die Partitionsbereinigung ist eine Strategie zur Abfrageoptimierung, die nicht benötigte Partitionen von Tabellenscans ausschließt und dadurch die Leistung verbessert. Das Beschneiden der Partition ist standardmäßig aktiviert. Wenn es ausgeschaltet ist, schalten Sie es wie folgt ein.

```
SET enable_partition_pruning = on;
```

Abfragen können die Partitionsbereinigung nutzen, wenn ihre `WHERE`-Klausel die für die Partitionierung verwendete Spalte enthält. Weitere Informationen finden Sie unter Partitionsbereinigung in der PostgreSQL-Dokumentation.

Entfernen unnötiger Indizes

Ihre Datenbank enthält möglicherweise nicht verwendete oder selten verwendete Indizes. Wenn ja, erwägen Sie, sie zu löschen. Führen Sie eine der folgenden Aufgaben aus:

- Erfahren Sie, wie Sie unnötige Indizes finden, indem Sie Ungenutzte Indizes im PostgreSQL-Wiki lesen.
- Führen Sie PG Collector aus. Dieses SQL-Skript sammelt Datenbankinformationen und präsentiert sie in einem konsolidierten HTML-Bericht. Überprüfen Sie den Abschnitt „Unbenutzte Indizes“. Weitere Informationen finden Sie unter pg-collector im AWS-Labs GitHub-Repository.

Optimieren Sie Ihre Abfragen für schnelles Pfadsperren

Um herauszufinden, ob Ihre Abfragen Fast Path Locking verwenden, fragen Sie die `fastpath`-Spalte in der `pg_locks`-Tabelle ab. Wenn Ihre Abfragen keine schnelle Pfadsperre verwenden, versuchen Sie, die Anzahl der Beziehungen pro Abfrage auf weniger als 16 zu reduzieren.

Tune auf andere Warteereignisse

Wenn `LWLock:lock_manager` in der Liste der Top-Waits an erster oder zweiter Stelle steht, überprüfen Sie, ob die folgenden Wait-Ereignisse auch in der Liste erscheinen:

- `Lock:Relation`

- `Lock:transactionid`
- `Lock:tuple`

Wenn die vorhergehenden Ereignisse in der Liste hoch erscheinen, sollten Sie zuerst diese Warteereignisse optimieren. Diese Ereignisse können ein Treiber für `LWLock:lock_manager`.

Reduzieren von Hardware-Engpässen

Möglicherweise haben Sie einen Hardware-Engpass wie CPU-Hunger oder maximale Auslastung Ihrer Amazon EBS-Bandbreite. Ziehen Sie in diesen Fällen die Verringerung der Hardware-Engpässe in Betracht. Berücksichtigen Sie die folgenden Aktionen:

- Skalieren Sie Ihre Instance-Klasse hoch.
- Optimieren Sie Abfragen, die große Mengen an CPU und Speicher verbrauchen.
- Ändern Sie Ihre Anwendungslogik.
- Archiviere deine Daten.

Weitere Informationen zu CPU, Arbeitsspeicher und EBS-Netzwerkbandbreite finden Sie unter Amazon RDS-Instance-Typen.

Verwenden eines Verbindungs-Poolers

Wenn Ihre Gesamtzahl aktiver Verbindungen die maximale vCPU überschreitet, benötigen mehr Betriebssystemprozesse CPU, als Ihr Instance-Typ unterstützen kann. Ziehen Sie in diesem Fall die Verwendung oder Abstimmung eines Verbindungspool in Betracht. Weitere Informationen zu den vCPUs für Ihren Instance-Typ finden Sie unter Amazon RDS-Instance-Typen.

Weitere Informationen zum Verbindungspooling finden Sie in den folgenden Ressourcen:

- Amazon-RDS-Proxy für Aurora
- pgbouncer
- Verbindungspools und Datenquellen in der PostgreSQL-Dokumentation

Aktualisieren Sie Ihre Aurora PostgreSQL Version

Wenn Ihre aktuelle Version von Aurora PostgreSQL niedriger als 12 ist, aktualisieren Sie auf Version 12 oder höher. PostgreSQL Versionen 12 und 13 haben einen verbesserten Partitionsmechanismus.

Weitere Informationen zu Version 12 finden Sie in den Versionshinweisen zu PostgreSQL 12.0. Weitere Informationen zum Aktualisieren von Aurora PostgreSQL finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL.

LWLock:MultiXact

Die Warteereignisse `LWLock:MultiXactMemberBuffer`, `LWLock:MultiXactOffsetBuffer`, `LWLock:MultiXactMemberSLRU` und `LWLock:MultiXactOffsetSLRU` geben an, dass eine Sitzung darauf wartet, eine Liste von Transaktionen abzurufen, die dieselbe Zeile in einer bestimmten Tabelle ändern.

- `LWLock:MultiXactMemberBuffer` – Ein Prozess wartet auf I/O in einem einfachen, am wenigsten zuletzt verwendeten (SLRU) Puffer für ein Multixact-Mitglied.
- `LWLock:MultiXactMemberSLRU` – Ein Prozess wartet darauf, auf den einfachen, am wenigsten zuletzt verwendeten (SLRU) Cache für ein Multixact-Mitglied zuzugreifen.
- `LWLock:MultiXactOffsetBuffer` – Ein Prozess wartet auf I/O in einem einfachen, am wenigsten zuletzt verwendeten (SLRU) Puffer für ein Multixact-Offset.
- `LWLock:MultiXactOffsetSLRU` – Ein Prozess wartet darauf, auf den einfachen, am wenigsten zuletzt verwendeten (SLRU) Cache für ein Multixact-Offset zuzugreifen.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Multixact ist eine Datenstruktur, die eine Liste von Transaktions-IDs (XIDs) speichert, welche dieselbe Tabellenzeile ändern. Wenn eine einzelne Transaktion auf eine Zeile in einer Tabelle verweist, wird die Transaktions-ID in der Kopfzeile der Tabelle gespeichert. Verweisen mehrere Transaktionen auf dieselbe Zeile in einer Tabelle, wird die Liste der Transaktions-IDs in der Multixact-Datenstruktur

gespeichert. Die Multixact-Warteereignisse geben an, dass eine Sitzung die Liste der Transaktionen, die sich auf eine bestimmte Zeile in einer Tabelle beziehen, aus der Datenstruktur abruft.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

3 häufige Ursachen für die Verwendung von Multixact sind die folgenden:

- Untertransaktionen expliziter Savepoints – Wenn Sie explizit einen Savepoint in Ihren Transaktionen erstellen, werden neue Transaktionen für dieselbe Zeile generiert. Verwenden Sie beispielsweise `SELECT FOR UPDATE`, `SAVEPOINT` und dann `UPDATE`.

  Einige Treiber, objektrelationale Mappers (ORMs) und Abstraktionsschichten verfügen über Konfigurationsoptionen, um alle Operationen automatisch mit Savepoints zu umschließen. Dies kann bei einigen Workloads zu vielen Multixact-Warteereignissen führen. Die `autosave`-Option des PostgreSQL-JDBC-Treibers ist ein Beispiel dafür. Weitere Informationen finden Sie unter pgJDBC in der PostgreSQL-JDBC-Dokumentation. Ein anderes Beispiel ist der PostgreSQL-ODBC-Treiber und seine `protocol`-Option. Weitere Informationen finden Sie unter psqlODBC-Konfigurationsoptionen in der Dokumentation zu PostgreSQL-ODBC-Treibern.
- Untertransaktionen von PL/pgSQL-EXCEPTION-Klauseln – Jede `EXCEPTION`-Klausel, die Sie in Ihren PL/pgSQL-Funktionen oder -Prozeduren schreiben, erstellt intern einen `SAVEPOINT`.
- Fremdschlüssel – Mehrere Transaktionen erwerben eine Freigabesperre für den übergeordneten Datensatz.

Wenn eine bestimmte Zeile in einer Operation mit mehreren Transaktionen enthalten ist, müssen für die Verarbeitung der Zeile die Transaktions-IDs aus den `multixact`-Auflistungen abgerufen werden. Wenn durch Lookups der Multixact nicht aus dem Speichercache abgerufen werden kann, muss die Datenstruktur aus der Aurora-Speicherschicht gelesen werden. Dieser I/O-Vorgang aus dem Speicher bedeutet, dass SQL-Abfragen länger dauern können. Speicher-Cache-Fehler können bei starker Auslastung aufgrund einer großen Anzahl von mehreren Transaktionen auftreten. All diese Faktoren tragen zu einer Zunahme dieses Warteereignisses bei.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen. Einige dieser Aktionen können dazu beitragen, die Warteereignisse sofort zu reduzieren. Andere erfordern jedoch möglicherweise eine Untersuchung und Korrektur, um Ihren Workload zu skalieren.

Themen

- Durchführen einer Bereinigungseinfrierung auf Tabellen mit diesem Warteereignis
- Erhöhen der Häufigkeit der Selbstbereinigung in Tabellen mit diesem Warteereignis
- Erhöhen der Speicherparameter
- Reduzieren von Transaktionen mit langer Laufzeit
- Langfristige Aktionen

Durchführen einer Bereinigungseinfrierung auf Tabellen mit diesem Warteereignis

Wenn die Anzahl dieses Warteereignisses plötzlich stark ansteigt und sich auf Ihre Produktionsumgebung auswirkt, können Sie eine der folgenden temporären Methoden verwenden, um die Anzahl zu reduzieren.

- Verwenden Sie VACUUM FREEZE für die betroffene Tabelle oder Tabellenpartition, um das Problem sofort zu beheben. Weitere Informationen finden Sie unter VACUUM.
- Verwenden Sie die Klausel VACUUM (FREEZE, INDEX_CLEANUP FALSE), um eine schnelle Bereinigung durch Überspringen von Indizes durchzuführen. Weitere Informationen finden Sie unter Möglichst schnelles Bereinigen einer Tabelle.

Erhöhen der Häufigkeit der Selbstbereinigung in Tabellen mit diesem Warteereignis

Nach dem Scannen aller Tabellen in allen Datenbanken entfernt die VACUUM-Operation schließlich Multixacts, wobei die ältesten Multixact-Werte weiterverwendet werden. Weitere Informationen finden Sie unter Multixacts und Wraparound. Um die LWLock:MultiXact-Warteereignisse auf ein Minimum zu beschränken, müssen Sie den VACUUM-Prozess so oft wie nötig ausführen. Stellen Sie dazu sicher, dass der VACUUM-Prozess in Ihrem Aurora PostgreSQL-DB-Cluster optimal konfiguriert ist.

Wenn die Verwendung von VACUUM FREEZE für die betroffene Tabelle oder Tabellenpartition das Problem mit dem Warteereignis behebt, empfehlen wir, einen Scheduler wie `pg_cron` für die VACUUM-Durchführung zu verwenden, anstatt den Selbstbereinigung auf Instance-Ebene anzupassen.

Damit die Selbstbereinigung häufiger stattfindet, können Sie den Wert des `autovacuum_multixact_freeze_max_age`-Speicherparameters in der betroffenen Tabelle reduzieren. Weitere Informationen finden Sie unter autovacuum_multixact_freeze_max_age.

Erhöhen der Speicherparameter

Sie können die Speichernutzung für Multixact-Caches optimieren, indem Sie die folgenden Parameter anpassen. Diese Einstellungen steuern, wie viel Speicher für diese Caches reserviert wird. Dies kann dazu beitragen, Multixact-Warteereignisse in Ihrem Workload zu reduzieren. Wir empfehlen, mit den folgenden Werten zu beginnen:

Für Aurora PostgreSQL 17 und höher:

- `multixact_offset_buffers` = 128
- `multixact_member_buffers` = 256

Für Aurora PostgreSQL 16 und niedriger:

- `multixact_offsets_cache_size` = 128
- `multixact_members_cache_size` = 256

> **Note**
>
> In Aurora PostgreSQL 17 wurden Parameternamen von `multixact_offsets_cache_size` in `multixact_offset_buffers` und von `multixact_members_cache_size` in `multixact_member_buffers` geändert, um sie an die Community von PostgreSQL 17 anzupassen.

Sie können diese Parameter auf Cluster-Ebene festlegen, sodass alle Instances in Ihrem Cluster konsistent bleiben. Wir empfehlen Ihnen, die Werte zu testen und anzupassen, damit sie Ihren spezifischen Workload-Anforderungen und Ihrer Instance-Klasse optimal entsprechen. Sie müssen die Writer-Instance neu starten, damit die Parameteränderung wirksam wird.

Die Parameter werden in Form von Multixact-Cache-Einträgen ausgedrückt. Jeder Cache-Eintrag belegt Speicherplatz Höhe von `8 KB`. Um den gesamten reservierten Speicher zu berechnen, multiplizieren Sie jeden Parameterwert mit `8 KB`. Wenn Sie beispielsweise einen Parameter auf 128 setzen, beträgt der gesamte reservierte Speicher `128 * 8 KB = 1 MB`.

Reduzieren von Transaktionen mit langer Laufzeit

Transaktionen mit langer Laufzeit führen dazu, dass der Bereinigungsprozess Informationen beibehält, bis die Transaktion bestätigt oder die schreibgeschützte Transaktion geschlossen wird. Wir empfehlen, langlebige Transaktionen proaktiv zu überwachen und zu verwalten. Weitere

Informationen finden Sie unter Die Datenbank läuft seit langem inaktiv in Transaktionsverbindung. Ändern Sie Ihre Anwendung, wenn möglich, um die Verwendung von Transaktionen mit langer Laufzeit zu vermeiden oder zu minimieren.

Langfristige Aktionen

Untersuchen Sie Ihren Workload, um die Ursache für den Multixact-Überlauf zu ermitteln. Sie müssen das Problem beheben, um Ihren Workload zu skalieren und Warteereignisse zu reduzieren.

- Sie müssen die DDL (Datendefinitionssprache) analysieren, die zum Erstellen Ihrer Tabellen verwendet wurde. Stellen Sie sicher, dass die Tabellenstrukturen und Indizes optimal gestaltet sind.
- Wenn die betroffenen Tabellen Fremdschlüssel enthalten, ermitteln Sie, ob sie benötigt werden oder ob es eine andere Möglichkeit gibt, die referenzielle Integrität durchzusetzen.
- Wenn eine Tabelle große, ungenutzte Indizes enthält, kann das dazu führen, dass der Selbstbereinigungsvorgang nicht zu Ihrem Workload passt und die Ausführung möglicherweise blockiert. Um dies zu vermeiden, suchen Sie nach ungenutzten Indizes und entfernen Sie diese vollständig. Weitere Informationen finden Sie unter Verwalten der automatischen Bereinigung mit großen Indizes.
- Reduzieren Sie die Verwendung von Savepoints in Ihren Transaktionen.

LWLock:pg_stat_statements

Das Warteereignis LWLock:pg_stat_statements tritt auf, wenn die `pg_stat_statements`-Erweiterung die Hash-Tabelle, die die SQL-Anweisungen nachverfolgt, exklusiv sperrt. Dies passiert in folgenden Szenarien:

- Wenn die Anzahl der nachverfolgten Anweisungen den konfigurierten `pg_stat_statements.max`-Parameterwert erreicht und Platz für weitere Einträge geschaffen werden muss, sortiert die Erweiterung die Anzahl der Aufrufe, entfernt die 5 % der am wenigsten ausgeführten Anweisungen und füllt den Hash mit den verbleibenden Einträgen neu auf.
- Wenn `pg_stat_statements` eine `garbage collection`-Operation für die `pgss_query_texts.stat`-Datei auf der Festplatte durchführt und die Datei neu schreibt

Themen

- Unterstützte Engine-Versionen

- Kontext
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Grundlagen der pg_stat_statements-Erweiterung – Die pg_stat_statements-Erweiterung verfolgt Statistiken zur Ausführung von SQL-Anweisungen in einer Hash-Tabelle nach. Die Erweiterung verfolgt SQL-Anweisungen bis zu dem durch den `pg_stat_statements.max`-Parameter definierten Grenzwert nach. Dieser Parameter bestimmt die maximale Anzahl von Anweisungen, die nachverfolgt werden können, was der maximalen Anzahl von Zeilen in der pg_stat_statements-Ansicht entspricht.

Persistenz von Anweisungsstatistiken – Die Erweiterung behält die Anweisungsstatistiken über Instance-Neustarts hinweg bei, indem sie Folgendes durchführt:

- Schreiben von Daten in eine Datei mit dem Namen pg_stat_statements.stat
- Steuern des Persistenzverhaltens mit dem pg_stat_statements.save-Parameter

Wenn pg_stat_statements.save auf Folgendes gesetzt ist:

- on (Standard): Statistiken werden beim Herunterfahren gespeichert und beim Serverstart neu geladen
- off: Statistiken werden weder beim Herunterfahren gespeichert noch beim Serverstart neu geladen

Abfragetextspeicher – Die Erweiterung speichert den Text nachverfolgter Abfragen in einer Datei mit dem Namen `pgss_query_texts.stat`. Diese Datei kann auf das Doppelte der durchschnittlichen Größe aller nachverfolgten SQL-Anweisungen anwachsen, ehe eine Garbage Collection erfolgt. Die Erweiterung erfordert eine exklusive Hash-Tabellen-Sperre während der Bereinigungsvorgänge und beim Neuschreiben der `pgss_query_texts.stat`-Datei.

Prozess der Freigabe von Anweisungen – Wenn die Anzahl der nachverfolgten Anweisungen den `pg_stat_statements.max`-Grenzwert erreicht und neue Anweisungen nachverfolgt werden müssen, führt die Erweiterung Folgendes durch:

- Sie sperrt die Hash-Tabelle exklusiv (LWLock:pg_stat_statements).
- Sie lädt vorhandene Daten in den lokalen Speicher.
- Sie führt eine Schnellsortierung auf der Grundlage der Anzahl der Aufrufe durch.
- Sie entfernt die am wenigsten aufgerufenen Anweisungen (unterste 5 %).
- Sie füllt die Hash-Tabelle mit den verbleibenden Einträgen neu auf.

Überwachung der Anweisungsfreigabe – In PostgreSQL 14 und höher können Sie die Freigabe von Anweisungen mit der pg_stat_statements_info-Ansicht überwachen. Diese Ansicht enthält eine Freigabe-Spalte, in der angezeigt wird, wie oft Anweisungen freigegeben wurden, um Platz für neue Anweisungen zu schaffen.

Wenn die Freigabe von Anweisungen häufig erfolgt, führt dies zu einer häufigeren Garbage Collection bei der `pgss_query_texts.stat`-Datei auf der Festplatte.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den typischen Ursachen für eine Zunahme bei den `LWLock:pg_stat_statements`-Wartezeiten gehören:

- eine Zunahme bei der Anzahl eindeutiger Abfragen, die von der Anwendung verwendet werden
- ein `pg_stat_statements.max`-Parameterwert, der im Vergleich zur Anzahl der verwendeten eindeutigen Abfragen klein ist

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen. Sie können `LWLock:pg_stat_statements`-Ereignisse identifizieren, indem Sie Erkenntnisse zur Amazon-RDS-Leistung verwenden oder die `pg_stat_activity`-Ansicht abfragen.

Passen Sie die folgenden `pg_stat_statements`-Parameter an, um das Nachverfolgungsverhalten zu steuern und die Anzahl der LWLock:pg_stat_statements-Warteereignisse zu reduzieren.

Themen

- Deaktivieren des pg_stat_statements.track-Parameters
- Erhöhen des pg_stat_statements.max-Parameters
- Deaktivieren des pg_stat_statements.track_utility-Parameters

Deaktivieren des pg_stat_statements.track-Parameters

Wenn das LWLock:pg_stat_statements-Warteereignis die Datenbankleistung negativ beeinträchtigt und eine schnelle Lösung erforderlich ist, bevor die `pg_stat_statements`-Ansicht weiter analysiert wird, um die Ursache zu ermitteln, kann der `pg_stat_statements.track`-Parameter deaktiviert werden, indem er auf `none` gesetzt wird. Dadurch wird die Erfassung von Anweisungsstatistiken deaktiviert.

Erhöhen des pg_stat_statements.max-Parameters

Erhöhen Sie den Wert des `pg_stat_statements.max`-Parameters, um die Freigabe zu reduzieren und die Garbage Collection für die `pgss_query_texts.stat`-Datei auf der Festplatte zu minimieren. Der Standardwert ist `5,000`.

> **Note**
>
> Der `pg_stat_statements.max`-Parameter ist statisch. Sie müssen Ihre DB-Instance neu starten, um Änderungen an diesem Parameter zu übernehmen.

Deaktivieren des pg_stat_statements.track_utility-Parameters

Sie können die pg_stat_statements-Ansicht analysieren, um festzustellen, welche Dienstprogrammbefehle die meisten Ressourcen verbrauchen, die von `pg_stat_statements` nachverfolgt werden.

Der `pg_stat_statements.track_utility`-Parameter steuert, ob das Modul Dienstprogrammbefehle nachverfolgt, zu denen alle Befehle außer SELECT, INSERT, UPDATE, DELETE und MERGE gehören. Dieser Parameter ist standardmäßig auf on festgelegt.

Wenn Ihre Anwendung beispielsweise viele Savepoint-Abfragen verwendet, die von Natur aus eindeutig sind, kann dies die Freigabe von Anweisungen erhöhen. Als Abhilfemaßnahme können Sie den `pg_stat_statements.track_utility`-Parameter deaktivieren, sodass Savepoint-Abfragen nicht mehr von `pg_stat_statements` nachverfolgt werden.

> **Note**
>
> Der `pg_stat_statements.track_utility`-Parameter ist ein dynamischer Parameter. Sie können dessen Wert ändern, ohne Ihre Datenbank-Instance neu starten zu müssen.

Example Beispiel für eindeutige Savepoint-Abfragen in pg_stat_statements

```
                    query                          |       queryid
---------------------------------------------------+---------------------
 SAVEPOINT JDBC_SAVEPOINT_495701                   | -7249565344517699703
 SAVEPOINT JDBC_SAVEPOINT_1320                     | -1572997038849006629
 SAVEPOINT JDBC_SAVEPOINT_26739                    |  54791337410474486
 SAVEPOINT JDBC_SAVEPOINT_1294466                  |  8170064357463507593
 ROLLBACK TO SAVEPOINT JDBC_SAVEPOINT_65016        | -33608214779996400
 SAVEPOINT JDBC_SAVEPOINT_14185                    | -2175035613806809562
 SAVEPOINT JDBC_SAVEPOINT_45837                    | -6201592986750645383
 SAVEPOINT JDBC_SAVEPOINT_1324                     |  6388797791882029332
```

PostgreSQL 17 führt mehrere Verbesserungen für die Nachverfolgung von Dienstprogrammbefehlen ein:

- Savepoint-Namen werden jetzt als Konstanten angezeigt.
- Globale Transaktions-IDs (GIDs) von zweiphasigen Commit-Befehlen werden jetzt als Konstanten angezeigt.
- Die Namen von DEALLOCATE-Anweisungen werden als Konstanten angezeigt.
- CALL-Parameter werden jetzt als Konstanten angezeigt.

Timeout:PgSleep

Das `Timeout:PgSleep`-Ereignis tritt ein, wenn ein Serverprozess die `pg_sleep`-Funktion aufgerufen hat und auf das Ablaufen des Sleep-Timeouts wartet.

Themen

- Unterstützte Engine-Versionen
- Wahrscheinliche Ursachen für erhöhte Wartezeiten
- Aktionen

Unterstützte Engine-Versionen

Diese Warteereignisinformationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Dieses Wait-Ereignis tritt auf, wenn eine Anwendung, eine gespeicherte Funktion oder ein Benutzer eine SQL-Anweisung ausgibt, die eine der folgenden Funktionen aufruft:

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

Die vorhergehenden Funktionen verzögern die Ausführung, bis die angegebene Anzahl von Sekunden verstrichen ist. Beispielsweise pausiert `SELECT pg_sleep(1)` für 1 Sekunde. Weitere Informationen finden Sie unter Ausführung verzögern in der PostgreSQL-Dokumentation.

Aktionen

Identifizieren Sie die Anweisung, die die `pg_sleep`-Funktion ausgeführt hat. Überprüfen Sie, ob die Verwendung der Funktion angemessen ist.

Optimierung von Aurora PostgreSQL mit proaktiven Einblicken von Amazon DevOps Guru

Proaktive DevOps-Guru-Einblicke erkennen Zustände auf Ihren Aurora-PostgreSQL-DB-Clustern, die Probleme verursachen können, und informiert Sie darüber, bevor diese auftreten. Proaktive Einblicke können Sie auf eine seit langem inaktive Transaktionsverbindung aufmerksam machen. Weitere Informationen zur Fehlerbehebung von seit langem inaktiven Transaktionsverbindungen finden Sie unter Die Datenbank läuft seit langem inaktiv in Transaktionsverbindung.

DevOps Guru kann Folgendes tun:

- Vermeiden vieler häufig auftretender Datenbankprobleme durch Abgleich der Datenbankkonfiguration mit den allgemein empfohlenen Einstellungen.
- Warnen vor kritischen Problemen in der Flotte, die, wenn sie nicht überprüft werden, später zu größeren Problemen führen können.

- Benachrichtigung bei neu erkannten Problemen.

Jeder proaktive Einblick beinhaltet eine Analyse der Problemursache und Empfehlungen für Korrekturmaßnahmen.

Weitere Informationen zu Amazon DevOps Guru für Amazon RDS finden Sie unter Analysieren von Aurora-Leistungsanomalien mit Amazon DevOps Guru für Amazon RDS.

Die Datenbank läuft seit langem inaktiv in Transaktionsverbindung

Eine Verbindung zur Datenbank befindet sich sein mehr als 1800 Sekunden im Status `idle in transaction`.

Themen

- Unterstützte Engine-Versionen
- Kontext
- Mögliche Ursachen für dieses Problem
- Aktionen
- Relevante Metriken

Unterstützte Engine-Versionen

Diese Einblick-Informationen werden für alle Versionen von Aurora PostgreSQL unterstützt.

Kontext

Eine Transaktion im Status `idle in transaction` kann Sperren enthalten, die andere Abfragen blockieren. Sie kann auch verhindern, dass `VACUUM` (einschließlich Autovacuum) tote Zeilen bereinigt, was zu einer Überlastung von Index oder Tabellen oder zu einem Wraparound der Transaktions-ID führt.

Mögliche Ursachen für dieses Problem

Eine Transaktion, die in einer interaktiven Sitzung mit BEGIN oder START TRANSACTION initiiert wurde, wurde nicht mit einem COMMIT-, ROLLBACK- oder END-Befehl beendet. Dadurch wird die Transaktion in den Status `idle in transaction` versetzt.

Aktionen

Sie können ungenutzte Transaktionen finden, indem Sie `pg_stat_activity` abfragen.

Führen Sie in Ihrem SQL-Client die folgende Abfrage aus, um alle Verbindungen im Status `idle in transaction` aufzulisten und sie nach Dauer zu sortieren:

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

Abhängig von den Ursachen Ihres Einblicks empfehlen wir verschiedene Aktionen.

Themen

- Transaktion beenden
- Die Verbindung beenden
- Konfigurieren Sie den Parameter idle_in_transaction_session_timeout
- Überprüfen Sie den AUTOCOMMIT-Status
- Überprüfen Sie die Transaktionslogik in Ihrem Anwendungscode

Transaktion beenden

Wenn Sie eine Transaktion in einer interaktiven Sitzung mit BEGIN oder START TRANSACTION initiieren, wechselt sie in den Status `idle in transaction`. Sie verbleibt in diesem Status, bis Sie die Transaktion beenden, indem Sie einen COMMIT-, ROLLBACK-, END-Befehl ausführen oder die Verbindung vollständig trennen, um die Transaktion rückgängig zu machen.

Die Verbindung beenden

Beenden Sie die Verbindung mit einer inaktiven Transaktion mit der folgenden Abfrage:

```
SELECT pg_terminate_backend(pid);
```

pid ist die Prozess-ID der Verbindung.

Konfigurieren Sie den Parameter idle_in_transaction_session_timeout

Stellen Sie den Parameter `idle_in_transaction_session_timeout` in der Parametergruppe ein. Der Vorteil der Konfiguration dieses Parameters besteht darin, dass kein manueller Eingriff erforderlich ist, um die lang dauernde inaktive Transaktion zu beenden. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation.

Die folgende Meldung wird in der PostgreSQL-Protokolldatei angezeigt, nachdem die Verbindung beendet wurde, wenn sich eine Transaktion länger als die angegebene Zeit im Status „idle_in_transaction“ befindet.

```
FATAL: terminating connection due to idle in transaction timeout
```

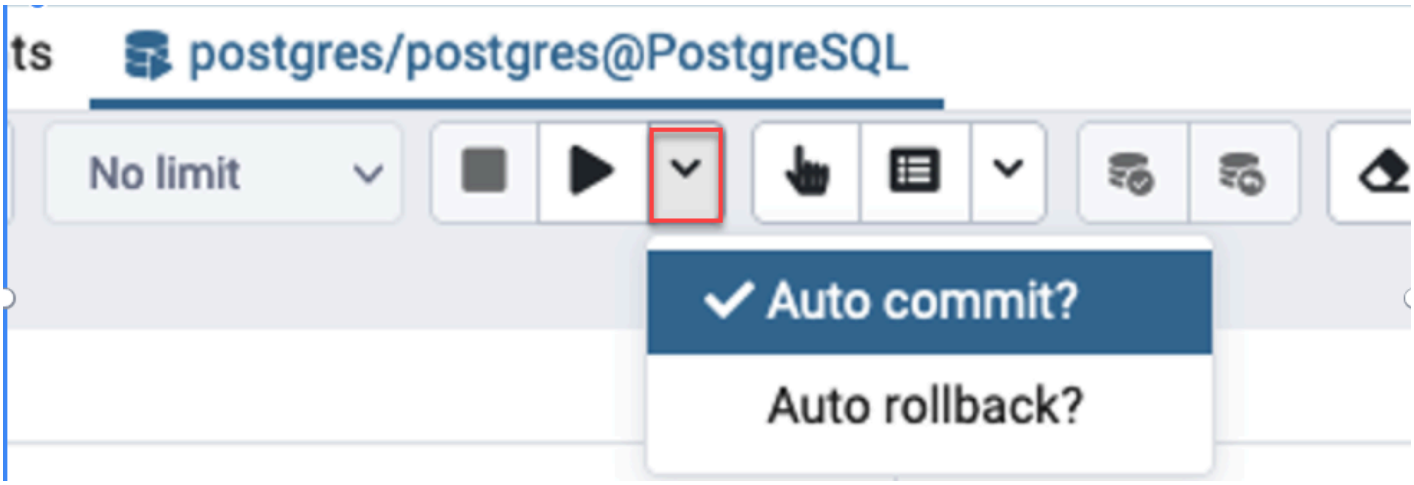
Überprüfen Sie den AUTOCOMMIT-Status

AUTOCOMMIT ist standardmäßig aktiviert. Wenn dies jedoch versehentlich im Client deaktiviert wurde, stellen Sie sicher, dass Sie es wieder aktivieren.

- Führen Sie in Ihrem psql-Client den folgenden Befehl aus:

  ```
  postgres=> \set AUTOCOMMIT on
  ```

- Schalten Sie dies in pgadmin ein, indem Sie die Option AUTOCOMMIT über den Abwärtspfeil wählen.



Überprüfen Sie die Transaktionslogik in Ihrem Anwendungscode

Untersuchen Sie Ihre Anwendungslogik auf mögliche Probleme. Berücksichtigen Sie die folgenden Aktionen:

- Prüfen Sie, ob das JDBC-Autocommit in Ihrer Anwendung auf „true“ gesetzt ist. Erwägen Sie auch, explizite `COMMIT`-Befehle in Ihrem Code zu verwenden.
- Überprüfen Sie Ihre Fehlerbehandlungslogik, um festzustellen, ob eine Transaktion nach einem Fehler geschlossen wird.
- Prüfen Sie, ob Ihre Anwendung lange braucht, um die von einer Abfrage zurückgegebenen Zeilen zu verarbeiten, während die Transaktion geöffnet ist. Wenn dies der Fall ist, erwägen Sie, die Anwendung so zu codieren, dass die Transaktion geschlossen wird, bevor die Zeilen verarbeitet werden.
- Prüfen Sie, ob eine Transaktion viele lang laufende Operationen enthält. Wenn dies der Fall ist, teilen Sie eine einzelne Transaktion in mehrere Transaktionen auf.

Relevante Metriken

Die folgenden PI-Metriken beziehen sich auf diesen Einblick:

- idle_in_transaction_count – Anzahl der Sitzungen im Status `idle in transaction`.
- idle_in_transaction_max_time – Die Dauer der am längsten laufenden Transaktion im Status `idle in transaction`.

Bewährte Methoden mit Amazon Aurora PostgreSQL

Im Folgenden finden Sie einige bewährte Methoden für die Verwaltung Ihres Amazon-Aurora-PostgreSQL-DB-Cluster. Überprüfen Sie auch die grundlegenden Wartungsaufgaben. Weitere Informationen finden Sie unter Leistung und Skalierung für Amazon Aurora PostgreSQL.

Themen

- Vermeiden von Leistungseinbußen, automatischem Neustart und Failover für DB-Instances von Aurora PostgreSQL
- Diagnostizieren einer Überlastung von Tabellen und Indizes
- Verbesserte Arbeitsspeicherverwaltung in Aurora PostgreSQL
- Schnelles Failover mit Amazon Aurora PostgreSQL
- Schnelle Wiederherstellung nach Failover mit der Cluster-Cacheverwaltung für Aurora PostgreSQL
- Verwalten von Aurora PostgreSQL Verbindungsabwanderung mit Pooling
- Behandeln von inaktiven Verbindungen in PostgreSQL

- Ändern von Speicherparametern für Aurora PostgreSQL
- Verwendung von CloudWatch Amazon-Metriken zur Analyse der Ressourcennutzung für Aurora PostgreSQL
- Verwenden der logischen Replikation, um ein Hauptversions-Upgrade für Aurora PostgreSQL durchzuführen
- Fehlerbehebung bei Speicherproblemen in Aurora PostgreSQL

Vermeiden von Leistungseinbußen, automatischem Neustart und Failover für DB-Instances von Aurora PostgreSQL

Wenn Sie hohe Workloads ausführen, die über die zugewiesenen Ressourcen Ihrer DB-Instance hinausgehen, können Sie die Ressourcen, auf denen Sie Ihre Anwendung und die Aurora-Datenbank ausführen, aufbrauchen. Wenn Sie Metriken für Ihre Datenbank-Instance wie CPU-Auslastung, Speichernutzung und Anzahl der verwendeten Datenbankverbindungen abrufen möchten, können Sie auf die von Amazon CloudWatch bereitgestellten Metriken, auf Performance-Insights und Enhanced Monitoring zurückgreifen. Informationen zur Überwachung Ihrer DB-Instance finden Sie unter Überwachung von Metriken in einem Amazon-Aurora-Cluster.

Wenn Ihre Workload die von Ihnen verwendeten Ressourcen aufbraucht, wird Ihre DB-Instance möglicherweise langsamer, neu gestartet oder es wird sogar ein Failover auf eine andere DB-Instance durchgeführt. Dies können Sie vermeiden, indem Sie Ihre Ressourcenauslastung überwachen, die Workload, die auf Ihrer DB-Instance ausgeführt wird, untersuchen und gegebenenfalls Optimierungen vornehmen. Wenn Optimierungen keine Verbesserungen der Instance-Metriken und keine Verringerung der Ressourcenauslastung ergeben, sollten Sie erwägen, Ihre DB-Instance hochzuskalieren, bevor diese an ihre Grenzen stößt. Weitere Hinweise zu verfügbaren DB-Instance-Klassen und ihren Spezifikationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Diagnostizieren einer Überlastung von Tabellen und Indizes

Sie können PostgreSQL Multiversion Concurrency Control (MVCC) verwenden, um die Datenintegrität zu wahren. PostgreSQL MVCC funktioniert, indem es eine interne Kopie aktualisierter oder gelöschter Zeilen (auch Tupel genannt) speichert, bis für eine Transaktion entweder ein Commit oder ein Rollback ausgeführt wird. Diese gespeicherte interne Kopie ist für Benutzer nicht sichtbar. Wenn diese nicht sichtbaren Kopien nicht regelmäßig von den Dienstprogrammen VACUUM oder AUTOVACUUM bereinigt werden, kann es jedoch zu einer Überlastung der Tabelle kommen. Wenn

diese Option nicht aktiviert ist, kann die Überlastung der Tabelle zu erhöhten Speicherkosten führen und Ihre Verarbeitungsgeschwindigkeit verlangsamen.

In vielen Fällen reichen die Standardeinstellungen für VACUUM oder AUTOVACUUM in Aurora aus, um eine ungewollte Überlastung von Tabellen zu vermeiden. Möglicherweise möchten Sie jedoch überprüfen, ob eine Überlastung vorliegt, wenn in Ihrer Anwendung die folgenden Bedingungen vorliegen:

- Verarbeitet eine große Anzahl von Transaktionen in relativ kurzer Zeit zwischen VACUUM-Prozessen.
- Funktioniert schlecht und der Speicherplatz ist knapp.

Sammeln Sie zunächst möglichst genaue Informationen darüber, wie viel Speicherplatz tote Tupel belegen und wie viel Sie wiederherstellen können, indem Sie die Überlastung der Tabellen und Indizes bereinigen. Verwenden Sie dazu die `pgstattuple`-Erweiterung, um Statistiken über Ihren Aurora-Cluster zu sammeln. Weitere Informationen finden Sie unter pgstattuple. Die Berechtigungen zum Verwenden der `pgstattuple`-Erweiterung sind auf die Rolle `pg_stat_scan_tables` und die Datenbank-Superuser beschränkt.

Um die `pgstattuple`-Erweiterung in Aurora zu erstellen, verbinden Sie eine Client-Sitzung mit dem Cluster, z. B. psql oder pgAdmin, und verwenden Sie den folgenden Befehl:

```
CREATE EXTENSION pgstattuple;
```

Erstellen Sie die Erweiterung in jeder Datenbank, für die Sie ein Profil erstellen möchten. Verwenden Sie nach dem Erstellen der Erweiterung die Befehlszeilenschnittstelle (CLI), um zu messen, wie viel unbenutzbaren Speicherplatz Sie zurückgewinnen können. Bevor Sie Statistiken sammeln, ändern Sie die Cluster-Parametergruppe, indem Sie AUTOVACUUM auf 0 festlegen. Die Einstellung 0 verhindert, dass Aurora automatisch alle toten Tupel bereinigt, die Ihre Anwendung hinterlassen hat, was sich auf die Genauigkeit der Ergebnisse auswirken kann. Verwenden Sie den folgenden Befehl, um eine einfache Tabelle zu erstellen:

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
SELECT 100001
```

Im folgenden Beispiel führen wir die Abfrage mit aktiviertem AUTOVACUUM für den DB-Cluster aus. Der Wert `dead_tuple_count` ist 0, was bedeutet, dass AUTOVACUUM die veralteten Daten oder Tupel aus der PostgreSQL-Datenbank gelöscht hat.

Wenn Sie Informationen über die Tabelle mit `pgstattuple` sammeln möchten, geben Sie in der Abfrage den Namen einer Tabelle oder eine Objektkennung (OID) an:

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                  | 16616      | 0.46
(1 row)
```

In der folgenden Abfrage schalten wir AUTOVACUUM aus und geben einen Befehl ein, mit dem 25 000 Zeilen aus der Tabelle gelöscht werden. Infolgedessen steigt der Wert `dead_tuple_count` auf 25 000.

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;

DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

```
table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
 | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
(1 row)
```

Starten Sie einen VACUUM-Prozess, um diese toten Tupel zurückzugewinnen.

Beobachten von Überlastungen ohne Unterbrechung Ihrer Anwendung

Die Einstellungen in einem Aurora-Cluster sind optimiert, um die bewährten Methoden für die meisten Workloads bereitzustellen. Möglicherweise möchten Sie jedoch einen Cluster optimieren, damit er besser zu Ihren Anwendungen und Nutzungsmustern passt. In diesem Fall können Sie die `pgstattuple`-Erweiterung verwenden, ohne eine aktive Anwendung zu unterbrechen. Führen Sie dazu die folgenden Schritte aus:

1. Klonen Sie Ihre Aurora-Instance.
2. Ändern Sie die Parameterdatei, um AUTOVACUUM im Klon zu deaktivieren.
3. Führen Sie eine `pgstattuple`-Abfrage durch, während Sie den Klon mit einem Beispiel-Workload oder mit pgbench testen, einem Programm zum Ausführen von Benchmark-Tests in PostgreSQL. Weitere Informationen finden Sie unter pgbench.

Nachdem Sie Ihre Anwendungen ausgeführt und das Ergebnis angezeigt haben, verwenden Sie pg_repack oder VACUUM FULL für die wiederhergestellte Kopie und vergleichen Sie die Unterschiede. Wenn Sie einen deutlichen Rückgang der Werte dead_tuple_count, dead_tuple_len oder dead_tuple_percent feststellen, passen Sie den Bereinigungsplan Ihres Produktions-Clusters an, um die Überlastung zu minimieren.

Vermeiden von Überlastung in temporären Tabellen

Wenn Ihre Anwendung temporäre Tabellen erstellt, vergewissern Sie sich, dass Ihre Anwendung diese temporären Tabellen entfernt, wenn sie nicht mehr benötigt werden. Automatische Bereinigungsprozesse finden keine temporären Tabellen. Wenn diese Option nicht aktiviert ist, können temporäre Tabellen schnell zu einer Überlastung der Datenbank führen. Darüber hinaus kann sich die Überlastung auf die Systemtabellen ausweiten. Hierbei handelt es sich um interne Tabellen, die PostgreSQL-Objekte und -Attribute verfolgen, wie pg_attribute und pg_depend.

Wenn eine temporäre Tabelle nicht mehr benötigt wird, können Sie eine TRUNCATE-Anweisung verwenden, um die Tabelle zu leeren und Speicherplatz freizugeben. Bereinigen Sie dann die Tabellen pg_attribute und pg_depend manuell. Durch das Bereinigen dieser Tabellen wird sichergestellt, dass durch das kontinuierliche Erstellen und Kürzen/Löschen temporärer Tabellen keine Tupel hinzugefügt werden und das System nicht überlastet wird.

Sie können dieses Problem beim Erstellen einer temporären Tabelle vermeiden, indem Sie die folgende Syntax verwenden, die die neuen Zeilen löscht, wenn für Inhalt ein Commit ausgeführt wird:

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

Die `ON COMMIT DELETE ROWS`-Klausel kürzt die temporäre Tabelle, wenn für die Transaktion ein Commit ausgeführt wird.

Vermeiden von Überlastung in Indizes

Wenn Sie ein indiziertes Feld in einer Tabelle ändern, führt die Indexaktualisierung zu einem oder mehreren toten Tupeln in diesem Index. Standardmäßig entfernt der automatische Bereinigungsprozess überflüssige Indizes, aber diese Bereinigung erfordert einen erheblichen Zeit- und Ressourcenaufwand. Um die Einstellungen für die Indexbereinigung beim Erstellen einer Tabelle anzugeben, schließen Sie die Klausel vacuum_index_cleanup ein. Standardmäßig ist die Klausel bei der Tabellenerstellung auf AUTO gesetzt, was bedeutet, dass der Server entscheidet, ob Ihr Index bereinigt werden muss, wenn er die Tabelle bereinigt. Sie können die Klausel auf EIN einstellen, um die Indexbereinigung für eine bestimmte Tabelle zu aktivieren, oder auf AUS, um die Indexbereinigung für diese Tabelle zu deaktivieren. Denken Sie daran, dass das Deaktivieren der Indexbereinigung zwar Zeit sparen kann, aber möglicherweise zu einer Überlastung des Index führen kann.

Sie können die Indexbereinigung manuell steuern, wenn Sie eine Tabelle über die Befehlszeile bereinigen. Wenn Sie eine Tabelle bereinigen und tote Tupel aus den Indizes entfernen möchten, schließen Sie die INDEX_CLEANUP-Klausel mit dem Wert EIN und dem Tabellennamen ein:

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

Wenn Sie eine Tabelle bereinigen möchten, ohne die Indizes zu bereinigen, geben Sie den Wert AUS an:

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

Verbesserte Arbeitsspeicherverwaltung in Aurora PostgreSQL

Aurora PostgreSQL bietet jetzt erweiterte Speicherverwaltungsfunktionen zur Optimierung der Datenbankleistung und Ausfallsicherheit bei unterschiedlichen Workloads. Diese Verbesserungen helfen Aurora PostgreSQL dabei, eine konsistente Verfügbarkeit und Reaktionsfähigkeit aufrechtzuerhalten, selbst in Zeiten mit hohem Speicherbedarf.

Diese Funktion ist in den folgenden Aurora-PostgreSQL-Versionen für bereitgestellte Instances verfügbar und standardmäßig aktiviert:

- 15.3 und alle höheren Nebenversionen
- 14.8 und höhere Nebenversionen
- 13.11 und höhere Nebenversionen
- 12.15 und höhere Nebenversionen
- 11.20 und höhere Nebenversionen

Diese Funktion ist in den folgenden Aurora PostgreSQL-Versionen für Aurora Serverless-Instances verfügbar und standardmäßig aktiviert:

- 16.3 und alle höheren Nebenversionen
- 15.7 und alle höheren Nebenversionen
- 14.12 und höhere Nebenversionen
- 13.15 und höhere Nebenversionen

Wenn Kunden-Workloads den gesamten verfügbaren freien Arbeitsspeicher verbrauchen, kann das Betriebssystem die Datenbank zum Schutz der Ressourcen neu starten, was zu einer vorübergehenden Nichtverfügbarkeit führt. Die neuen Verbesserungen der Speicherverwaltung in Aurora PostgreSQL brechen bestimmte Transaktionen proaktiv ab, wenn das System unter hohem Arbeitsspeicherdruck steht, und tragen so zur Aufrechterhaltung der Datenbankstabilität bei.

Die wichtigsten Features der verbesserten Speicherverwaltung lauten wie folgt:

- Datenbanktransaktionen, die mehr Arbeitsspeicher anfordern, werden abgebrochen, wenn sich das System einem kritischen Arbeitsspeicherdruck nähert.
- Das System steht unter kritischem Arbeitsspeicherdruck, wenn der gesamte physische Arbeitsspeicher bereits ausgeschöpft und auch der Swapspeicher nahezu ausgeschöpft ist. Unter

diesen Umständen werden alle Transaktionen abgebrochen, die Speicher anfordern, um den Arbeitsspeicherdruck in der DB-Instance sofort zu reduzieren.

- Wichtige PostgreSQL-Launcher und Hintergrund-Worker wie Autovacuum Worker sind immer geschützt.

Verarbeiten von Parametern zur Speicherverwaltung

So aktivieren Sie die Speicherverwaltung

Diese Option ist standardmäßig aktiviert. Eine Fehlermeldung wird angezeigt, wenn eine Transaktion aufgrund von zu wenig Arbeitsspeicher abgebrochen wird, wie im folgenden Beispiel gezeigt:

```
ERROR: out of memory Detail: Failed on request of size 16777216.
```

So deaktivieren Sie die Speicherverwaltung

Zum Deaktivieren dieser Funktion stellen Sie mit psql eine Verbindung mit dem DB-Cluster von Aurora PostgreSQL her und verwenden Sie die SET-Anweisung für die Parameterwerte, wie unten beschrieben.

> **Note**
>
> Wir empfehlen, die Speicherverwaltung aktiviert zu lassen. Auf diese Weise können potenzielle Speicherprobleme vermieden werden, die aufgrund von Speichererschöpfung zu Workload-bedingten Datenbankneustarts führen können.

Die folgende Tabelle zeigt, wie die Speicherverwaltungsfunktion für die verschiedenen Aurora-PostgreSQL-Versionen deaktiviert wird:

Aurora-PostgreSQL-Versionen	Parameter	Standard	Befehl zum Deaktivieren der Speicherverwaltung auf Sitzungsebene
11.20, 11.21, 12.15, 12.16, 13.11, 13.12, 14.8, 14.9, 15.3, 15.4	`rds.memory_allocation_guard`	`false`	`SET rds.memory_allocat`

Aurora-PostgreSQL-Versionen	Parameter	Standard	Befehl zum Deaktivieren der Speicherverwaltung auf Sitzungsebene
			`ion_guard = true;`
12.17, 13.13, 14.10, 15.5 und höhere Versionen	`rds.enable_memory_management`	`true`	`SET rds.enable_memory_management = false;`

> **Note**
>
> Der Parameter `rds.memory_allocation_guard` wurde in Aurora PostgreSQL 12.17, 13.13, 14.10, 15.5 und höheren Versionen als veraltet gekennzeichnet.

Wenn die Werte dieser Parameter in der Parametergruppe des DB-Clusters festgelegt werden, können die Abfragen nicht abgebrochen werden. Weitere Informationen zu DB-Cluster-Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Einschränkung

- Diese Funktion wird in der Instance-Klasse db.t3.medium nicht unterstützt.

Schnelles Failover mit Amazon Aurora PostgreSQL

Im Folgenden erfahren Sie, wie Sie sicherstellen können, dass ein Failover so schnell wie möglich erfolgt. Zur schnellen Wiederherstellung nach dem Failover können Sie die Cluster-Cache-Verwaltung für Ihren Aurora-PostgreSQL-DB-Cluster verwenden. Weitere Informationen finden Sie unter Schnelle Wiederherstellung nach Failover mit der Cluster-Cacheverwaltung für Aurora PostgreSQL.

Zu den Schritten, die Sie ausführen können, um ein schnelles Failover durchzuführen, gehören die folgenden:

- Setzen Sie Transmission Control Protocol (TCP)-Keepalives mit kurzen Zeitrahmen, um länger laufende Abfragen zu stoppen, bevor das Lese-Timeout abläuft, wenn ein Fehler auftritt.
- Legen Sie strikte Timeouts für das DNS (Java Domain Name System) fest. Auf diese Weise wird sichergestellt, dass der schreibgeschützte Aurora-Endpunkt bei späteren Verbindungsversuchen ordnungsgemäß die schreibgeschützten Knoten durchlaufen kann.
- Legen Sie die Timeout-Variablen, die in der JDBC-Verbindungszeichenfolge verwendet werden, auf einen möglichst niedrigen Wert fest. Setzen Sie für kurze und lange dauernde Abfragen unterschiedliche Verbindungsobjekte ein.
- Verwenden Sie die bereitgestellten Aurora-Endpunkte für Lese- und Schreibvorgänge, um eine Verbindung zum Cluster herzustellen.
- Verwenden Sie RDS-API-Operationen, um die Anwendungsreaktion bei serverseitigen Ausfällen zu testen. Testen Sie die Anwendungsreaktionen auf clientseitige Ausfälle mithilfe eines Packet-Dropping-Tool.
- Verwenden Sie den AWS-JDBC-Treiber, um die Failover-Fähigkeiten von Aurora PostgreSQL voll auszuschöpfen. Weitere Informationen zum AWS-JDBC-Treiber und vollständige Anleitungen zur Verwendung finden Sie im GitHub-Repository für den JDBC-Treiber von Amazon Web Services (AWS).

Diese werden im Folgenden ausführlicher erläutert.

Themen

- Einstellen von TCP-Keepalive-Parametern
- Konfigurieren Ihrer Anwendung für schnelles Failover
- Testen eines Failovers
- Beispiel für schnelles Java-Failover

Einstellen von TCP-Keepalive-Parametern

Wenn Sie eine TCP-Verbindung einrichten, wird ein Satz von Timern mit der Verbindung verknüpft. Wenn der Keepalive-Timer den Wert null erreicht, wird ein Keepalive-Prüfpaket an den Endpunkt der Verknüpfung gesendet. Wenn das Prüfpaket eine Antwort erhält, können Sie davon ausgehen, dass die Verbindung nach wie vor aktiv ist.

Indem Sie TCP-Keepalive-Parameter aktivieren und auf strikte Werte festlegen, können Sie sicherstellen, dass im Falle, dass der Client keine Verbindung zur Datenbank herstellen kann, alle

aktiven Verbindungen schnell beendet werden. Die Anwendung kann dann eine Verbindung zu einem neuen Endpunkt herstellen.

Die folgenden TCP-Keepalive-Parameter müssen festgelegt werden:

- `tcp_keepalives_idle` gibt die Zeitspanne in Sekunden an, nach der ein Keepalive-Paket gesendet wird, wenn keine Daten vom Socket gesendet wurden. ACKs werden nicht als Daten betrachtet. Wir empfehlen die folgende Einstellung:

  ```
  tcp_keepalives_idle = 1
  ```

- `tcp_keepalives_interval` gibt die Zeitspanne in Sekunden an, die zwischen dem ersten gesendeten Paket und dem Senden von nachfolgenden Keepalive-Paketen verstreicht. Stellen Sie diese Zeit ein, indem Sie die `tcp_keepalives_idle`-Parameter verwenden. Wir empfehlen die folgende Einstellung:

  ```
  tcp_keepalives_interval = 1
  ```

- `tcp_keepalives_count` gibt die Anzahl von unbeantworteten Keepalive-Prüfpaketen an, nach denen die Anwendung benachrichtigt wird. Wir empfehlen die folgende Einstellung:

  ```
  tcp_keepalives_count = 5
  ```

Mit diesen Einstellungen sollte die Anwendung innerhalb von fünf Sekunden benachrichtigt werden, wenn die Datenbank nicht mehr antwortet. Falls die Keepalive-Pakete innerhalb des Anwendungsnetzwerks häufig verworfen werden, können Sie einen höheren `tcp_keepalives_count`-Wert angegeben. Dies ermöglicht mehr Puffer in weniger zuverlässigen Netzwerken, erhöht jedoch die Zeit, die benötigt wird, um einen tatsächlichen Ausfall zu erkennen.

Einrichten von TCP-Keepalive-Parametern in Linux

1. Testen Sie, wie Sie Ihre TCP-Keealive-Parameter konfigurieren.

   Wir empfehlen, dies über die Befehlszeile mit den folgenden Befehlen zu tun. Diese vorgeschlagene Konfiguration ist systemweit. Mit anderen Worten, es wirkt sich auch auf alle anderen Anwendungen aus, die Sockets mit der `SO_KEEPALIVE`-Option erstellen.

   ```
   sudo sysctl net.ipv4.tcp_keepalive_time=1
   sudo sysctl net.ipv4.tcp_keepalive_intvl=1
   sudo sysctl net.ipv4.tcp_keepalive_probes=5
   ```

2. Nachdem Sie eine geeignete Konfiguration für Ihre Anwendung gefunden haben, schreiben Sie diese Einstellungen fest, indem Sie die folgenden Zeilen zu `/etc/sysctl.conf` hinzufügen, einschließlich aller vorgenommenen Änderungen:

```
tcp_keepalive_time = 1
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

Konfigurieren Ihrer Anwendung für schnelles Failover

Im Folgenden finden Sie eine Beschreibung verschiedener Konfigurationsänderungen für Aurora PostgreSQL, die Sie für ein schnelles Failover vornehmen können. Weitere Informationen zur Einrichtung und Konfiguration des PostgreSQL-JDBC-Treibers finden Sie in der Dokumentation zum PostgreSQL-JDBC-Treiber.

Themen

- Reduzieren von DNS-Cache-Timeouts
- Einstellen einer Aurora PostgreSQL-Verbindungszeichenfolge für ein schnelles Failover
- Weitere Optionen zum Abrufen der Hostzeichenfolge

Reduzieren von DNS-Cache-Timeouts

Wenn Ihre Anwendung versucht, eine Verbindung nach einem Failover aufzubauen, wird der neue Aurora-PostgreSQL-Writer ein früherer Reader sein. Sie können ihn finden, indem Sie den schreibgeschützten Aurora-Endpunkt verwenden, bevor DNS-Updates vollständig verbreitet wurden. Den Java DNS-TTL–(time to live)-Wert niedrig festzulegen, z. B. unter 30 Sekunden, sorgt dafür, dass bei späteren Verbindungsversuchen die Reader-Knoten durchlaufen werden können.

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

Einstellen einer Aurora PostgreSQL-Verbindungszeichenfolge für ein schnelles Failover

Für die Verwendung eines schnellen Failover in Aurora PostgreSQL stellen Sie sicher, dass die Verbindungszeichenfolge Ihrer Anwendung eine Liste mit Hosts statt einen einzelnen Host enthält.

Im Folgenden eine beispielhafte Verbindungszeichenfolge, die Sie verwenden können, um sich mit einem Aurora-PostgreSQL-Cluster zu verbinden. In diesem Beispiel sind die Hosts fett dargestellt.

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=primary
```

Für die bestmögliche Verfügbarkeit und um eine Abhängigkeit von der RDS-API zu vermeiden, empfehlen wir, eine Datei zu pflegen, mit der Sie eine Verbindung herstellen können. Diese Datei enthält eine Hostzeichenfolge, aus der die Anwendung Daten lesen kann, wenn Sie eine Verbindung zur Datenbank herstellen. Die Hostzeichenfolge enthält alle für das Cluster verfügbaren Aurora-Endpunkte. Weitere Informationen zu Aurora-Endpunkten finden Sie unter Amazon-Aurora-Endpunktverbindungen.

Beispielsweise können Sie Ihre Endpunkte wie folgt in einer lokalen Datei speichern.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

Ihre Anwendung liest aus dieser Datei. um den Hostdatenabschnitt der JDBC-Verbindungzeichenfolge mit Daten füllen. Das Umbenennen des DB-Clusters bewirkt eine Änderung dieser Endpunkte. Stellen Sie sicher, dass Ihre Anwendung dieses Ereignis behandelt, falls es auftritt.

Eine weitere Option ist die Verwendung einer Liste mit DB-Instance-Knoten wie folgt.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

Der Vorteil dieses Ansatzes besteht darin, dass der PostgreSQL-JDBC-Verbindungstreiber alle Knoten dieser Liste in einer Schleife durchläuft, um eine gültige Verbindung zu finden. Im Gegensatz dazu werden bei Verwendung der Aurora-Endpunkte bei jedem Verbindungsversuch nur zwei Knoten ausprobiert. Die Verwendung von DB-Instance-Knoten hat jedoch einen Nachteil. Wenn Sie Knoten zum Cluster hinzufügen oder daraus entfernen und dadurch die Liste der Instance-Endpunkte nicht mehr aktuell ist, findet der Verbindungstreiber möglicherweise keinen geeigneten Host mehr für einen Verbindungsaufbau.

Um sicherzustellen, dass die Anwendung mit dem Verbindungsaufbau zu einem Host nicht zu lange wartet, legen Sie für die folgenden Parameter strikte Werte fest:

- `targetServerType` – Steuert, ob der Treiber eine Verbindung zu einem Schreib- oder Leseknoten herstellt. Um sicherzustellen, dass Ihre Anwendungen nur eine Verbindung zu einem Schreibknoten herstellen, legen Sie den `targetServerType` auf `primary` fest.

  Werte für den Parameter `targetServerType` sind `primary`, `secondary`, `any` und `preferSecondary`. Der `preferSecondary`-Wert versucht, zuerst die Verbindung zu einem Lese-Knoten herzustellen. Er stellt eine Verbindung zum Schreiber her, wenn keine Leser-Verbindung hergestellt werden kann.
- `loginTimeout` – Steuert, wie lange Ihre Anwendung auf die Anmeldung bei der Datenbank wartet, nachdem eine Socket-Verbindung hergestellt wurde.
- `connectTimeout` – Steuert, wie lange der Socket wartet, um eine Verbindung zur Datenbank herzustellen.

Sie können weitere Anwendungsparameter ändern, um den Verbindungsvorgang zu beschleunigen, anhängig davon, wie strikt die Anwendung sein soll:

- `cancelSignalTimeout` – In einigen Anwendungen soll möglicherweise ein Best-Effort-Abbruchsignal für eine abgelaufene Abfrage gesendet werden. Falls dieses Abbruchsignal in Ihrem Failover-Pfad ist, ziehen Sie dafür eine strikte Einstellung in Erwägung, damit dieses Signal nicht an inaktiven Host gesendet wird.
- `socketTimeout` – Dieser Parameter steuert, wie lange vom Socket auf Lesevorgänge gewartet wird. Er kann als globales Abfrage-Timeout genutzt werden, um sicherzustellen, dass eine Abfrage nicht länger als für diesen Wert wartet. Eine gute Praxis ist es, zwei Verbindungs-Handler zu haben. Ein Verbindungs-Handler führt kurzlebige Abfragen aus und legt diesen Wert niedriger fest. Bei einem anderen Verbindungs-Handler für Abfragen mit langer Ausführungsdauer ist dieser Wert viel höher festgelegt. Auf diese Weise können Sie sich sicher sein, dass TCP-Keepalive-Parameter im Falle eines Serverausfalls Abfragen beenden, die über eine lange Zeit ausgeführt werden.
- `tcpKeepAlive` – Aktivieren Sie diesen Parameter, um sicherzustellen, dass die festgelegten TCP-Keepalive-Parameter angewendet werden.
- `loadBalanceHosts` – Ist der Wert dieses Parameters auf `true` gesetzt, stellt die Anwendung die Verbindung zu einem zufällig aus der Liste der Host-Kandidaten ausgewählten Host her.

Weitere Optionen zum Abrufen der Hostzeichenfolge

Sie können die Hostzeichenfolge aus mehreren Quellen erhalten, einschließlich der `aurora_replica_status`-Funktion und durch Verwenden der Amazon-RDS-API.

In vielen Fällen müssen Sie bestimmen, wer Schreiber im Cluster ist oder um weitere Leseknoten im Cluster zu finden. Dazu kann sich Ihre Anwendung mit einer beliebigen DB-Instance im DB-Cluster verbinden und die `aurora_replica_status`-Funktion abfragen. Sie können diese Funktion verwenden, um die Dauer für das Auffinden eines Host für eine Verbindung zu reduzieren. In bestimmten Netzwerkausfallszenarien zeigt die `aurora_replica_status`-Funktion möglicherweise veraltete oder unvollständige Informationen an.

Eine gute Möglichkeit, um sicherzustellen, dass Ihre Anwendung einen Knoten findet, zu dem eine Verbindung hergestellt werden kann, besteht darin, zu versuchen, eine Verbindung zum Cluster-Schreiber-Endpunkt und dann zum Cluster-Reader-Endpunkt herzustellen. Sie tun dies, bis Sie eine lesbare Verbindung herstellen können. Diese Endpunkte ändern sich nur, wenn Sie Ihren DB-Cluster umbenennen. Sie können diese daher normalerweise als statische Elemente Ihrer Anwendung beibehalten oder in einer Ressourcendatei speichern, aus der Ihre Anwendung liest.

Nachdem Sie eine Verbindung zu einem dieser Endpunkte hergestellt haben, können Sie Informationen über den Rest des Clusters erhalten. Dazu rufen Sie die `aurora_replica_status`-Funktion auf. Der folgende Befehl ruft z. B. Informationen mit `aurora_replica_status` ab.

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
 now(), last_update_timestamp
FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now |
 last_update_timestamp
-----------+--------------------------------------+-----------------
+---------------------------+-------------------------------+-----------------------
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-3 | MASTER_SESSION_ID | | | 2017-03-07 19:50:24.695322+00 | 2017-03-07
 19:50:23+00
(3 rows)
```

Beispielsweise könnte der Host-Abschnitt Ihrer Verbindungszeichenfolge sowohl mit den Schreiber- als auch den Reader-Cluster-Endpunkten, wie im Folgenden dargestellt, beginnen.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

In diesem Szenario versucht die Anwendung, eine Verbindung zu einem beliebigen Knotentyp (primärer oder sekundärer) herzustellen. Wenn Ihre Anwendung verbunden ist, empfiehlt es sich, zuerst den Lese-/Schreibstatus des Knotens zu untersuchen. Fragen Sie dazu das Ergebnis des Befehls `SHOW transaction_read_only` ab.

Gibt die Abfrage den Wert `OFF` zurück, haben Sie erfolgreich eine Verbindung zum primären Knoten hergestellt. Angenommen, der Rückgabewert ist jedoch `ON` und Ihre Anwendung benötigt eine Lese-/Schreibverbindung. In diesem Fall können Sie die `aurora_replica_status`-Funktion aufrufen, um festzustellen, ob `server_id session_id='MASTER_SESSION_ID'` enthält. Diese Funktion gibt Ihnen den Namen des primären Knotens zurück. Sie können dies mit dem `endpointPostfix` verwenden, das wie folgt beschrieben ist.

Stellen Sie sicher, dass Sie wissen, wenn Sie eine Verbindung zu einem Replica herstellen, das veraltete Daten enthält. In diesem Fall zeigt die `aurora_replica_status` Funktion möglicherweise veraltete Informationen an. Sie können einen Schwellenwert für Alterung auf Anwendungsebene festlegen. Um dies zu überprüfen, können Sie den Unterschied zwischen der Serverzeit und dem `last_update_timestamp`-Wert betrachten. Im Allgemeinen sollte Ihre Anwendung es vermeiden, aufgrund widersprüchlicher Informationen, die von der Funktion `aurora_replica_status` zurückgegeben werden, zwischen zwei Hosts hin- und herzuschalten. Ihre Anwendung sollte zuerst alle bekannten Hosts ausprobieren, anstatt den von `aurora_replica_status` zurückgegebenen Daten zu folgen.

Java-Beispiel zum Auflisten von Instances, die den Vorgang DescribeDBClusters-API verwenden

Sie können programmgesteuert mit AWS SDK für Java nach der Instance-Liste suchen, genauer gesagt, mit dem Vorgang DescribeDBClusters-API.

Hier ist ein kleines Beispiel, wie Sie dies in Java 8 durchführen können.

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
   .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);

DBCluster singleClusterResult = result.getDBClusters().get(0);
```

```
String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
   .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
   .reversed()) // This puts the writer at the front of the list
   .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
 singleClusterResult.getPort()))
   .collect(Collectors.joining(","));
```

Hier enthält `pgJDBCEndpointStr` eine formatierte Liste mit Endpunkten, wie im Folgenden.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

Die Variable `endpointPostfix` kann eine Konstante sein, die Ihre Anwendung festlegt. Oder Ihre Anwendung kann es erhalten, indem sie den `DescribeDBInstances`-API-Vorgang für eine einzelne Instance in Ihrem Cluster abfragt. Dieser Wert bleibt konstant innerhalb einer AWS-Region und für einen einzelnen Kunden. Es wird also ein API-Aufruf gespeichert, um diese Konstante in einer Ressourcendatei zu speichern, aus der Ihre Anwendung liest. Im vorherigen Beispiel ist es auf den folgenden festgelegt.

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

Eine gute Methode zur Gewährleistung der Verfügbarkeit ist die standardmäßige Nutzung der Aurora-Endpunkte Ihres DB-Clusters für den Fall, dass die API nicht oder zu langsam reagiert. Diese Endpunkte sind garantiert auf dem aktuellen Stand – im Zeitraum, der für das Update eines DNS-Datensatzes benötigt wird. Das Aktualisieren des DNS-Eintrags mit einem Endpunkt dauert in der Regel weniger als 30 Sekunden. Sie können den Endpunkt in einer Ressourcendatei speichern, die von Ihrer Anwendung verwendet wird.

Testen eines Failovers

In allen Fällen müssen Sie einen DB-Cluster mit zwei oder mehr DB-Instances haben.

Serverseitig können bestimmte API-Vorgänge einen Ausfall verursachen, anhand dessen Sie testen können, wie Ihre Anwendungen reagieren:

- FailoverDBCluster – Dieser Vorgang versucht, eine neue DB-Instance in Ihrem DB-Cluster an den Scrhreiber weiterzuleiten.

Das folgende Codebeispiel zeigt, wie Sie mit `failoverDBCluster` einen Ausfall verursachen können. Weitere Informationen zum Einrichten eines Amazon-RDS-Clients finden Sie unter Verwenden des AWS-SDK für Java.

```
public void causeFailover() {

    final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

    FailoverDBClusterRequest request = new FailoverDBClusterRequest();
    request.setDBClusterIdentifier("cluster-identifier");

    rdsClient.failoverDBCluster(request);
}
```

- RebootDBInstance – Failover ist nicht in diesem API-Vorgang garantiert. Es fährt jedoch die Datenbank auf dem Schreiber herunter. Sie können es verwenden, um zu testen, wie Ihre Anwendung auf das Abbrechen von Verbindungen reagiert. Der `ForceFailover`-Parameter gilt nicht für Aurora-Engines. Verwenden Sie stattdessen den `FailoverDBCluster`-API-Vorgang.
- ModifyDBCluster – Das Modifizieren des `Port`-Parameters wird einen Nutzungsausfall verursachen, wenn die Knoten im Cluster über einen neuen Port kommunizieren. Im Allgemeinen kann Ihre Anwendung zuerst auf diesen Fehler reagieren, indem sie sicherstellt, dass nur Ihre Anwendung die Portänderungen kontrolliert. Stellen Sie außerdem sicher, dass die Endpunkte, von denen es abhängig ist, ordnungsgemäß aktualisiert werden können. Sie können dies tun, indem Sie jemanden beauftragen, den Port manuell zu aktualisieren, wenn er Änderungen auf API-Ebene vornimmt. Oder Sie können dies tun, indem Sie die RDS-API in Ihrer Anwendung verwenden, um festzustellen, ob sich der Port geändert hat.
- ModifyDBInstance – Ändern des `DBInstanceClass`-Parameters verursacht einen Ausfall.
- DeleteDBInstance – Das Löschen des primären (Schreibers) wird eine Weiterleitung einer neuen DB-Instance an den Schreiber in Ihrem DB-Cluster verursachen.

Wenn Sie Linux verwenden, können Sie auf Anwendungs- oder Clientseite testen, wie die Anwendung auf plötzliche Paketverluste reagiert. Sie können dies basierend darauf tun, ob Port, Host oder ob TCP-Keepalive-Pakete gesendet oder empfangen werden, indem Sie den Befehl „iptables“ verwenden.

Beispiel für schnelles Java-Failover

Der folgende Beispiel-Code zeigt, wie eine Anwendung einen Aurora-PostgreSQL-Treibermanager einrichten kann.

Die Anwendung ruft die `getConnection`-Funktion auf, wenn eine Verbindung hergestellt werden soll. Ein Anruf bei `getConnection` kann möglicherweise keinen gültigen Host finden. Ein Beispiel ist, wenn kein Schreiber gefunden wird, aber der `targetServerType`-Parameter auf `primary` festgelegt ist. In diesem Fall sollte die aufrufende Anwendung einfach erneut versuchen, die Funktion aufzurufen.

Um zu vermeiden, dass das Wiederholungsverhalten auf die Anwendung geschoben wird, können Sie diesen Wiederholungsaufruf in einen Verbindungspooler verpacken. Bei den meisten Verbindungspoolern können Sie eine JDBC-Verbindungszeichenfolge angeben. Ihre Anwendung ewerbung kann sich also bei `getJdbcConnectionString` einwählen und das an den Verbindungspooler weitergeben. Dadurch können Sie ein schnelleres Failover mit Aurora PostgreSQL verwenden.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
   private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
   private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

   public FastFailoverDriverManager() {
       try {
            Class.forName("org.postgresql.Driver");
       } catch (ClassNotFoundException e) {
            e.printStackTrace();
       }
```

```
      /*
        * RO endpoint has a TTL of 1s, we should honor that here. Setting this
aggressively makes sure that when
        * the PG JDBC driver creates a new connection, it will resolve a new different
RO endpoint on subsequent attempts
        * (assuming there is > 1 read node in your cluster)
        */
       java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
      // If the lookup fails, default to something like small to retry
      java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
  }

  public Connection getConnection(String targetServerType) throws SQLException {
      return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
  }

  public Connection getConnection(String targetServerType, Duration queryTimeout)
throws SQLException {
       Connection conn =
DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

      /*
        * A good practice is to set socket and statement timeout to be the same thing
since both
        * the client AND server will stop the query at the same time, leaving no
running queries
        * on the backend
        */
       Statement st = conn.createStatement();
       st.execute("set statement_timeout to " + queryTimeout.getMillis());
       st.close();

      return conn;
  }

  private static String urlFormat = "jdbc:postgresql://%s"
          + "/postgres"
          + "?user=%s"
          + "&password=%s"
          + "&loginTimeout=%d"
          + "&connectTimeout=%d"
          + "&cancelSignalTimeout=%d"
          + "&socketTimeout=%d"
          + "&targetServerType=%s"
```

```
            + "&tcpKeepAlive=true"
            + "&ssl=true"
            + "&loadBalanceHosts=true";
    public String getJdbcConnectionString(String targetServerType, Duration
 queryTimeout) {
        return String.format(urlFormat,
                getFormattedEndpointList(getLocalEndpointList()),
                CredentialManager.getUsername(),
                CredentialManager.getPassword(),
                LOGIN_TIMEOUT.getStandardSeconds(),
                CONNECT_TIMEOUT.getStandardSeconds(),
                CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
                queryTimeout.getStandardSeconds(),
                targetServerType
        );
    }

    private List<String> getLocalEndpointList() {
        /*
          * As mentioned in the best practices doc, a good idea is to read a local
 resource file and parse the cluster endpoints.
          * For illustration purposes, the endpoint list is hardcoded here
          */
         List<String> newEndpointList = new ArrayList<>();
         newEndpointList.add("myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");
         newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");

        return newEndpointList;
    }

    private static String getFormattedEndpointList(List<String> endpoints) {
        return IntStream.range(0, endpoints.size())
                .mapToObj(i -> endpoints.get(i).toString())
                .collect(Collectors.joining(","));
    }
}
```

Schnelle Wiederherstellung nach Failover mit der Cluster-Cacheverwaltung für Aurora PostgreSQL

Verwenden Sie bei einem Failover zur schnellen Wiederherstellung der Writer-DB-Instance in den Aurora PostgreSQL-Clustern die Cluster-Cacheverwaltung für Amazon Aurora PostgreSQL. Die Cluster-Cacheverwaltung stellt bei einem Failover die Anwendungsleistung sicher.

Bei einem typischen Failover kommt es möglicherweise zu einem temporären, aber erheblichen Leitungsabfall. Dieser Abfall ist darauf zurückzuführen, dass der Puffercache beim Start der Failover-DB-Instance leer ist. Ein leerer Cache wird auch als kalter Cache bezeichnet. Ein kalter Cache beeinträchtigt die Leistung, da die DB-Instance vom langsameren Datenträger lesen muss, statt die im Puffercache gespeicherten Werte nutzen zu können.

Mit der Cluster-Cacheverwaltung legen Sie eine bestimmte Reader-DB-Instance als Failover-Ziel fest. Die Cluster-Cacheverwaltung stellt sicher, dass die Daten im Cache des designierten Readers mit den Cachedaten der Writer-DB-Instance synchronisiert werden. Der vorab gefüllte Cache des designierten Readers wird als warmer Cache bezeichnet. Wenn ein Failover auftritt, verwendet der designierte Reader Werte in seinem warmen Cache, sobald er auf die neue Writer-DB-Instance hochgestuft wird. Dank dieses Ansatzes wird die Wiederherstellungsleistung Ihrer Anwendung deutlich verbessert.

Die Cluster-Cacheverwaltung erfordert, dass die zugeordnete Reader-Instance den gleichen Instance-Klassentyp und dieselbe Größe (zum Beispiel `db.r5.2xlarge` oder `db.r5.xlarge`) hat wie der Writer. Denken Sie daran, wenn Sie Ihre Aurora PostgreSQL-DB-Cluster erstellen, damit Ihr Cluster während eines Failovers wiederhergestellt werden kann. Eine Auflistung der Instance-Klassentypen und -größen finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

> **Note**
>
> Die Verwaltung des Cluster-Cache wird für sekundäre DB-Cluster von Aurora PostgreSQL, die Teil der globalen Aurora-Datenbanken sind, nicht unterstützt. Für primäre Cluster, in denen diese Funktion unterstützt wird, wird empfohlen, dass kein Workload auf dem angegebenen Stufe-0-Leser ausgeführt wird.

Inhalt

- Konfigurieren der Cluster-Cache-Verwaltung

- Aktivieren der Cluster-Cache-Verwaltung
- Festlegen der Prioritätsstufe für das Hochstufen für die Writer-DB-Instance
- Festlegen der Prioritätsstufe für das Hochstufen für eine Reader-DB-Instance
- Überwachung des Puffercaches
- Fehlerbehebung bei der CCM-Konfiguration

Konfigurieren der Cluster-Cache-Verwaltung

Führen Sie die folgenden Prozesse der Reihe nach aus, um die Cluster-Cache-Verwaltung zu konfigurieren.

Themen

- Aktivieren der Cluster-Cache-Verwaltung
- Festlegen der Prioritätsstufe für das Hochstufen für die Writer-DB-Instance
- Festlegen der Prioritätsstufe für das Hochstufen für eine Reader-DB-Instance

> **Note**
>
> Warten Sie nach Abschluss dieser Schritte mindestens 1 Minute ab, damit die Cluster-Cache-Verwaltung voll einsatzbereit ist.

Aktivieren der Cluster-Cache-Verwaltung

Um die Cluster-Cache-Verwaltung zu aktivieren, führen Sie die im Folgenden beschriebenen Schritte aus,.

Konsole

So aktivieren Sie die Cluster-Cacheverwaltung:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie die Parametergruppe für Ihren Aurora PostgreSQL-DB-Cluster in der Liste aus.

Der DB-Cluster muss eine andere als die Standardparametergruppe verwenden, da die Werte in dieser Gruppe nicht geändert werden können.

4. Wählen Sie für Parameter group actions (Parametergruppenaktionen) die Option Bearbeiten.
5. Legen Sie den Wert des Cluster-Parameters `apg_ccm_enabled` auf 1 fest.
6. Wählen Sie Änderungen speichern aus.

AWS CLI

Verwenden Sie zum Aktivieren der Cluster-Cacheverwaltung für einen Aurora-PostgreSQL-DB-Cluster den AWS CLI `modify-db-cluster-parameter-group`-Befehl mit den folgenden erforderlichen Parametern:

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my-db-cluster-parameter-group \
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my-db-cluster-parameter-group ^
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Festlegen der Prioritätsstufe für das Hochstufen für die Writer-DB-Instance

Für die Cluster-Cache-Verwaltung muss die Prioritätsstufe für das Hochstufen für die Writer-DB-Instance des Aurora PostgreSQL-DB-Clusters tier-0 sein. Die Prioritätsstufe für das Hochstufen gibt die Reihenfolge an, in der ein Aurora-Reader nach einem Ausfall zur Writer-DB-Instance hochgestuft wird. Gültige Werte sind 0 – 15, wobei 0 die erste und 15 die letzte Prioritätsstufe darstellt. Weitere Informationen zur Hochstufungspriorität finden Sie unter Fehlertoleranz für einen Aurora-DB-Cluster.

Konsole

So legen Sie die Prioritätsstufe für das Hochstufen auf die Writer-DB-Instance auf „tier-0“ fest

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie die Writer-DB-Instance des Aurora PostgreSQL-DB-Clusters aus.
4. Wählen Sie Ändern aus. Die Seite Modify DB Instance (DB-Instance ändern) wird angezeigt.
5. Wählen Sie im Bereich Zusätzliche Konfiguration für Failover priority (Failover-Priorität) die Option tier-0 aus.
6. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
7. Wählen Sie Apply immediately (Sofort anwenden) aus, um die Änderungen nach dem Speichern sofort zu übernehmen.
8. Klicken Sie auf Modify DB Instance (DB-Instance ändern), um Ihre Änderungen zu speichern.

AWS CLI

Rufen Sie zum Festlegen der Prioritätsstufe für das Hochstufen auf die Writer-DB-Instance über die AWS CLI den Befehl modify-db-instance mit den folgenden erforderlichen Parametern auf:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier writer-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Für Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier writer-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

Festlegen der Prioritätsstufe für das Hochstufen für eine Reader-DB-Instance

Für die Verwaltung des Cluster-Cache müssen Sie nur eine Leser-DB-Instance festlegen. Wählen Sie dazu einen Reader aus dem Aurora PostgreSQL-Cluster aus, der dieselbe Instance-Klasse und -Größe hat wie die Writer-DB-Instance. Wenn der Writer beispielsweise `db.r5.xlarge` verwendet, dann wählen Sie einen Reader aus, der denselben Instance-Klassentyp und dieselbe Größe verwendet. Legen Sie dann die Prioritätsstufe für das Hochstufen auf 0 fest.

Die Prioritätsstufe für das Hochstufen gibt die Reihenfolge an, in der ein Aurora-Reader nach einem Ausfall zur Writer-DB-Instance hochgestuft wird. Gültige Werte sind 0 – 15, wobei 0 die erste und 15 die letzte Prioritätsstufe darstellt.

Konsole

So legen Sie die Prioritätsstufe für das Hochstufen auf die Reader-DB-Instance auf „tier-0“ fest

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie eine Reader-DB-Instance aus dem Aurora PostgreSQL-DB-Cluster aus, der der Instance-Klasse der Writer-DB-Instance entspricht.
4. Wählen Sie Ändern aus. Die Seite Modify DB Instance (DB-Instance ändern) wird angezeigt.
5. Wählen Sie im Bereich Zusätzliche Konfiguration für Failover priority (Failover-Priorität) die Option tier-0 aus.
6. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
7. Wählen Sie Apply immediately (Sofort anwenden) aus, um die Änderungen nach dem Speichern sofort zu übernehmen.
8. Klicken Sie auf Modify DB Instance (DB-Instance ändern), um Ihre Änderungen zu speichern.

AWS CLI

Rufen Sie zum Festlegen der Prioritätsstufe für das Hochstufen auf die Reader-DB-Instance über die AWS CLI den Befehl modify-db-instance mit den folgenden erforderlichen Parametern auf:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier reader-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Für Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier reader-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

Überwachung des Puffercaches

Nach dem Einrichten der Cluster-Cache-Verwaltung können Sie den Synchronisierungsstatus zwischen dem Puffercache der Writer-DB-Instance und dem Warm-Puffercache des jeweiligen Readers überwachen. Um den Inhalt des Puffercache sowohl auf der Writer-DB-Instance als auch auf der angegebenen Reader-DB-Instance zu untersuchen, verwenden Sie das PostgreSQL-Modul `pg_buffercache`. Weitere Informationen finden Sie in der PostgreSQL `pg_buffercache`-Dokumentation.

Verwendung der **`aurora_ccm_status`**-Funktion

Die Cluster-Cacheverwaltung stellt außerdem die Funktion `aurora_ccm_status` zur Verfügung. Verwenden Sie die Funktion `aurora_ccm_status` auf der Writer-DB-Instance, um die folgenden Informationen zum Verlauf des Cache-Warmings auf dem designierten Reader abzurufen.

- `buffers_sent_last_minute` – Die Anzahl der Puffer, die in der letzten Minute an den designierten Reader gesendet wurden.
- `buffers_found_last_minute` – Die Anzahl der Puffer, auf die in der letzten Minute häufig zugegriffen wurde.
- `buffers_sent_last_scan` – Die Anzahl der Puffer, die während des letzten vollständigen Scanvorgangs des Puffercaches an den designierten Reader gesendet wurden.
- `buffers_found_last_scan` – Die Anzahl der Puffer, die häufig aufgerufen wurden und die während des letzten vollständigen Scanvorgangs des Puffercaches gesendet werden mussten. Puffer, die bereits im Cache des designierten Readers abgelegt wurden, werden nicht gesendet.
- `buffers_sent_current_scan` – Die Anzahl der Puffer, die während des aktuellen Scanvorgangs bisher gesendet wurden.
- `buffers_found_current_scan` – Die Anzahl der Puffer, die während des aktuellen Scanvorgangs häufig aufgerufen wurden.
- `current_scan_progress` – Die Anzahl der Puffer, die während des aktuellen Scanvorgangs bisher besucht wurden.

Das folgende Beispiel veranschaulicht, wie ein Teil der Ausgabe mithilfe der Funktion `aurora_ccm_status` in eine aktive Rate und einen aktiven Prozentsatz konvertiert wird.

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
   100*(1.0-buffers_sent_last_scan::float/buffers_found_last_scan) AS warm_percent
   FROM aurora_ccm_status();
```

Fehlerbehebung bei der CCM-Konfiguration

Wenn Sie den Cluster-Parameter `apg_ccm_enabled` aktivieren, wird die Verwaltung des Cluster-Cache automatisch auf Instance-Ebene auf der Schreiber-DB-Instance und einer Leser-DB-Instance auf dem DB-Cluster von Aurora PostgreSQL aktiviert. Die Schreiber- und Leser-Instance sollten denselben Instance-Klassentyp und dieselbe Größe verwenden. Ihre Prioritätsstufe für das Hochstufen ist auf `0` gesetzt. Für andere Leser-Instances im DB-Cluster sollte eine Nicht-Null-Hochstufungsstufe gelten, und die Verwaltung des Cluster-Cache ist für diese Instances deaktiviert.

Die folgenden Gründe können zu Problemen bei der Konfiguration und der Deaktivierung der Verwaltung des Cluster-Cache führen:

- Wenn keine einzige Leser-DB-Instance auf Hochstufungsstufe 0 gesetzt ist.

- Wenn die Schreiber-DB-Instance nicht auf Hochstufungsstufe 0 gesetzt ist.
- Wenn mehr als eine Leser-DB-Instance auf Hochstufungsstufe 0 gesetzt ist.
- Wenn die Schreiber- und eine Leser-DB-Instance mit Hochstufungsstufe 0 nicht dieselbe Instance-Größe haben.

Verwalten von Aurora PostgreSQL Verbindungsabwanderung mit Pooling

Wenn Clientanwendungen so oft eine Verbindung herstellen und trennen, dass sich die Reaktionszeit des Aurora PostgreSQL DB-Clusters verlangsamt, erfährt der Cluster Verbindungsprobleme. Jede neue Verbindung zum Aurora PostgreSQL DB-Cluster-Endpunkt verbraucht Ressourcen und reduziert so die Ressourcen, die zur Verarbeitung der tatsächlichen Workload verwendet werden können. Verbindungsprobleme sollten Sie anhand einiger der unten beschriebenen bewährten Methoden lösen.

Für den Anfang können Sie die Antwortzeiten auf Aurora PostgreSQL DB-Clustern mit hohen Verbindungsproblemvorkommnissen verbessern. Sie können beispielsweise einen Verbindungspooler wie RDS-Proxy verwenden. Ein Verbindungspooler stellt einen Cache mit gebrauchsfertigen Verbindungen für Clients bereit. Fast alle Versionen von Aurora PostgreSQL unterstützen RDS-Proxy. Weitere Informationen finden Sie unter Amazon-RDS-Proxy mit Aurora PostgreSQL.

Wenn Ihre spezielle Version von Aurora PostgreSQL RDS-Proxy nicht unterstützt, können Sie einen anderen PostgreSQL-kompatiblen Verbindungspooler wie PGBouncer verwenden. Weitere Informationen hierzu finden Sie auf der PG Bouncer-Website.

Um zu sehen, ob Ihr Aurora-PostgreSQL-DB-Cluster vom Verbindungspooling profitieren kann, können Sie die `postgresql.log`-Datei für Verbindungen und Verbindungstrennungen prüfen. Sie können Performance Insights auch verwenden, um herauszufinden, wie viel Verbindungsprobleme in Ihrem Aurora PostgreSQL DB-Cluster auftreten. Im Folgenden finden Sie Informationen zu beiden Themen.

Verbindungen und Verbindungstrennungen protokollieren

Die PostgreSQL `log_connections`- und `log_disconnections`-Parameter können Verbindungen und Verbindungsabbrüche zur Writer-Instance des Aurora-PostgreSQL-DB-Clusters. Diese Parameter sind standardmäßig deaktiviert. Um diese Parameter zu aktivieren, verwenden Sie eine benutzerdefinierte Parametergruppe, und aktivieren Sie sie, indem Sie den Wert in 1 ändern. Weitere Informationen zu benutzerdefinierten DB-Parametergruppen finden Sie unter DB-Cluster-

Parametergruppen für Amazon-Aurora-DB-Cluster. Um die Einstellungen zu überprüfen, stellen Sie mithilfe von psql eine Verbindung zu Ihrem DB-Cluster-Endpunkt für Aurora PostgreSQL her und fragen Sie wie folgt ab.

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
 setting
---------
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
---------
on
(1 row)
```

Wenn beide Parameter aktiviert sind, erfasst das Protokoll alle neuen Verbindungen und Verbindungsabbrüche. Sie sehen den Benutzer und die Datenbank für jede neue autorisierte Verbindung. Beim Verbindungstrennungen wird auch die Sitzungsdauer protokolliert, wie im folgenden Beispiel gezeigt.

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek
 database=labdb application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
 user=labtek database=labdb host=[local]
```

Um Ihre Anwendung auf Verbindungsprobleme zu überprüfen, aktivieren Sie diese Parameter, falls sie noch nicht aktiviert sind. Sammeln Sie dann Daten im PostgreSQL-Protokoll zur Analyse, indem Sie Ihre Anwendung mit einer realistischen Workload und einem realistischen Zeitraum ausführen. Sie können die Protokolldatei in der RDS-Konsole anzeigen. Wählen Sie die Writer-Instance Ihres Aurora-PostgreSQL-DB-Clusters aus und klicken Sie dann auf den Tab Logs & Ereignisse. Weitere Informationen finden Sie unter Anzeigen und Auflisten von Datenbank-Protokolldateien.

Sie können auch die Protokolldatei von der Konsole herunterladen und die folgende Befehlssequenz verwenden. Diese Sequenz ermittelt die Gesamtzahl der autorisierten und unterbrochenen Verbindungen pro Minute.

```
grep "connection authorized\|disconnection: session time:"
 postgresql.log.2022-03-21-16|\
```

```
awk {'print $1,$2}' |\
sort |\
uniq -c |\
sort -n -k1
```

In der Beispielausgabe sehen Sie einen Anstieg der autorisierten Verbindungen, gefolgt von Verbindungsabbrüchen ab 16:12:10.

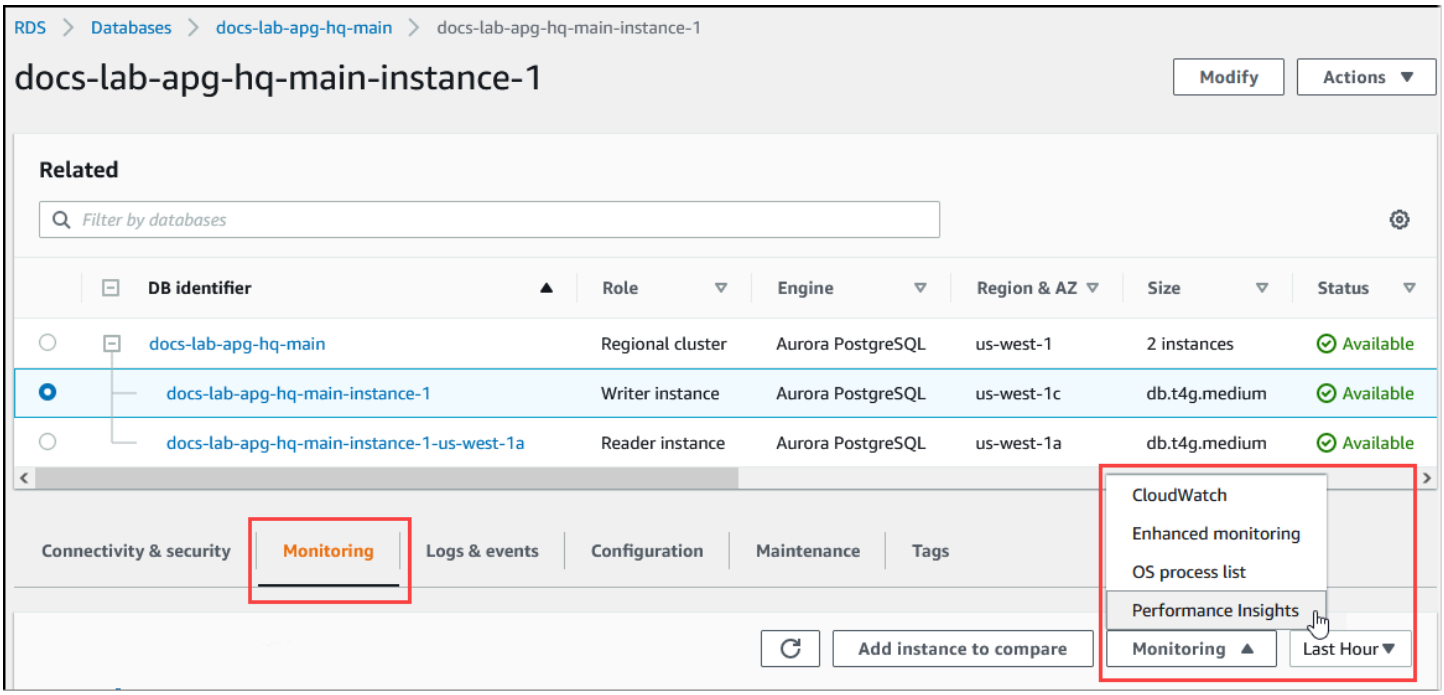
```
.....
,......
.........
5 2022-03-21 16:11:55 connection authorized:
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
,......
.........
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
1 2022-03-21 16:16:00 connection authorized:
```

Anhand dieser Informationen können Sie entscheiden, ob Ihr Workload von einem Verbindungspooler profitieren kann. Für detailliertere Analysen können Sie Performance Insights verwenden.

Erkennen eines Verbindungsproblems mit Performance Insights

Mit Performance Insights können Sie den Umfang der Verbindungsprobleme auf Ihrem Aurora-PostgreSQL-kompatiblen Edition-DB-Cluster beurteilen. Wenn Sie einen Aurora-PostgreSQL-DB-Cluster erstellen, ist die Einstellung für Performance Insights standardmäßig aktiviert. Wenn Sie diese Auswahl beim Erstellen des DB-Clusters deaktiviert haben, ändern Sie Ihren Cluster, um die Funktion zu aktivieren. Weitere Informationen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

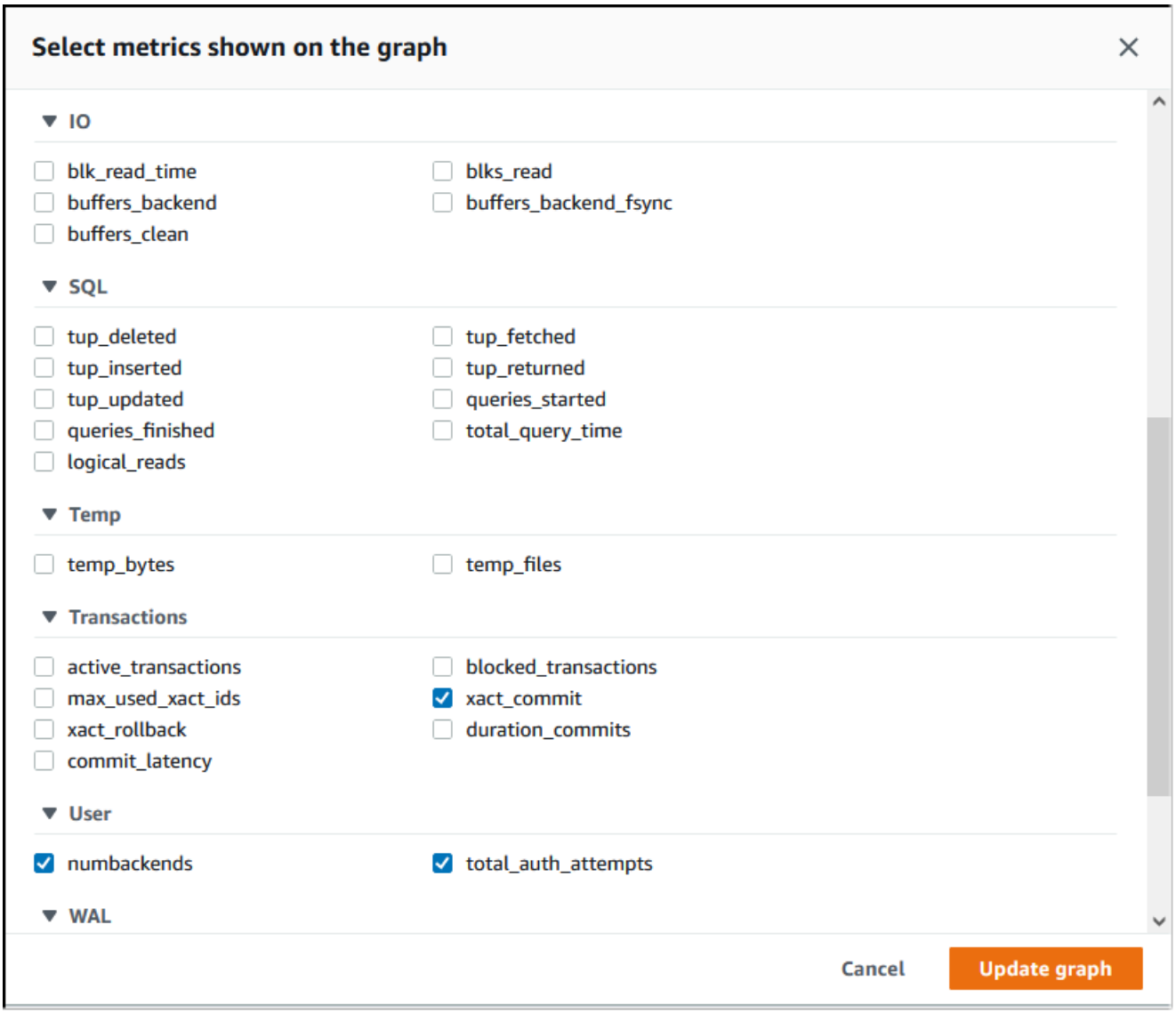
Wenn Performance Insights auf Ihrem Aurora PostgreSQL DB-Cluster ausgeführt wird, können Sie die Metriken auswählen, die Sie überwachen möchten. Sie können über den Navigationsbereich der -Konsole auf Performance Insights zugreifen. Sie können auch auf Performance Insights vom Tab Überwachung der Writer-Instance für Ihren Aurora PostgreSQL DB-Cluster, wie in der folgenden Abbildung gezeigt, zugreifen.



Wählen Sie in der Performance-Insights-Konsole Metriken verwalten. Wählen Sie die folgenden Metriken aus, um die Verbindungs- und Trennungsaktivitäten Ihres Aurora PostgreSQL DB-Clusters zu analysieren. Dies sind alles Metriken von PostgreSQL.

- `xact_commit` – Die Anzahl von committeter Transaktionen.

- `total_auth_attempts` – Die Anzahl von versuchten authentifizierten Benutzerverbindungen pro Minute.
- `numbackends` – Die Anzahl der Backends, die derzeit mit der Datenbank verbunden sind.

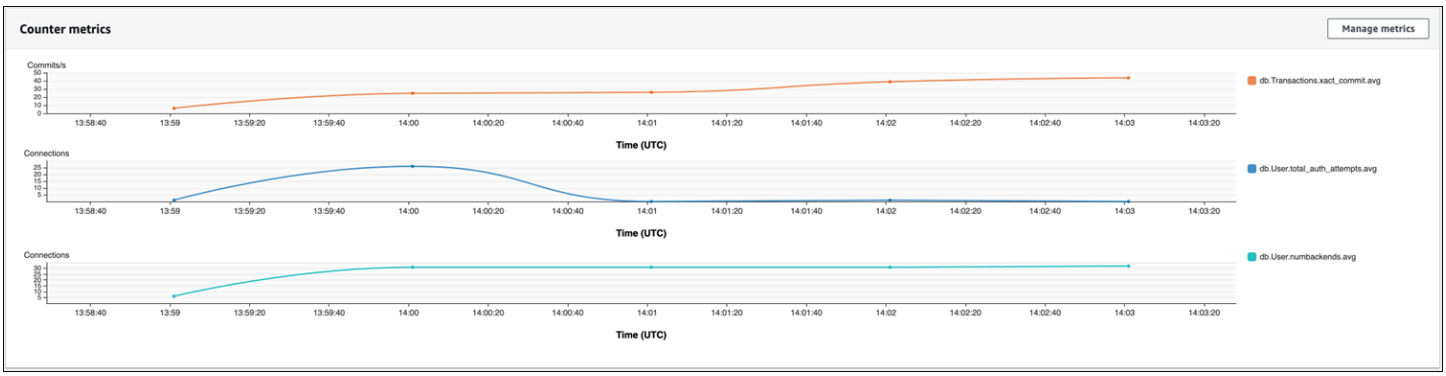


Wählen Sie zum Speichern der Einstellungen und Anzeigen der Verbindungsaktivität Grafik aktualisieren.

In der folgenden Abbildung sehen Sie die Auswirkungen der Ausführung von pgbench mit 100 Benutzern. Die Linie, die Verbindungen zeigt, ist konstant nach oben geneigt. Weitere Informationen zu pgbench finden Sie unter pgbench in der PostgreSQL-Dokumentation.

Das Bild zeigt, dass das Ausführen einer Workload mit nur 100 Benutzern ohne Verbindungspooler zu einem signifikanten Anstieg der Anzahl von `total_auth_attempts` während der gesamten Dauer der Workloadverarbeitung führt. Beachten Sie, dass es am besten ist, `total_auth_attempts` möglichst bei Null zu halten.

Beim RDS-Proxy-Verbindungspooling nehmen die Verbindungsversuche zu Beginn der Workload zu. Nach dem Einrichten des Verbindungspools sinkt der Durchschnitt. Die für Transaktionen und Backend-Nutzung verwendeten Ressourcen bleiben während der gesamten Workloadverarbeitung konsistent.



Weitere Informationen über die Verwendung von Performance Insights mit Ihrem Aurora-PostgreSQL-DB-Cluster finden Sie unter Überwachung mit Performance Insights auf . Informationen zum Analysieren der Metriken finden Sie unter Analyse der Metriken mit dem Performance-Insights-Dashboard.

Vorführen der Vorteile von Verbindungspooling

Wie bereits erwähnt, können Sie RDS-Proxy verwenden, um die Leistung zu verbessern, wenn Sie feststellen, dass Ihr Aurora PostgreSQL DB-Cluster ein Verbindungsproblem aufweist. Im Folgenden finden Sie ein Beispiel, das die Unterschiede bei der Verarbeitung einer Workload zeigt, wenn

Verbindungen gepoolt werden und wenn sie nicht gepoolt sind. Das Beispiel verwendet pgbench, um einen Transaktionsworkload zu modellieren.

Wie psql ist pgbench eine PostgreSQL-Clientanwendung, die Sie von Ihrem lokalen Client-Computer aus installieren und ausführen können. Sie können es auch über die Amazon-EC2-Instance installieren und ausführen, die Sie für die Verwaltung Ihres Aurora-PostgreSQL-DB-Clusters verwenden. Weitere Informationen finden Sie unter pgbench in der PostgreSQL-Dokumentation.

Um dieses Beispiel Schritt für Schritt durchzugehen, erstellen Sie zunächst die pgbench-Umgebung in Ihrer Datenbank. Der folgende Befehl ist die grundlegende Vorlage für die Initialisierung der pgbench-Tabellen in der angegebenen Datenbank. In diesem Beispiel wird das standardmäßige Hauptbenutzerkonto, `postgres`, für die Anmeldung verwendet. Ändern Sie es nach Bedarf für Ihren Aurora-PostgreSQL-DB-Cluster. Sie erstellen die pgbench-Umgebung in einer Datenbank auf der Writer-Instance Ihres Clusters.

> **Note**
>
> Der Initialisierungsprozess von pgbench löscht Tabellen mit den Namen `pgbench_accounts`, `pgbench_branches`, `pgbench_history` und `pgbench_tellers` erstellt sie neu. Stellen Sie sicher, dass die Datenbank, die Sie für *`dbname`* wählen, diese Namen nicht verwendet, wenn Sie pgbench initialisieren.

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com
 -p 5432 -d -i -s 50 dbname
```

Geben Sie folgende Parameter für pgbench an.

-d

Gibt einen Debugging-Bericht aus, während pgbench ausgeführt wird.

-h

Gibt den Endpunkt der Writer-DB-Instance des Aurora-PostgreSQL-DB-Clusters an.

-i

Initialisiert die pgbench-Umgebung in der Datenbank für die Benchmark-Tests.

-p

Identifiziert den Port, der für Datenbankverbindungen verwendet wird. Die Standardeinstellung für Aurora PostgreSQL ist typischerweise 5432 oder 5433.

-s

Gibt den Skalierungsfaktor an, der zum Füllen der Tabellen mit Zeilen verwendet werden soll. Der Standard-Skalierungsfaktor ist 1, wodurch 1 Zeile in der `pgbench_branches`-Tabelle, 10 Zeilen in der `pgbench_tellers`-Tabelle und 100000 Zeilen in der `pgbench_accounts`-Tabelle erzeugt werden.

-U

Gibt das Benutzerkonto für die Writer-Instance des Aurora-PostgreSQL-DB-Clusters an.

Nachdem die pgbench-Umgebung eingerichtet wurde, können Sie Benchmarking-Tests mit und ohne Verbindungspooling ausführen. Der Standardtest besteht aus einer Reihe von fünf SELECT-, UPDATE- und INSERT-Befehlen pro Transaktion, die für die angegebene Zeit wiederholt ausgeführt werden. Sie können Skalierungsfaktor, Anzahl der Clients und andere Details angeben, um Ihre eigenen Anwendungsfälle zu modellieren.

Der folgende Befehl führt beispielsweise den Benchmark 60 Sekunden lang aus (Option -T, für Zeit) mit 20 gleichzeitigen Verbindungen (Option -c). Mit der Option -C wird der Test jedes Mal mit einer neuen Verbindung ausgeführt, anstatt einmal pro Clientsitzung. Diese Einstellung gibt Ihnen einen Hinweis auf den Verbindungsaufwand.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 -C labdb
Password:**********
pgbench (14.3, server 13.3)
  starting vacuum...end.
  transaction type: <builtin: TPC-B (sort of)>
    scaling factor: 50
    query mode: simple
    number of clients: 20
    number of threads: 1
    duration: 60 s
    number of transactions actually processed: 495
    latency average = 2430.798 ms
    average connection time = 120.330 ms
    tps = 8.227750 (including reconnection times)
```

Das Ausführen von pgbench auf der Writer-Instance eines Aurora PostgreSQL DB-Clusters ohne Wiederverwendung von Verbindungen zeigt, dass nur etwa 8 Transaktionen pro Sekunde verarbeitet werden. Dies ergibt insgesamt 495 Transaktionen während des 1-Minutentests.

Wenn Sie Verbindungen wiederverwenden, ist die Antwort des Aurora PostgreSQL DB-Clusters für die Anzahl der Benutzer fast 20-mal schneller. Bei der Wiederverwendung werden insgesamt 9.042 Transaktionen verarbeitet, verglichen mit 495 in derselben Zeit und für dieselbe Anzahl von Benutzerverbindungen. Der Unterschied besteht darin, dass im Folgenden jede Verbindung wiederverwendet wird.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 labdb
Password:*********
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 9042
      latency average = 127.880 ms
      initial connection time = 2311.188 ms
      tps = 156.396765 (without initial connection time)
```

Dieses Beispiel zeigt, dass das Pooling von Verbindungen die Reaktionszeiten erheblich verbessern kann. Informationen über die Einrichtung von RDS-Proxy für Ihren Aurora-PostgreSQL-DB-Cluster finden Sie unter Amazon-RDS-Proxy für Aurora.

Behandeln von inaktiven Verbindungen in PostgreSQL

Inaktive Verbindungen treten auf, wenn eine Datenbanksitzung auf dem Server aktiv bleibt, obwohl die Client-Anwendung abgebrochen oder auf ungewöhnliche Weise beendet wurde. Eine solche Situation entsteht in der Regel, wenn Client-Prozesse abstürzen oder unerwartet beendet werden, ohne dass ihre Datenbankverbindungen ordnungsgemäß geschlossen oder laufende Anfragen abgebrochen werden.

PostgreSQL identifiziert und bereinigt inaktive Verbindungen, wenn Serverprozesse inaktiv sind oder versuchen, Daten an Clients zu senden. Die Erkennung ist jedoch schwierig bei Sitzungen, die im

Leerlauf sind, auf Eingaben vom Client warten oder Abfragen aktiv ausführen. Um diese Szenarien zu handhaben, stellt PostgreSQL die Parameter `tcp_keepalives_*`, `tcp_user_timeout` und `client_connection_check_interval` bereit.

Themen

- Grundlagen zu TCP-Keepalive
- Wichtige TCP-Keepalive-Parameter in Aurora PostgreSQL
- Anwendungsfälle für TCP-Keepalive-Einstellungen
- Bewährte Methoden

Grundlagen zu TCP-Keepalive

TCP-Keepalive ist ein Mechanismus auf Protokollebene, der dabei hilft, die Verbindungsintegrität aufrechtzuerhalten und zu überprüfen. Jede TCP-Verbindung verwaltet Einstellungen auf Kernelebene, die das Keepalive-Verhalten steuern. Wenn der Keepalive-Timer abläuft, führt das System folgenden Aktionen durch:

- Es sendet ein Testpaket ohne Daten und mit gesetztem ACK-Flag.
- Es erwartet eine Antwort vom Remote-Endpunkt gemäß den TCP/IP-Spezifikationen.
- Es verwaltet den Verbindungsstatus auf der Grundlage der Antwort oder des Fehlens einer Antwort.

Wichtige TCP-Keepalive-Parameter in Aurora PostgreSQL

Parameter	Beschreibung	Standardwerte
`tcp_keepalives_idle`	Specifies number of seconds of inactivity before sending keepalive message.	300
`tcp_keepalives_interval`	Specifies number of seconds between retransmissions of unacknowledged keepalive messages.	30

Parameter	Beschreibung	Standardwerte
`tcp_keepalives_count`	Maximum lost keepalive messages before declaring connection dead	2
`tcp_user_timeout`	Specifies how long (in Milliseconds) unacknowledged data can remain before forcibly closing the connection.	0
`client_connection_check_interval`	Sets the interval (in Milliseconds) for checking client connection status during long-running queries. This ensures quicker detection of closed connections.	0

Anwendungsfälle für TCP-Keepalive-Einstellungen

Aufrechterhalten von Sitzungen im Leerlauf

So verhindern Sie, dass inaktive Verbindungen aufgrund von Inaktivität durch Firewalls oder Router beendet werden:

- Konfigurieren Sie `tcp_keepalives_idle`, um Keepalive-Pakete in regelmäßigen Abständen zu senden.

Erkennen von inaktiven Verbindungen

Gehen Sie wie folgt vor, um inaktive Verbindungen umgehend zu erkennen:

- Passen Sie `tcp_keepalives_idle`, `tcp_keepalives_interval` und `tcp_keepalives_count` an. Mit den Standardeinstellungen von Aurora PostgreSQL dauert es beispielsweise etwa 1 Minute (2 Proben × 30 Sekunden), um eine interaktive Verbindung zu erkennen. Eine Senkung dieser Werte kann die Erkennung beschleunigen.
- Verwenden Sie `tcp_user_timeout`, um die maximale Wartezeit für eine Bestätigung festzulegen.

TCP-Keepalive-Einstellungen helfen dem Kernel, inaktive Verbindungen zu erkennen, aber PostgreSQL reagiert möglicherweise erst, wenn der Socket verwendet wird. Wenn in einer Sitzung eine lange Abfrage ausgeführt wird, werden inaktive Verbindungen möglicherweise erst nach Abschluss der Abfrage erkannt. In PostgreSQL 14 und höheren Versionen kann `client_connection_check_interval` die Erkennung inaktiver Verbindungen beschleunigen, indem der Socket während der Abfrageausführung regelmäßig abgefragt wird.

Bewährte Methoden

- Angemessene Keepalive-Intervalle festlegen: Optimieren Sie `tcp_user_timeout`, `tcp_keepalives_idle`, `tcp_keepalives_count` und `tcp_keepalives_interval`, um für ein ausgewogenes Verhältnis zwischen Erkennungsgeschwindigkeit und Ressourcennutzung zu sorgen.
- Für Ihre Umgebung optimieren. Richten Sie Einstellungen am Netzwerkverhalten, an Firewall-Richtlinien und an Sitzungsanforderungen aus.
- PostgreSQL-Funktionen nutzen: Verwenden Sie `client_connection_check_interval` in PostgreSQL 14 und höheren Versionen für effiziente Verbindungsprüfungen.

Ändern von Speicherparametern für Aurora PostgreSQL

In Amazon Aurora PostgreSQL können Sie mehrere Parameter verwenden, die die Speichermenge steuern, die für verschiedene Verarbeitungsaufgaben verwendet wird. Wenn eine Aufgabe mehr Speicher beansprucht als die für einen bestimmten Parameter festgelegte Menge, verwendet Aurora PostgreSQL andere Ressourcen für die Verarbeitung, z. B. durch Schreiben auf die Festplatte. Dies kann dazu führen, dass Ihr Aurora PostgreSQL DB-Cluster langsamer wird oder möglicherweise angehalten wird, mit einem Fehler bei unzureichendem Arbeitsspeicher.

Die Standardeinstellung für jeden Speicherparameter kann normalerweise die beabsichtigten Verarbeitungsaufgaben verarbeiten. Sie können jedoch auch Ihre speicherbezogenen Parameter von Aurora PostgreSQL DB-Cluster optimieren. Sie führen diese Optimierung durch, um sicherzustellen, dass genügend Speicher für die Verarbeitung Ihrer spezifischen Workload zugewiesen ist.

Nachstehend finden Sie Informationen über Parameter, die die Speicherverwaltung steuern. Sie können auch lernen, wie Sie die Speicherauslastung bewerten.

Überprüfen und Einstellen von Parameterwerten

Zu den Parametern, die Sie festlegen können, um den Speicher zu verwalten und die Speicherauslastung Ihres Aurora PostgreSQL DB-Clusters zu bewerten, gehören die folgenden:

- `work_mem` – Gibt die Speichermenge an, die der Aurora PostgreSQL DB-Cluster für interne Sortiervorgänge und Hash-Tabellen verwendet, bevor er in temporäre Plattendateien schreibt.
- `log_temp_files` – Protokolliert die Erstellung temporärer Dateien, Dateinamen und -größen. Wenn dieser Parameter aktiviert ist, wird für jede temporäre Datei, die erstellt wird, ein Protokolleintrag gespeichert. Aktivieren Sie diese Option, um zu sehen, wie oft Ihr Aurora PostgreSQL DB-Cluster auf die Festplatte schreiben muss. Schalten Sie es wieder aus, nachdem Sie Informationen über die temporäre Dateigenerierung Ihres Aurora PostgreSQL DB-Clusters gesammelt haben, um eine übermäßige Protokollierung zu vermeiden.
- `logical_decoding_work_mem` – Gibt die Speichermenge (in Kilobyte) an, die von jedem internen Neuordnungspuffer vor dem Schreiben auf die Festplatte genutzt werden soll. Dieser Speicher wird für die logische Dekodierung verwendet, ein Prozess zur Erstellung eines Replikats. Dies erfolgt durch Konvertieren von Daten aus der Write-Ahead-Log-Datei (WAL) in die logische Streaming-Ausgabe, die vom Ziel benötigt wird.

  Der Wert dieses Parameters erstellt einen einzelnen Puffer mit der für jede Replikationsverbindung angegebenen Größe. Standardmäßig sind es 65 536 KB. Nachdem dieser Puffer gefüllt ist, wird der Überschuss als Datei auf die Festplatte geschrieben. Um Plattenaktivität zu minimieren, können Sie den Wert dieses Parameters auf einen viel höheren Wert setzen als den von `work_mem`.

Dies sind alles dynamische Parameter, sodass Sie sie für die aktuelle Sitzung ändern können. Stellen Sie dazu eine Verbindung mit dem Aurora PostgreSQL DB-Cluster mit „psql“ und unter Verwendung der `SET`-Anweisung, wie unten angezeigt.

```
SET parameter_name TO parameter_value;
```

Sitzungseinstellungen sind nur für die Dauer der Sitzung gültig. Wenn die Sitzung endet, kehrt der Parameter auf seine Einstellung in der DB-Cluster-Parametergruppe zurück. . Bevor Sie Parameter ändern, überprüfen Sie zunächst die aktuellen Werte, indem Sie die `pg_settings`-Tabelle wie folgt abfragen.

```
SELECT unit, setting, max_val
```

```
   FROM pg_settings WHERE name='parameter_name';
```

Um zum Beispiel den Wert des `work_mem`-Parameters zu finden, stellen Sie eine Verbindung zur Writer-Instance des Aurora PostgreSQL DB-Clusters her und führen Sie die folgende Abfrage aus.

```
SELECT unit, setting, max_val, pg_size_pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit  | setting  | max_val   | pg_size_pretty
------+----------+-----------+----------------
 kB   | 1024     | 2147483647| 2048 MB
(1 row)
```

Um Parametereinstellungen so zu ändern, dass sie bestehen bleiben, müssen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe verwenden. . Nach dem Trainieren Ihrer Aurora PostgreSQL DB-Cluster mit unterschiedlichen Werten für diese Parameter unter Verwendung der `SET`-Anweisung können Sie eine benutzerdefinierte Parametergruppe erstellen und auf Ihre Parametergruppe Aurora PostgreSQL DB-Cluster anwenden. Weitere Informationen finden Sie unter Parametergruppen für Amazon Aurora.

Den Arbeitsspeicherparameter verstehen

Der Arbeitsspeicherparameter (`work_mem`) gibt die maximale Speichermenge an, die Aurora PostgreSQL zur Verarbeitung komplexer Abfragen verwenden kann. Zu komplexen Abfragen gehören solche, die Sortier- oder Gruppierungsvorgänge beinhalten, also Abfragen, die die folgenden Klauseln verwenden:

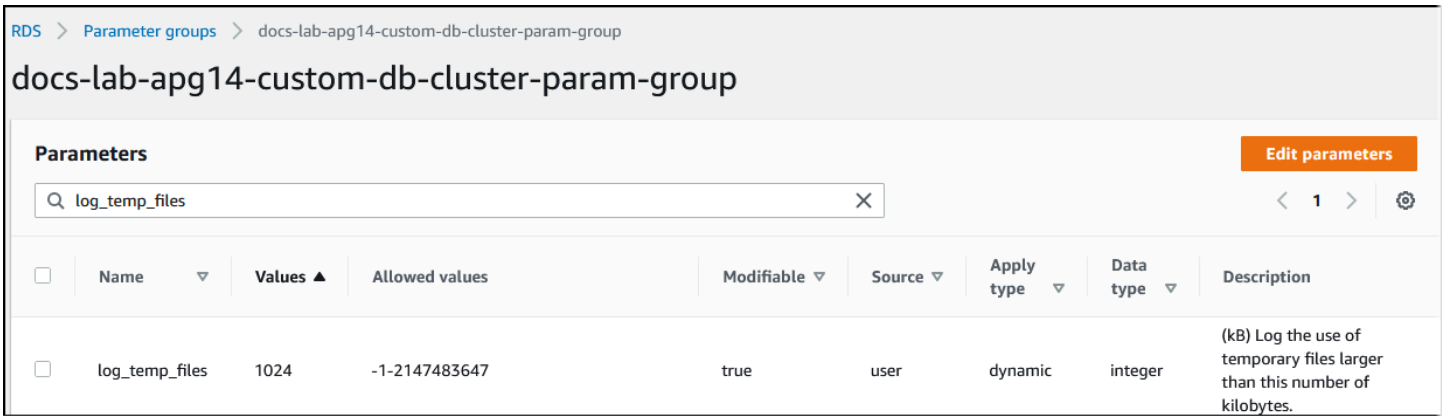
- ORDER BY
- DISTINCT
- GROUP BY
- JOIN (MERGE und HASH)

Der Abfrageplaner beeinflusst indirekt, wie Ihr Aurora PostgreSQL DB-Cluster Arbeitsspeicher verwendet. Der Abfrageplaner generiert Ausführungspläne für die Verarbeitung von SQL-Anweisungen. Ein bestimmter Plan kann eine komplexe Abfrage in mehrere Arbeitseinheiten aufteilen, die parallel ausgeführt werden können. Wenn möglich, verwendet Aurora PostgreSQL die Speichermenge, die im `work_mem`-Parameter für jede Sitzung vor dem Schreiben auf die Festplatte für jeden parallelen Prozess angegeben ist.

Mehrere Datenbankbenutzer, die mehrere Operationen gleichzeitig ausführen und mehrere Arbeitseinheiten parallel generieren, können den zugewiesenen Arbeitsspeicher Ihres Aurora PostgreSQL DB-Clusters erschöpfen. Dies kann zu einer übermäßigen Erstellung temporärer Dateien und Festplatten-I/O führen, oder schlimmer noch, es kann zu einem Speicherausfall führen.

Identifizieren temporärer Dateiverwendung

Immer, wenn der für die Verarbeitung von Abfragen erforderliche Speicher den im `work_mem`-Parameter angegebenen Wert übersteigt, werden die Arbeitsdaten in einer temporären Datei auf die Festplatte ausgelagert. Sie können sehen, wie oft dies geschieht, indem Sie den `log_temp_files`-Parameter aktivieren. Standardmäßig ist dieser Parameter deaktiviert (Einstellung auf -1). Um alle temporären Dateiinformationen zu erfassen, setzen Sie diesen Parameter auf 0. Setzen Sie `log_temp_files` auf eine andere positive Ganzzahl, um temporäre Dateiinformationen für Dateien zu erfassen, die dieser Datenmenge entsprechen oder größer sind (in Kilobyte). In der folgenden Abbildung sehen Sie ein Beispiel von AWS-Managementkonsole.



Nachdem Sie die temporäre Dateiprotokollierung konfiguriert haben, können Sie mit Ihrer eigenen Workload testen, ob Ihre Arbeitsspeichereinstellung ausreichend ist. Sie können eine Workload auch mithilfe von pgbench simulieren, einer einfachen Benchmarking-Anwendung aus der PostgreSQL-Community.

Das folgende Beispiel initialisiert (`-i`)`pgbench`, indem es die notwendigen Tabellen und Zeilen für die Ausführung der Tests erstellt. In diesem Beispiel erstellt der Skalierungsfaktor (`-s` 50) 50 Zeilen in der `pgbench_branches`-Tabelle, 500 Zeilen in `pgbench_tellers` und 5 000 000 Zeilen in der `pgbench_accounts`-Tabelle in der `labdb`Datenbank.

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -i -s 50 labdb
Password:
dropping old tables...
```

```
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85
 s, vacuum 2.30 s, primary keys 3.51 s)
```

Nach der Initialisierung der Umgebung können Sie den Benchmark für eine bestimmte Zeit (`-T`) und die Anzahl der Kunden (`-c`) ausführen. In diesem Beispiel wird auch die `-d`-Option zur Ausgabe von Debugging-Informationen verwendet, während die Transaktionen vom Aurora PostgreSQL DB-Cluster verarbeitet werden.

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -d -T 60 -c 10 labdb
Password:*******
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

Weitere Informationen über „pgbench“ finden Sie in pgbench in der PostgreSQL-Dokumentation.

Sie können den Befehl „psql metacommand“ (\d) verwenden, um die von „pgbench“ erstellten Relationen wie Tabellen, Ansichten und Indizes aufzulisten.

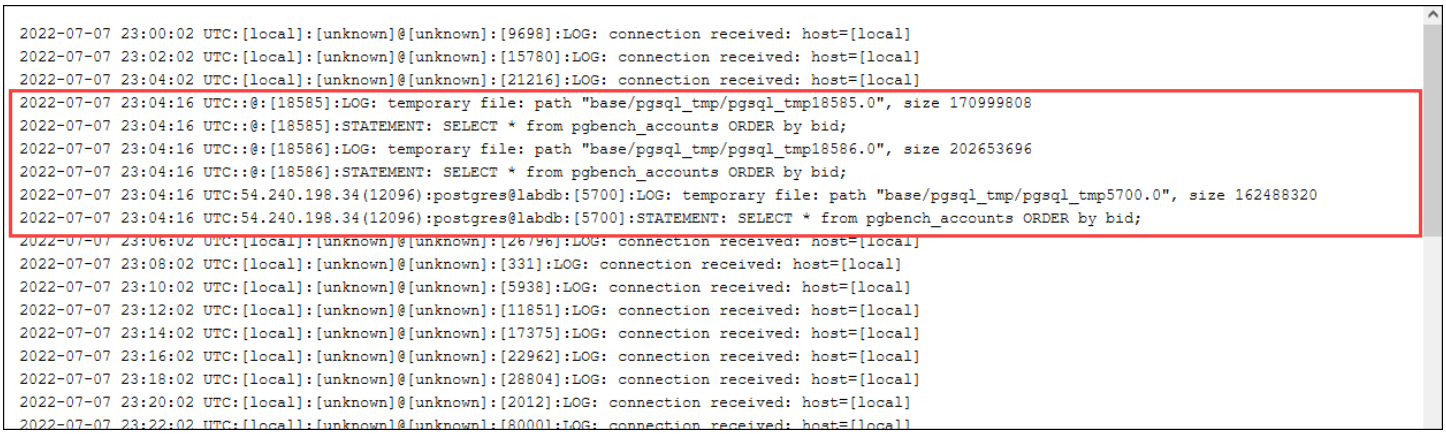
```
labdb=>  \d pgbench_accounts
 Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
```

```
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

Wie in der Ausgabe gezeigt, wird die `pgbench_accounts`-Tabelle nach der `aid`-Spalte indiziert. Um sicherzustellen, dass diese nächste Abfrage Arbeitsspeicher verwendet, fragen Sie eine beliebige nicht indizierte Spalte ab, z. B. die im folgenden Beispiel gezeigte.

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

Überprüfen Sie das Protokoll auf die temporären Dateien. Öffnen Sie dazu AWS-Managementkonsole, wählen Sie die Aurora PostgreSQL DB-Cluster-Instance und dann den Tab Logs & Events (Protokolle und Ereignisse). Zeigen Sie die Protokolle in der Konsole an oder laden Sie sie zur weiteren Analyse herunter. Wie in der folgenden Abbildung dargestellt, zeigt die Größe der temporären Dateien, die für die Verarbeitung der Abfrage benötigt werden, an, dass Sie erwägen sollten, die für die `work_mem`-Parameter angegebene Menge zu erhöhen.



Sie können diesen Parameter für Einzelpersonen und Gruppen unterschiedlich konfigurieren, je nach Ihren betrieblichen Anforderungen. Sie können beispielsweise den `work_mem`-Parameter auf 8 GB für die Rolle mit dem Namen `dev_team` festlegen.

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

Mit dieser Einstellung für `work_mem` wird jeder Rolle, die ein Mitglied der `dev_team`-Rolle ist, bis zu 8 GB Arbeitsspeicher zugewiesen.

Verwenden von Indizes für schnellere Reaktionszeit

Wenn Ihre Abfragen zu lange dauern, bis Ergebnisse zurückgegeben werden, können Sie überprüfen, ob Ihre Indizes erwartungsgemäß verwendet werden. Aktivieren Sie zuerst `\timing`, das psql-Metakommando, wie folgt.

```
postgres=>  \timing on
```

Verwenden Sie nach dem Aktivieren des Timings eine einfache SELECT-Anweisung.

```
postgres=> SELECT COUNT(*) FROM
  (SELECT * FROM pgbench_accounts
  ORDER BY bid)
  AS accounts;
count
-------
5000000
(1 row)
Time: 3119.049 ms (00:03.119)
```

Wie in der Ausgabe gezeigt, dauerte die Ausführung dieser Abfrage etwas mehr als 3 Sekunden. Um die Reaktionszeit zu verbessern, erstellen Sie einen Index für `pgbench_accounts` wie folgt.

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

Führen Sie die Abfrage erneut aus und beachten Sie die kürzere Antwortzeit. In diesem Beispiel wurde die Abfrage etwa fünfmal schneller abgeschlossen, in etwa einer halben Sekunde.

```
postgres=>  SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS
 accounts;
 count
-------
 5000000
(1 row)
Time: 567.095 ms
```

Arbeitsspeicher für logische Dekodierung anpassen

Logische Replikation war in allen Versionen von Aurora PostgreSQL seit seiner Einführung in PostgreSQL Version 10 verfügbar. Wenn Sie die logische Replikation konfigurieren, können Sie auch

die `logical_decoding_work_mem`-Parameter einstellen, um die Speichermenge anzugeben, die der logische Dekodierungsprozess für den Dekodierungs- und Streaming-Prozess verwenden kann.

Bei der logischen Dekodierung werden Write-Ahead-Log-Datensätze (WAL) in SQL-Anweisungen konvertiert, die dann zur logischen Replikation oder einer anderen Aufgabe an ein anderes Ziel gesendet werden. Wenn eine Transaktion in die WAL geschrieben und dann konvertiert wird, muss die gesamte Transaktion in den für `logical_decoding_work_mem` angegebenen Wert passen. Standardmäßig ist dieser Parameter auf 65 536 KB eingestellt. Jeder Überlauf wird auf den Datenträger geschrieben. Dies bedeutet, dass es erneut von der Festplatte gelesen werden muss, bevor es an sein Ziel gesendet werden kann, wodurch der Gesamtprozess verlangsamt wird.

Sie können den Betrag des Transaktionsüberlaufs in Ihre aktuelle Workload zu einem bestimmten Zeitpunkt bewerten, indem Sie die `aurora_stat_file`-Funktion wie im folgenden Beispiel gezeigt, verwenden.

```
SELECT split_part (filename, '/', 2)
   AS slot_name, count(1) AS num_spill_files,
   sum(used_bytes) AS slot_total_bytes,
   pg_size_pretty(sum(used_bytes)) AS slot_total_size
   FROM aurora_stat_file()
   WHERE filename like '%spill%'
   GROUP BY 1;
  slot_name | num_spill_files | slot_total_bytes | slot_total_size
------------+-----------------+------------------+-----------------
 slot_name  |       590       |      411600000   | 393 MB
(1 row)
```

Diese Abfrage gibt die Anzahl und Größe der Spill-Dateien auf Ihrem Aurora PostgreSQL DB-Cluster zurück, wenn die Abfrage aufgerufen wird. Bei länger laufenden Workloads befinden sich möglicherweise noch keine Spill-Dateien auf der Festplatte. Um ein Profil lang andauernder Workloads zu erstellen, empfehlen wir Ihnen, eine Tabelle zu erstellen, in der die Informationen der Spill-Datei während der Ausführung der Workload erfasst werden. Sie können die Tabelle wie folgt erstellen.

```
CREATE TABLE spill_file_tracking AS
    SELECT now() AS spill_time,*
    FROM aurora_stat_file()
    WHERE filename LIKE '%spill%';
```

Um zu sehen, wie Spill-Dateien während der logischen Replikation verwendet werden, richten Sie einen Publisher und einen Abonnenten ein und starten Sie dann eine einfache Replikation. Weitere Informationen finden Sie unter Einrichten der logischen Replikation für Ihren DB-Cluster von Aurora PostgreSQL. Während der Replikation können Sie einen Auftrag erstellen, der die Ergebnismenge aus der `aurora_stat_file()`-spill-File-Funktion wie folgt erfasst.

```
INSERT INTO spill_file_tracking
  SELECT now(),*
  FROM aurora_stat_file()
  WHERE filename LIKE '%spill%';
```

Verwenden Sie den folgenden psql-Befehl, um den Job einmal pro Sekunde auszuführen.

```
\watch 0.5
```

Stellen Sie während der Ausführung des Auftrags eine Verbindung zur Writer-Instance von einer anderen psql-Sitzung her. Verwenden Sie die folgende Reihe von Anweisungen, um eine Workload auszuführen, die die Speicherkonfiguration überschreitet und Aurora PostgreSQL veranlasst, eine Spill-Datei zu erstellen.

```
labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
UPDATE 10000001
```

Diese Anweisungen dauern einige Minuten. Wenn Sie fertig sind, drücken Sie die Strg-Taste und die C-Taste gleichzeitig, um die Überwachungsfunktion zu beenden. Verwenden Sie dann den folgenden Befehl, um eine Tabelle zu erstellen, in der die Informationen über die Verwendung der Spill-Datei des Aurora PostgreSQL DB-Clusters gespeichert werden.

```
SELECT spill_time, split_part (filename, '/', 2)
    AS slot_name, count(1)
    AS spills, sum(used_bytes)
    AS slot_total_bytes, pg_size_pretty(sum(used_bytes))
    AS slot_total_size FROM spill_file_tracking
  GROUP BY 1,2 ORDER BY 1;
                  spill_time | slot_name             | spills | slot_total_bytes |
 slot_total_size
```

```
------------------------------+----------------------+--------+-----------------
+-----------------
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1      | 142352280        | 136
 MB
2022-04-15 14:11:33.962216+00 | replication_slot_name | 4      | 467637996        | 446
 MB
2022-04-15 14:12:00.997636+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:03.030245+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:05.059761+00 | replication_slot_name | 5      | 618410996        | 590
 MB
2022-04-15 14:12:07.22905+00  | replication_slot_name | 5      | 640585316        | 611
 MB
(6 rows)
```

Die Ausgabe zeigt, dass beim Ausführen des Beispiels fünf Spill-Dateien erstellt wurden, die 611 MB Speicher verbrauchten. Um zu vermeiden, dass auf den Datenträger geschrieben wird, empfehlen wir, den `logical_decoding_work_mem`-Parameter auf die nächsthöhere Speichergröße, 1024, einzustellen.

Verwendung von CloudWatch Amazon-Metriken zur Analyse der Ressourcennutzung für Aurora PostgreSQL

Aurora sendet automatisch Metrikdaten innerhalb von 1 Minute an CloudWatch . Sie können die Ressourcennutzung für Aurora PostgreSQL mithilfe CloudWatch von Metriken analysieren. Mit den Metriken können Sie den Netzwerkdurchsatz und die Netzwerknutzung bewerten.

Bewertung des Netzwerkdurchsatzes mit CloudWatch

Wenn sich Ihre Systemauslastung den Ressourcenlimits für Ihren Instance-Typ nähert, kann sich die Verarbeitung verlangsamen. Sie können CloudWatch Logs Insights verwenden, um die Nutzung Ihrer Speicherressourcen zu überwachen und sicherzustellen, dass ausreichend Ressourcen verfügbar sind. Bei Bedarf können Sie die DB-Instance auf eine größere Instance-Klasse umstellen.

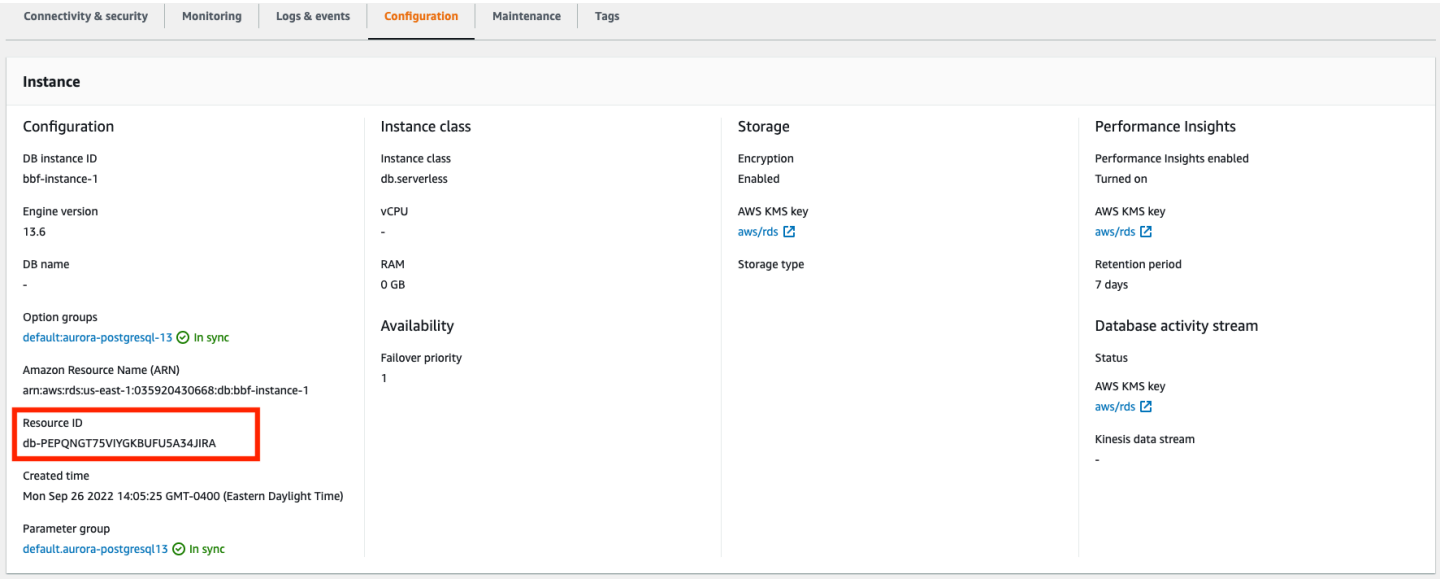
Die Verarbeitung des Aurora-Speichers kann aus folgenden Gründen langsam sein:

- Unzureichende Netzwerkbandbreite zwischen Client und DB-Instance.
- Unzureichende Netzwerkbandbreite für das Speichersubsystem.
- Eine Workload, die im Hinblick auf Ihren Instance-Typ groß ist.

Sie können CloudWatch Logs Insights abfragen, um ein Diagramm der Aurora-Speicherressourcennutzung zur Überwachung der Ressourcen zu erstellen. Das Diagramm zeigt die CPU-Auslastung und Metriken, anhand derer Sie entscheiden können, ob Sie auf eine größere Instance-Größe hochskalieren sollten. Informationen zur Abfragesyntax für CloudWatch Logs Insights finden Sie unter CloudWatch Logs Insights-Abfragesyntax

Um es zu verwenden CloudWatch, müssen Sie Ihre Aurora PostgreSQL-Protokolldateien nach exportieren. CloudWatch Sie können Ihren vorhandenen Cluster auch ändern, um Logs in diesen zu exportieren. CloudWatch Informationen zum Exportieren von Protokollen nach CloudWatch finden Sie unterAktivieren der Option zum Veröffentlichen von Protokollen in Amazon CloudWatch.

Sie benötigen die Ressourcen-ID Ihrer DB-Instance, um CloudWatch Logs Insights abzufragen. Die Resource ID (Ressourcen-ID) ist auf der Registerkarte Configuration (Konfiguration) in Ihrer Konsole verfügbar:



So fragen Sie Ihre Protokolldateien nach Metriken für den Ressourcenspeicher ab:

1. Öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.

   Die CloudWatch Übersichts-Startseite wird angezeigt.
2. Ändern Sie, falls erforderlich, die AWS-Region. Wählen Sie in der Navigationsleiste aus, AWS-Region wo sich Ihre AWS Ressourcen befinden. Weitere Informationen finden Sie unter - Regionen und Endpunkte.
3. Wählen Sie im Navigationsbereich Logs (Protokolle) und dann Logs Insights aus.

   Die Seite Logs Insights wird angezeigt.

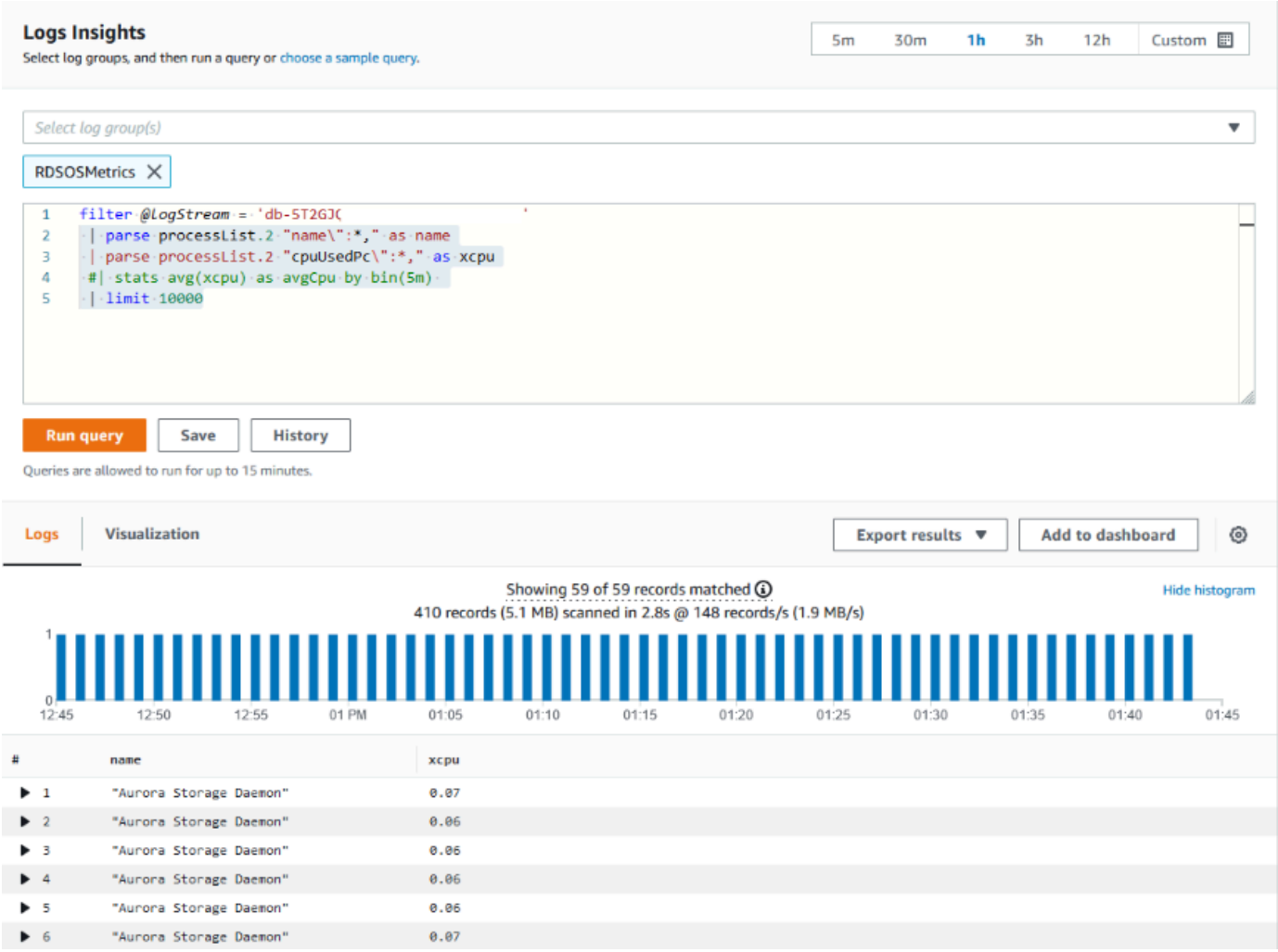
4. Wählen Sie die Protokolldateien, die Sie analysieren möchten, aus der Dropdown-Liste aus.
5. Geben Sie die folgende Abfrage in das Feld ein und ersetzen Sie `<resource ID>` durch die Ressourcen-ID Ihres DB-Clusters:

   ```
   filter @logStream = <resource ID> | parse @message "\"Aurora Storage Daemon\"*memoryUsedPc\":*,\"cpuUsedPc\":*," as a,memoryUsedPc,cpuUsedPc | display memoryUsedPc,cpuUsedPc #| stats avg(xcpu) as avgCpu by bin(5m) | limit 10000
   ```

6. Klicken Sie auf Run query (Abfrage ausführen).

   Das Diagramm zur Speicherauslastung wird angezeigt.

   Die folgende Abbildung veranschaulicht die Seite Logs Insights und die Diagrammanzeige.



Bewertung der DB-Instance-Nutzung für Aurora PostgreSQL anhand von Metriken CloudWatch

Mithilfe von CloudWatch Metriken können Sie den Durchsatz Ihrer Instance beobachten und herausfinden, ob Ihre Instance-Klasse ausreichend Ressourcen für Ihre Anwendungen bereitstellt. Informationen zu Ihren DB-Instance-Klassenlimits finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora. Suchen Sie die Spezifikationen für Ihre DB-Instance-Klasse, um Ihre Netzwerkleistung zu ermitteln.

Wenn sich Ihre DB-Instance-Nutzung dem Instance-Klassenlimit nähert, kann sich die Leistung verschlechtern. Die CloudWatch Metriken können diese Situation bestätigen, sodass Sie planen können, manuell auf eine größere Instance-Klasse hochzuskalieren.

Kombinieren Sie die folgenden CloudWatch Metrikwerte, um herauszufinden, ob Sie sich dem Instance-Klassenlimit nähern:

- NetworkThroughput— Die Menge des Netzwerkdurchsatzes, der von den Clients für jede Instance im Aurora-DB-Cluster empfangen und übertragen wird. Dieser Durchsatzwert berücksichtigt nicht den Netzwerkdatenverkehr zwischen den Instances im DB-Cluster und dem Cluster-Volume.
- StorageNetworkThroughput— Die Menge des Netzwerkdurchsatzes, der von jeder Instance im Aurora-DB-Cluster empfangen und an das Aurora-Speichersubsystem gesendet wurde.

Fügen Sie den hinzu NetworkThroughput, StorageNetworkThroughputum den Netzwerkdurchsatz zu ermitteln, der von jeder Instance in Ihrem Aurora-DB-Cluster vom Aurora-Speichersubsystem empfangen und an dieses gesendet wurde. Das Instance-Klassenlimit für Ihre Instance sollte größer sein als die Summe dieser beiden kombinierten Metriken.

Sie können die folgenden Metriken verwenden, um zusätzliche Details des Netzwerkverkehrs Ihrer Client-Anwendungen beim Senden und Empfangen zu überprüfen:

- NetworkReceiveThroughput— Die Menge des Netzwerkdurchsatzes, den jede Instance im Aurora PostgreSQL-DB-Cluster von Clients erhält. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances im -DB-Cluster und dem Cluster-Volume.
- NetworkTransmitThroughput – Der Umfang des von Clients gesendeten Netzwerkdurchsatzes für jede Instance im Aurora-DB-Cluster Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances im -DB-Cluster und dem Cluster-Volume.
- StorageNetworkReceiveThroughput – Der Umfang des vom Aurora-Speicheruntersystem empfangenen Netzwerkdurchsatzes für jede Instance im DB-Cluster

- StorageNetworkTransmitThroughput— Die Menge des Netzwerkdurchsatzes, der von jeder Instance im DB-Cluster an das Aurora-Speichersubsystem gesendet wird.

Addieren Sie all diese Metriken, um zu bewerten, wie Ihre Netzwerknutzung im Vergleich zum Instance-Klassenlimit abschneidet. Das Instance-Klassenlimit sollte größer sein als die Summe dieser kombinierten Metriken.

Die Netzwerklimits und die CPU-Auslastung für Speicher bedingen sich gegenseitig. Wenn der Netzwerkdurchsatz steigt, steigt auch die CPU-Auslastung. Die Überwachung der CPU- und Netzwerknutzung gibt Aufschluss darüber, wie und warum die Ressourcen erschöpft werden.

Sie können Folgendes in Betracht ziehen, um die Netzwerknutzung zu minimieren:

- Verwenden einer größeren Instance-Klasse
- Verwenden von `pg_partman`-Partitionierungsstrategien
- Aufteilen der Schreibanforderungen in Batches, um die Gesamttransaktionen zu reduzieren
- Weiterleiten der schreibgeschützten Workload an eine schreibgeschützte Instance
- Löschen aller ungenutzten Indizes
- Suchen nach aufgeblähten Objekten und VACUUM. Verwenden Sie bei stark aufgeblähten Objekten die PostgreSQL-Erweiterung `pg_repack`. Weitere Informationen zu `pg_repack` finden Sie unter Reorganisieren von Tabellen in PostgreSQL-Datenbanken mit minimalen Sperren.

Verwenden der logischen Replikation, um ein Hauptversions-Upgrade für Aurora PostgreSQL durchzuführen

Wenn Sie die logische Replikation und schnelles Aurora-Klonen verwenden, können Sie ein Hauptversions-Upgrade unter Verwendung der aktuellen Version der Aurora-PostgreSQL-Datenbank durchführen, während Sie die sich ändernden Daten schrittweise auf die neue Hauptversionsdatenbank migrieren. Dieser Upgrade-Prozess mit geringen Ausfallzeiten wird als Blau/Grün-Upgrade bezeichnet. Die aktuelle Version der Datenbank wird als „blaue“ Umgebung und die neue Datenbankversion als „grüne“ Umgebung bezeichnet.

Dank des schnellen Klonens von Aurora werden die vorhandenen Daten vollständig geladen, indem ein Snapshot der Quelldatenbank erstellt wird. Das schnelle Klonen verwendet ein Copy-on-Write-Protokoll, das auf der Aurora-Speicherschicht aufbaut, und ermöglicht Ihnen, in kurzer Zeit einen Datenbankklon zu erstellen. Diese Methode ist sehr effektiv beim Upgrade auf eine große Datenbank.

Die logische Replikation in PostgreSQL verfolgt und überträgt Ihre Datenänderungen von der ursprünglichen Instance auf eine neue Instance, die parallel läuft, bis Sie zur neueren Version von PostgreSQL wechseln. Für die logische Replikation wird ein Veröffentlichungs- und Abonnementmodell verwendet. Weitere Informationen über die logische Replikation von Aurora PostgreSQL finden Sie unter Replikation mit Amazon Aurora PostgreSQL.

> **Tip**
>
> Sie können die für ein Hauptversion-Upgrade erforderliche Ausfallzeit minimieren, indem Sie die verwaltete Blau/Grün-Bereitstellungsfunktion von Amazon RDS verwenden. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

Themen

- Voraussetzungen
- Einschränkungen
- Festlegen und Überprüfen von Parameterwerten
- Durchführen eines Upgrades von Aurora PostgreSQL auf eine neue Hauptversion
- Durchführen von Aufgaben nach dem Upgrade

Voraussetzungen

Sie müssen die folgenden Anforderungen erfüllen, um diesen Upgrade-Prozess mit geringen Ausfallzeiten durchzuführen:

- Sie müssen über rds_superuser-Berechtigungen verfügen.
- Auf dem DB-Cluster von Aurora PostgreSQL, den Sie aktualisieren möchten, muss eine unterstützte Version ausgeführt werden, die mithilfe logischer Replikation Hauptversions-Upgrades durchführen kann. Sie müssen alle Nebenversions-Updates und -Patches auf Ihren DB-Cluster anwenden. Die `aurora_volume_logical_start_lsn`-Funktion, die bei dieser Methode zum Einsatz kommt, wird in den folgenden Versionen von Aurora PostgreSQL unterstützt:
  - 15.2 und höhere 15-Versionen
  - 14.3 und höhere 14-Versionen
  - 13.6 und höhere 13-Versionen

- 12.10 und höhere 12-Versionen
- 11.15 und höhere 11-Versionen
- 10.20 und höhere 10-Versionen

Weitere Informationen über die `aurora_volume_logical_start_lsn`-Funktion finden Sie unter aurora_volume_logical_start_lsn.

- Alle Ihre Tabellen müssen einen Primärschlüssel haben oder eine PostgreSQL-Identitätsspalte enthalten.
- Konfigurieren Sie die Sicherheitsgruppe für Ihre VPC, um eingehenden und ausgehenden Zugriff zwischen den beiden DB-Clustern von Aurora PostgreSQL (dem alten und dem neuen) zu ermöglichen. Sie können Zugriff auf einen speziellen Classless Inter-Domain Routing (CIDR)-Bereich oder auf eine andere Sicherheitsgruppe in Ihrer VPC oder einer Peer-VPC gewähren. (Eine Peer-VPC erfordert eine VPC-Peering-Verbindung.)

> **Note**
>
> Detaillierte Informationen zu den Berechtigungen, die für die Konfiguration und Verwaltung eines laufenden logischen Replikationsszenarios erforderlich sind, finden Sie in der PostgreSQL-Core-Dokumentation.

Einschränkungen

Wenn Sie ein Upgrade mit niedriger Ausfallzeit für Ihren DB-Cluster von Aurora PostgreSQL durchführen, um ihn auf eine neue Hauptversion zu aktualisieren, verwenden Sie die native logische PostgreSQL-Replikationsfunktion. Sie weist die gleichen Fähigkeiten und Einschränkungen wie die logische PostgreSQL-Replikation auf. Weitere Informationen finden Sie unter Die logische Replikation von PostgreSQL.

- Data Definition Language (DDL)-Befehle werden nicht repliziert.
- Die Replikation unterstützt keine Schemaänderungen in einer Live-Datenbank. Das Schema wird während des Klonvorgangs in seiner ursprünglichen Form neu erstellt. Wenn Sie das Schema nach dem Klonen, aber vor Abschluss des Upgrades ändern, werden die Änderungen nicht in der aktualisierten Instance angezeigt.
- Große Objekte werden nicht repliziert, aber Sie können Daten in normalen Tabellen speichern.

- Die Replikation wird nur von Tabellen unterstützt, einschließlich partitionierter Tabellen. Die Replikation auf andere Relationstypen wie Ansichten, materialisierte Ansichten oder fremde Tabellen wird nicht unterstützt.
- Sequenzdaten werden nicht repliziert und müssen nach dem Failover manuell aktualisiert werden.

> **Note**
>
> Dieses Upgrade unterstützt kein Auto-Scripting. Sie sollten alle Schritte manuell ausführen.

Festlegen und Überprüfen von Parameterwerten

Vor dem Upgrade müssen Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL so konfigurieren, dass sie als Veröffentlichungsserver fungiert. Die Instance sollte eine benutzerdefinierte DB-Cluster-Parametergruppe mit den folgenden Einstellungen verwenden:

- `rds.logical_replication` – Stellen Sie diesen Parameter auf 1 ein. Der `rds.logical_replication`-Parameter dient demselben Zweck wie der Parameter `wal_level` eines eigenständigen PostgreSQL-Servers und andere Parameter, die die Write-Ahead-Protokolldateiverwaltung steuern.
- `max_replication_slots` – Legen Sie diesen Parameter auf die Gesamtzahl der Abonnements fest, die Sie erstellen möchten. Wenn Sie AWS DMS verwenden, legen Sie diesen Parameter auf die Anzahl von AWS DMS-Aufgaben fest, die Sie für die Erfassung von geänderten Daten aus diesem DB-Cluster verwenden möchten.
- `max_wal_senders` – Legen Sie die Anzahl der gleichzeitigen Verbindungen sowie einige zusätzliche Verbindungen fest, um sie für Verwaltungsaufgaben und neue Sitzungen verfügbar zu machen. Wenn Sie AWS DMS verwenden, sollte die Anzahl der max_wal_senders der Anzahl der gleichzeitigen Sitzungen plus der Anzahl der AWS DMS-Aufgaben entsprechen, die zu einem bestimmten Zeitpunkt ausgeführt werden können.
- `max_logical_replication_workers` – Legen Sie diesen Parameter auf die Anzahl der Worker für logische Replikation und Tabellensynchronisierung fest, die Sie erwarten. Im Allgemeinen ist es sicher, die Anzahl der Replikations-Worker auf den gleichen Wert festzulegen, der für max_wal_senders verwendet wird. Die Worker stammen aus dem Pool der Hintergrundprozesse (max_worker_processes), die dem Server zugewiesen sind.
- `max_worker_processes` – Legen Sie diesen Parameter auf die Anzahl der Hintergrundprozesse für den Server fest. Diese Anzahl sollte groß genug sein, um Worker für Replikation,

Selbstbereinigungsprozesse und andere Wartungsprozesse, die möglicherweise gleichzeitig stattfinden, zur Verfügung zu stellen.

Wenn Sie auf eine neuere Version von Aurora PostgreSQL aktualisieren, müssen Sie alle Parameter duplizieren, die Sie in der früheren Version der Parametergruppe geändert haben. Diese Parameter werden auf die aktualisierte Version angewendet. Sie können die `pg_settings`-Tabelle abfragen, um eine Liste der Parametereinstellungen zu erhalten, sodass Sie sie in Ihrem neuen DB-Cluster von Aurora PostgreSQL neu erstellen können.

Führen Sie beispielsweise die folgende Abfrage aus, um die Einstellungen für Replikationsparameter abzurufen:

```
SELECT name, setting FROM pg_settings WHERE name in
('rds.logical_replication', 'max_replication_slots',
'max_wal_senders', 'max_logical_replication_workers',
'max_worker_processes');
```

Durchführen eines Upgrades von Aurora PostgreSQL auf eine neue Hauptversion

So bereiten Sie den Herausgeber vor (blau)

1. Im folgenden Beispiel ist die Writer-Quell-Instance (blau) ein DB-Cluster von Aurora PostgreSQL, auf dem PostgreSQL Version 11.15 ausgeführt wird. Dies ist der Veröffentlichungsknoten in unserem Replikationsszenario. Für diese Demonstration hostet unsere Writer-Quell-Instance eine Beispieltabelle, die eine Reihe von Werten enthält:

   ```
   CREATE TABLE my_table (a int PRIMARY KEY);
   INSERT INTO my_table VALUES (generate_series(1,100));
   ```

2. Um eine Publikation auf der Quell-Instance zu erstellen, stellen Sie mit psql (die CLI für PostgreSQL) oder mit dem Client Ihrer Wahl eine Verbindung zum Writer-Knoten der Instance her. Geben Sie in jeder Datenbank den folgenden Befehl ein:

   ```
   CREATE PUBLICATION publication_name FOR ALL TABLES;
   ```

   Der Wert publication_name gibt den Namen der Publikation an.

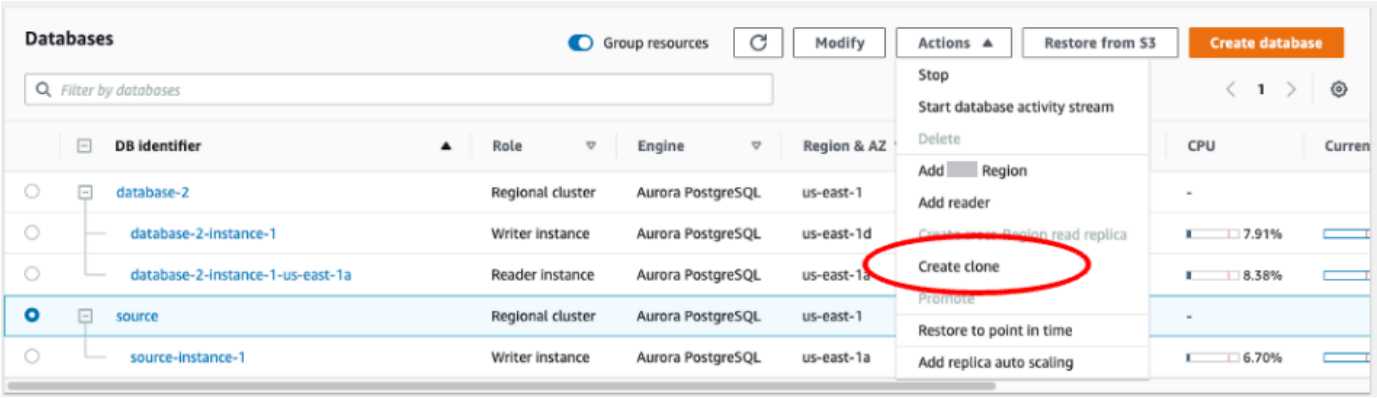
3. Sie müssen auch einen Replikations-Slot auf der Instance erstellen. Der folgende Befehl erstellt einen Replikations-Slot und lädt das logische Dekodierungs-Plug-in `pgoutput`. Das Plug-in

ändert den von Write-Ahead Logging (WAL) in das logische Replikationsprotokoll gelesenen Inhalt und filtert die Daten gemäß der Veröffentlichungsspezifikation.

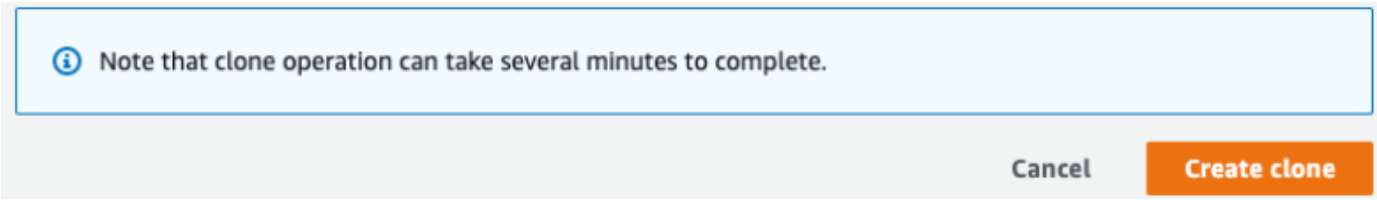
```
SELECT pg_create_logical_replication_slot('replication_slot_name', 'pgoutput');
```

So klonen Sie den Herausgeber

1. Verwenden Sie die Amazon-RDS-Konsole, um einen Klon der Quell-Instance zu erstellen. Markieren Sie den Instance-Namen in der Amazon-RDS-Konsole und wählen Sie dann im Menü Actions (Aktionen) die Option Create Clone (Klon erstellen) aus.



2. Geben Sie einen eindeutigen Namen für die Instance ein. Die meisten Einstellungen sind Standardeinstellungen aus der Quell-Instance. Wenn Sie die für die neue Instance erforderlichen Änderungen vorgenommen haben, wählen Sie Create clone (Klon erstellen) aus.



3. Während die Ziel-Instance initiiert wird, wird in der Spalte Status des Writer-Knotens Creating angezeigt. Sobald die Instance bereit ist, ändert sich ihr Status in „Available“.

So bereiten Sie den Klon für ein Upgrade vor

1. Der Klon ist die „grüne“ Instance im Bereitstellungsmodell. Er ist der Host des Replikationsabonnementknotens. Wenn der Knoten verfügbar ist, stellen Sie eine Verbindung mit psql her und fragen Sie den neuen Writer-Knoten ab, um die Log-Sequenznummer (LSN) zu erhalten. Die LSN identifiziert den Anfang eines Datensatzes im WAL-Stream.

```
SELECT aurora_volume_logical_start_lsn();
```

2. Die LSN-Nummer finden Sie in der Antwort auf die Abfrage. Da Sie diese Nummer zu einem späteren Zeitpunkt des Vorgangs benötigen, sollten Sie sich diese notieren.

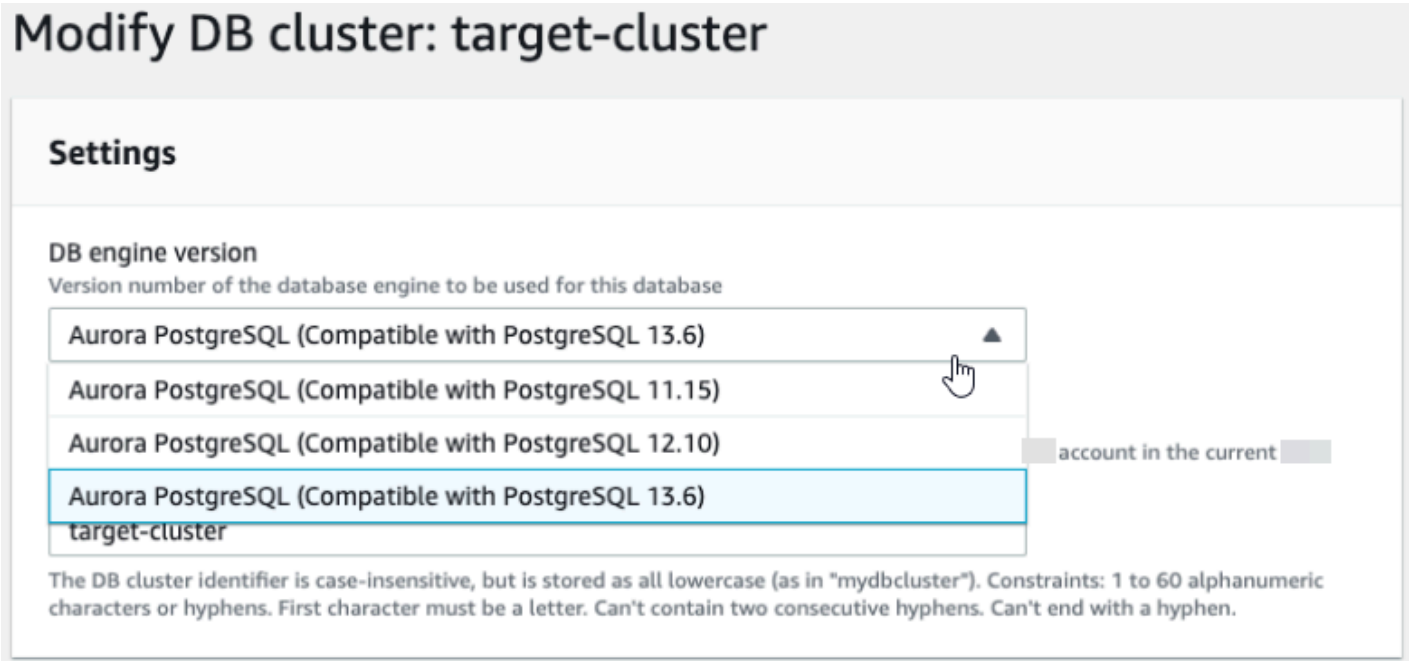
```
postgres=> SELECT aurora_volume_logical_start_lsn();
aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

3. Bevor Sie den Klon aktualisieren, löschen Sie den Replikationsslot des Klons.

```
SELECT pg_drop_replication_slot('replication_slot_name');
```

So aktualisieren Sie einen Cluster auf eine neue Hauptversion

- Verwenden Sie nach dem Klonen des Provider-Knotens die Amazon-RDS-Konsole, um ein Hauptversions-Upgrade auf dem Abonnementknoten zu initiieren. Markieren Sie den Instance-Namen in der RDS-Konsole und klicken Sie auf die Schaltfläche Modify (Ändern). Wählen Sie die aktualisierte Version und Ihre aktualisierten Parametergruppen aus und wenden Sie die Einstellungen sofort an, um die Ziel-Instance zu aktualisieren.



- Sie können auch die CLI verwenden, um ein Upgrade durchzuführen:

```
aws rds modify-db-cluster –db-cluster-identifier $TARGET_Aurora_ID –engine-version
 13.6 –allow-major-version-upgrade –apply-immediately
```

So bereiten Sie den Abonnenten (grün) vor

1. Sobald der Klon nach dem Upgrade verfügbar ist, stellen Sie eine Verbindung mit psql her und definieren Sie das Abonnement. Dazu müssen Sie die folgenden Optionen im `CREATE SUBSCRIPTION`-Befehl angeben:

   - `subscription_name` – Der Name des Abonnements.
   - `admin_user_name` – Der Name eines Administratorbenutzers mit rds_superuser-Berechtigungen.
   - `admin_user_password` – Das Passwort, das dem Administrator zugeordnet ist.
   - `source_instance_URL` – Die URL der Veröffentlichungsserver-Instance.
   - `database` – Die Datenbank, mit der der Abonnementserver eine Verbindung herstellen wird.
   - `publication_name` – Der Name des Veröffentlichungsservers.
   - `replication_slot_name` – Der Name der Replikationsgruppe.

   ```
   CREATE SUBSCRIPTION subscription_name
   CONNECTION 'postgres://admin_user_name:admin_user_password@source_instance_URL/
   database' PUBLICATION publication_name
   WITH (copy_data = false, create_slot = false, enabled = false, connect = true,
    slot_name = 'replication_slot_name');
   ```

2. Nachdem Sie das Abonnement erstellt haben, fragen Sie die Ansicht pg_replication_origin ab, um den Roname-Wert abzurufen. Dieser Wert ist die ID des Replikationsursprungs. Jede Instance hat einen `roname`-Wert:

   ```
   SELECT * FROM pg_replication_origin;
   ```

   Zum Beispiel:

   ```
   postgres=>
   SELECT * FROM pg_replication_origin;
   ```

```
roident | roname
---------+----------
1 | pg_24586
```

3. Geben Sie im Befehl die LSN an, die Sie aus der vorherigen Abfrage des Veröffentlichungsknotens gespeichert haben, und den `roname`-Wert, der vom Abonnementknoten [INSTANCE] zurückgegeben wurde. Dieser Befehl verwendet die pg_replication_origin_advance-Funktion, um den Ausgangspunkt in der Protokollsequenz für die Replikation anzugeben.

   ```
   SELECT pg_replication_origin_advance('roname', 'log_sequence_number');
   ```

   `roname` ist die ID, die von der Ansicht pg_replication_origin zurückgegeben wird.

   `log_sequence_number` ist der Wert, der von der vorherigen Abfrage der `aurora_volume_logical_start_lsn`-Funktion zurückgegeben wurde.

4. Verwenden Sie dann die `ALTER SUBSCRIPTION... ENABLE`-Klausel, um die logische Replikation zu aktivieren.

   ```
   ALTER SUBSCRIPTION subscription_name ENABLE;
   ```

5. An dieser Stelle können Sie bestätigen, dass die Replikation funktioniert. Fügen Sie der Veröffentlichungs-Instance einen Wert hinzu und bestätigen Sie dann, dass der Wert auf den Abonnementknoten repliziert wird.

   Verwenden Sie dann den folgenden Befehl, um die Replikationsverzögerung auf dem Veröffentlichungsknoten zu überwachen:

   ```
   SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
    pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
   confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
   confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
    'logical';
   ```

   Zum Beispiel:

   ```
   postgres=> SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
    pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
   ```

```
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
 confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';

current_time                   | slot_name             | active | active_pid |
 diff_size | diff_bytes
-------------------------------+-----------------------+--------+------------
+-----------+------------
2022-04-13 15:11:00.243401+00  | replication_slot_name | t      | 21854      | 136
 bytes | 136
(1 row)
```

Sie können die Replikationsverzögerung mit den Werten `diff_size` und`diff_bytes` überwachen. Wenn diese Werte 0 erreichen, hat das Replikat den Stand der Quell-DB-Instance erreicht.

Durchführen von Aufgaben nach dem Upgrade

Wenn das Upgrade abgeschlossen ist, wird der Instance-Status in der Spalte Status des Konsolen-Dashboards als Available angezeigt. Für die neue Instance empfehlen wir Ihnen Folgendes:

- Leiten Sie Ihre Anwendungen so um, dass sie auf den Writer-Knoten verweisen.
- Fügen Sie Reader-Knoten hinzu, um die Fallzahl zu verwalten und eine hohe Verfügbarkeit zu gewährleisten, falls ein Problem mit dem Writer-Knoten auftritt.
- DB-Cluster von Aurora PostgreSQL erfordern gelegentlich Betriebssystemupdates. Diese Updates können eine neuere Version der Glibc-Bibliothek umfassen. Bei solchen Updates empfehlen wir Ihnen, die unter In Aurora PostgreSQL unterstützte Sortierungen beschriebenen Richtlinien zu befolgen.
- Aktualisieren Sie die Benutzerberechtigungen für die neue Instance, um den Zugriff sicherzustellen.

Nachdem Sie Ihre Anwendung und Daten auf der neuen Instance getestet haben, empfehlen wir Ihnen, ein letztes Backup Ihrer ersten Instance zu erstellen, bevor Sie sie entfernen. Weitere Informationen zur Verwendung der logischen Replikation auf einem Aurora-Host finden Sie unter Einrichten der logischen Replikation für Ihren DB-Cluster von Aurora PostgreSQL.

Fehlerbehebung bei Speicherproblemen in Aurora PostgreSQL

Wenn der von Sortier- oder Indexerstellungsoperationen benötigte Arbeitsspeicher den vom `work_mem`-Parameter zugewiesene Arbeitsspeicher überschreitet, schreibt Aurora PostgreSQL die überschüssigen Daten in temporäre Datenträgerdateien. Dabei verwendet Aurora Postgre SQL den gleichen Speicherort wie für Fehler- und Meldungsprotokolle, d. h. den lokalen Speicher. Für jede Instance in Ihrem DB-Cluster von Aurora PostgreSQL ist eine bestimmte Menge an lokalem Speicher verfügbar. Die Speichermenge basiert auf der entsprechenden DB-Instance-Klasse. Wenn Sie den lokalen Speicherplatz erhöhen möchten, müssen Sie die Instance so ändern, dass sie eine größere DB-Instance-Klasse verwendet. Spezifikationen für DB-Instance-Klassen finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

Sie können den lokalen Speicherplatz Ihres DB-Clusters von Aurora PostgreSQL überwachen, indem Sie sich die Amazon-CloudWatch-Metrik für `FreeLocalStorage` ansehen. Diese Metrik meldet die Menge des verfügbaren Speicherplatzes jeder DB-Instance im Aurora-DB-Cluster für temporäre Tabellen und Protokolle. Weitere Informationen finden Sie unter Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch.

Sortier-, Indizierungs- und Gruppierungsoperationen beginnen im Arbeitsspeicher, müssen jedoch häufig in den lokalen Speicher ausgelagert werden. Wenn für Ihren DB-Cluster von Aurora PostgreSQL aufgrund dieser Art von Operationen der lokale Speicherplatz nicht ausreicht, können Sie das Problem lösen, indem Sie eine der folgenden Maßnahmen ergreifen.

- Rüsten Sie den Arbeitsspeicher auf. Dadurch muss weniger lokaler Speicher verwendet werden. Standardmäßig weist PostgreSQL 4 MB für jede Sortier-, Gruppierungs- und Indizierungsoperation zu. Wenn Sie den aktuellen Arbeitsspeicherwert für die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL überprüfen möchten, stellen Sie eine Verbindung mit der Instance über `psql` her und führen Sie den folgenden Befehl aus.

  ```
  postgres=> SHOW work_mem;
  work_mem
  ----------
   4MB
  (1 row)
  ```

  Sie können den Arbeitsspeicher auf Sitzungsebene vor Sortier-, Gruppierungs- und anderen Operationen wie folgt erhöhen.

```
SET work_mem TO '1 GB';
```

Weitere Informationen zum Arbeitsspeicher finden Sie unter Ressourcennutzung in der PostgreSQL-Dokumentation.

- Ändern Sie den Aufbewahrungszeitraum für Protokolle, sodass die Protokolle für kürzere Zeiträume gespeichert werden. Um zu erfahren wie dies geht, vgl. Datenbank-Protokolldateien von Aurora PostgreSQL.

Wenn Ihr DB-Cluster von Aurora PostgreSQL größer als 40 TB ist, verwenden Sie nicht die Instance-Klassen db.t2, db.t3 oder db.t4g. Wir empfehlen, die T-DB-Instance-Klassen nur für Entwicklungs- und Testserver oder andere Nicht-Produktionsserver zu verwenden. Weitere Informationen finden Sie unter DB-Instance-Klassenarten.

Replikation mit Amazon Aurora PostgreSQL

Im Folgenden finden Sie Informationen zur Replikation mit Amazon Aurora PostgreSQL und wie Sie die logische Replikation überwachen und verwenden.

Themen

- Verwendung von Aurora-Replicas
- Verbesserung der Leseverfügbarkeit von Aurora Replicas
- Überwachung einer Aurora PostgreSQL-Replikation
- Überblick über die logische Replikation von PostgreSQL mit Aurora
- Einrichten der logischen Replikation für Ihren DB-Cluster von Aurora PostgreSQL
- Deaktivieren der logischen Replikation
- Überwachen des Write-Through-Caches und der logischen Slots für die logische Replikation in Aurora PostgreSQL
- Beispiel: Verwenden der logischen Replikation mit DB-Clustern von Aurora PostgreSQL
- Beispiel: Logische Replikation mit Aurora PostgreSQL und AWS Database Migration Service
- Konfiguration der IAM-Authentifizierung für logische Replikationsverbindungen

Verwendung von Aurora-Replicas

Eine Aurora Replica ist ein unabhängiger Endpunkt in einem Aurora-DB-Cluster, die die beste Methode für das Skalieren von Leseoperationen und Erhöhen der Verfügbarkeit darstellen. Ein Aurora-DB-Cluster kann bis zu 15 Aurora Replicas enthalten, die sich in den Availability Zones der AWS Region des Aurora-DB-Clusters befinden.

Das DB-Cluster-Volume besteht aus mehreren Kopien der Daten für den DB-Cluster. Die Daten im Cluster-Volume werden jedoch als ein einzelnes logisches Volume der primären Writer-DB-Instance und der Aurora-Replicas im DB-Cluster dargestellt. Weitere Informationen über Aurora-Replicas finden Sie unter Aurora-Replikate.

Aurora Replicas eignen sich für das Skalieren von Lesevorgängen, da sie in Ihrem Cluster-Volume ausschließlich für Lesevorgänge bereitstehen. Schreibvorgänge werden von der Writer-DB-Instance verwaltet. Das Cluster-Volume wird von allen Instance in Ihrem Aurora PostgreSQL-DB-Cluster gemeinsam genutzt. Daher ist keine zusätzliche Arbeit erforderlich, um eine Kopie der Daten für jede Aurora Replica zu replizieren.

Mit Aurora PostgreSQL wird beim Löschen eines Aurora-Replica der Instance-Endpunkt sofort entfernt und das Aurora-Replica vom Leser-Endpunkt entfernt. Wenn Anweisungen vorhanden sind, die auf dem Aurora-Replica ausgeführt werden, das gerade gelöscht wird, besteht eine Übergangsfrist von drei Minuten. Vorhandene Anweisungen können während der Nachfrist geordnet beendet werden. Nach Ablauf der Nachfrist wird das Aurora-Replikat geschlossen und gelöscht.

Aurora PostgreSQL-DB-Cluster unterstützen Aurora Replicas in verschiedenen AWS Regionen unter Verwendung der globalen Aurora-Datenbank. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.

> **Note**
>
> Mit der Leseverfügbarkeitsfunktion müssen Sie die Aurora-Replikate im DB-Cluster ggf. manuell neu starten. Für die DB-Cluster, die vor Einführung dieser Funktion erstellt wurden, werden durch einen Neustart der Writer-DB-Instance automatisch die Aurora Replicas neu gestartet. Durch den automatischen Neustart wird ein Einstiegspunkt wiederhergestellt, der die read/write Konsistenz im gesamten DB-Cluster gewährleistet.

Verbesserung der Leseverfügbarkeit von Aurora Replicas

Aurora PostgreSQL verbessert die Leseverfügbarkeit im DB-Cluster, indem die Leseanforderungen kontinuierlich bearbeitet werden, wenn die Writer-DB-Instance neu gestartet wird oder wenn die Aurora Replica nicht mit dem Schreibverkehr Schritt halten kann.

Die Leseverfügbarkeitsfunktion ist standardmäßig in den folgenden Versionen von Aurora PostgreSQL verfügbar:

- 16.1 und alle höheren Versionen
- 15.2 und höhere 15-Versionen
- 14.7 und höhere 14-Versionen
- 13.10 und höhere 13-Versionen
- 12.14 und höhere 12-Versionen

Die Leseverfügbarkeitsfunktion wird von der globalen Aurora-Datenbank in den folgenden Versionen unterstützt:

- 16.1 und alle höheren Versionen
- 15.4 und höhere 15-Versionen
- 14.9 und höhere 14-Versionen
- 13.12 und höhere Versionen (13)
- 12.16 und höhere 12-Versionen

Um die Leseverfügbarkeitsfunktion für einen DB-Cluster zu verwenden, der vor der Einführung der Funktion in einer dieser Versionen erstellt wurde, starten Sie die Writer-Instance des DB-Clusters neu.

Wenn Sie statische Parameter Ihres DB-Clusters von Aurora PostgreSQL ändern, müssen Sie die Writer-Instance neu starten, damit die Parameteränderungen wirksam werden. Sie müssen beispielsweise die Writer-Instance neu starten, wenn Sie den Wert `shared_buffers` festlegen. Dank der Leseverfügbarkeitsfunktion von Aurora-Replikaten behält der DB-Cluster die verbesserte Verfügbarkeit bei, wodurch sich ein Neustart der Schreiber-Instance weniger stark auf ihn auswirkt. Die Reader-Instances werden nicht neu gestartet und antworten weiterhin auf die Leseanfragen. Um statische Parameteränderungen anzuwenden, starten Sie jede einzelne Reader-Instance neu.

Eine Aurora Replica eines DB-Clusters von Aurora PostgreSQL kann nach Replikationsfehlern wie Writer-Neustarts, Failover, langsamer Replikation und Netzwerkproblemen wiederhergestellt werden, indem sie nach einer erneuten Verbindung mit dem Writer schnell den In-Memory-Datenbankstatus wiederherstellt. Dieser Ansatz ermöglicht es Aurora-Replica-Instances, die Konsistenz mit den neuesten Speicher-Updates zu erreichen, solange die Client-Datenbank noch verfügbar ist.

Bei laufenden Transaktionen, die mit der Replikationswiederherstellung kollidieren, wird möglicherweise ein Fehler angezeigt, aber der Client kann diese Transaktionen erneut versuchen, sobald die Reader wieder mit dem Writer mithalten.

Überwachen von Aurora Replicas

Sie können die Aurora Replicas bei der Wiederherstellung nach einer Writer-Unterbrechung überwachen. Verwenden Sie die folgenden Metriken, um nach den neuesten Informationen zur Reader-Instance zu suchen und um in Bearbeitung befindliche schreibgeschützte Transaktionen zu verfolgen.

- Die `aurora_replica_status` Funktion wurde aktualisiert, sodass sie die meisten up-to-date Informationen für die Reader-Instanz zurückgibt, wenn diese noch verbunden ist. Der Zeitstempel der letzten Aktualisierung in `aurora_replica_status` ist für die Zeile, die der DB-Instance entspricht, auf der die Abfrage ausgeführt wird, immer leer. Dies bedeutet, dass die Reader-Instance über die neuesten Daten verfügt.
- Wenn das Aurora Replica die Verbindung zur Writer-Instance trennt und wieder eine Verbindung herstellt, wird das folgende Datenbankereignis ausgelöst:

  ```
  Read replica has been disconnected from the writer instance and
  reconnected.
  ```

- Wenn eine schreibgeschützte Abfrage aufgrund eines Wiederherstellungskonflikts abgebrochen wird, wird möglicherweise eine oder mehrere der folgenden Fehlermeldungen im Datenbankfehlerprotokoll angezeigt:

  ```
  Canceling statement due to conflict with recovery.

  User query may not have access to page data to replica disconnect.

  User query might have tried to access a file that no longer exists.

  When the replica reconnects, you will be able to repeat your command.
  ```

Einschränkungen

Die folgenden Einschränkungen gelten für Aurora-Replikate mit der Leseverfügbarkeitsfunktion:

- Aurora-Replikate des sekundären DB-Clusters können neu gestartet werden, wenn die Daten während der Replikationswiederherstellung nicht von der Schreiber-Instance gestreamt werden können.
- Aurora Replicas unterstützen keine Online-Replikationswiederherstellung, wenn eine solche Wiederherstellung bereits läuft und neu gestartet wird.
- Aurora Replicas werden neu gestartet, wenn sich Ihre DB-Instance dem Transaktions-ID-Wraparound nähert. Weitere Informationen zum Transaktions-ID-Wraparound finden Sie unter Verhindern von Transaktions-ID-Wraparound-Fehlern.
- Aurora Replicas können unter bestimmten Umständen neu gestartet werden, wenn der Replikationsprozess blockiert ist.

Überwachung einer Aurora PostgreSQL-Replikation

Skalieren von Lesevorgängen und hohe Verfügbarkeit hängen von der minimalen Verzögerungszeit ab. Sie können überwachen, wie weit eine Aurora Replica der Writer-DB-Instance Ihres Aurora PostgreSQL-DB-Clusters hinterherhinkt, indem Sie die Amazon-Metrik überwachen. CloudWatch `ReplicaLag` Da Aurora Replicas aus demselben Cluster-Volume wie die Writer-DB-Instance lesen, hat die `ReplicaLag`-Metrik für einen Aurora PostgreSQL-DB-Cluster eine andere Bedeutung. Die `ReplicaLag`-Metrik für eine Aurora Replica ermittelt die Verzögerung für den Seiten-Cache der Aurora Replica im Vergleich zu der der Writer-DB-Instance.

Weitere Informationen zur Überwachung von RDS-Instances und CloudWatch -Metriken finden Sie unter. Überwachung von Metriken in einem Amazon-Aurora-Cluster

Überblick über die logische Replikation von PostgreSQL mit Aurora

Durch die Verwendung der logischen Replikationsfunktion von PostgreSQL mit Ihrem DB-Cluster von Aurora PostgreSQL können Sie einzelne Tabellen statt der gesamten Datenbank-Instance replizieren und synchronisieren. Für die logische Replikation wird ein Veröffentlichungs- und Abonnementmodell verwendet, um Änderungen aus einer Quelle an einen oder mehrere Empfänger zu replizieren. Dazu werden Änderungsdatensätze aus dem Write-Ahead-Protokoll (WAL) von PostgreSQL verwendet. Die Quelle oder der Herausgeber sendet WAL-Daten für die angegebenen Tabellen an einen oder mehrere Empfänger (Abonnenten). Dadurch werden die Änderungen repliziert und die

Tabelle eines Abonnenten wird mit der Tabelle des Herausgebers synchronisiert. Die Änderungen des Herausgebers werden anhand einer Veröffentlichung identifiziert. Abonnenten erhalten die Änderungen, indem sie ein Abonnement erstellen, das die Verbindung mit der Datenbank des Herausgebers und den entsprechenden Veröffentlichungen definiert. Ein Replikationsslot ist der Mechanismus, der in diesem Schema verwendet wird, um den Fortschritt eines Abonnements zu verfolgen.

Bei DB-Clustern von Aurora PostgreSQL werden die WAL-Datensätze im Aurora-Speicher abgelegt. Der DB-Cluster von Aurora PostgreSQL, der in einem logischen Replikationsszenario als Herausgeber fungiert, liest die WAL-Daten aus dem Aurora-Speicher, dekodiert sie und sendet sie an den Abonnenten, sodass die Änderungen auf die Tabelle in dieser Instance angewendet werden können. Der Herausgeber verwendet einen logischen Decoder, um die Daten für die Verwendung durch Abonnenten zu dekodieren. Standardmäßig verwenden DB-Cluster von Aurora PostgreSQL beim Senden von Daten das native `pgoutput`-Plugin von PostgreSQL. Es sind auch andere logische Decoder verfügbar. Zum Beispiel unterstützt Aurora PostgreSQL auch das `wal2json`-Plugin, das WAL-Daten in JSON konvertiert.

Ab Aurora PostgreSQL Version 14.5, 13.8, 12.12 und 11.17 erweitert Aurora PostgreSQL den logischen Replikationsprozess von PostgreSQL um einen Write-Through-Cache, um die Leistung zu verbessern. Die WAL-Transaktionsprotokolle werden lokal in einem Puffer zwischengespeichert, um die Menge an Festplatten-I/O zu reduzieren, d. h. es wird während der logischen Dekodierung aus dem Aurora-Speicher gelesen. Der Write-Through-Cache wird standardmäßig verwendet, wenn Sie die logische Replikation für Ihren DB-Cluster von Aurora PostgreSQL verwenden. Aurora bietet mehrere Funktionen, mit denen Sie den Cache verwalten können. Weitere Informationen finden Sie unter Überwachen des Write-Through-Caches für die logische Replikation in Aurora PostgreSQL.

Die logische Replikation wird von allen derzeit verfügbaren Aurora-PostgreSQL-Versionen unterstützt. Weitere Informationen finden Sie unter Aktualisierungen von Amazon Aurora PostgreSQL im Abschnitt Versionshinweise für Aurora PostgreSQL.

Die logische Replikation wird von Babelfish für Aurora PostgreSQL in den folgenden Versionen unterstützt:

- 15.7 und höhere Versionen
- 16.3 und höhere Versionen

> **Note**
>
> Zusätzlich zu der in PostgreSQL 10 eingeführten nativen logischen Replikationsfunktion von PostgreSQL unterstützt Aurora PostgreSQL auch die `pglogical`-Erweiterung. Weitere Informationen finden Sie unter Verwenden von pglogical, um Daten zwischen Instances zu synchronisieren.

Weitere Informationen über die logische Replikation in PostgreSQL finden Sie unter Logical Replication (Logische Replikation) und Logical Decoding Concepts (Logische Dekodierungskonzepte) in der PostgreSQL-Dokumentation.

> **Note**
>
> PostgreSQL 16 hat Unterstützung für die logische Dekodierung von Read Replicas hinzugefügt. Diese Funktion wird in Aurora PostgreSQL nicht unterstützt.

Einrichten der logischen Replikation für Ihren DB-Cluster von Aurora PostgreSQL

Für das Einrichten der logischen Replikation sind `rds_superuser`-Berechtigungen erforderlich. Ihr DB-Cluster von Aurora PostgreSQL muss für die Verwendung einer benutzerdefinierten DB-Cluster-Parametergruppe konfiguriert sein, damit Sie die erforderlichen Parameter wie im folgenden Verfahren beschrieben festlegen können. Weitere Informationen finden Sie unter DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster.

So richten Sie die logische Replikation von PostgreSQL für einen DB-Cluster von Aurora PostgreSQL ein

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Ihren DB-Cluster von Aurora PostgreSQL im Navigationsbereich aus.
3. Öffnen Sie die Registerkarte Configuration (Konfiguration). Suchen Sie in den Instance-Details den Link Parametergruppe mit der DB-Cluster-Parametergruppe als Typ.
4. Wählen Sie den Link aus, um die benutzerdefinierten Parameter zu öffnen, die Ihrem DB-Cluster von Aurora PostgreSQL zugeordnet sind.

5. Geben Sie in das Suchfeld Parameters (Parameter) `rds` ein, um den `rds.logical_replication`-Parameter zu finden. Der Standardwert für diesen Parameter ist `0`, was bedeutet, dass er standardmäßig deaktiviert ist.
6. Wählen Sie Edit parameters (Parameter bearbeiten) aus, um auf die Eigenschaftswerte zuzugreifen. Wählen Sie dann 1 aus der Auswahl aus, um die Funktion zu aktivieren. Abhängig von Ihrer erwarteten Auslastung müssen Sie möglicherweise auch die Einstellungen für die folgenden Parameter ändern. In vielen Fällen sind die Standardwerte jedoch ausreichend.

   - `max_replication_slots`: Legen Sie diesen Parameter auf einen Wert fest, der mindestens der geplanten Gesamtzahl der Veröffentlichungen und Abonnements der logischen Replikation entspricht. Wenn Sie verwenden AWS DMS, sollte dieser Parameter mindestens Ihren geplanten Change-Data-Capture-Aufgaben aus dem Cluster sowie Veröffentlichungen und Abonnements für logische Replikation entsprechen.
   - `max_wal_sendersund max_logical_replication_workers` — Stellen Sie diese Parameter auf einen Wert ein, der mindestens der Anzahl der logischen Replikationsslots entspricht, die Sie aktiv sein möchten, oder der Anzahl der aktiven AWS DMS Aufgaben für die Datenänderungserfassung. Wenn Sie einen logischen Replikationsslot inaktiv lassen, wird die Bereinigung durch Entfernen überholter Tupel aus Tabellen verhindert. Wir empfehlen daher, Replikationsslots zu überwachen und inaktive Slots nach Bedarf zu entfernen.
   - `max_worker_processes`: Legen Sie diesen Parameter auf einen Wert fest, der mindestens der Summe der Werte `max_logical_replication_workers`, `autovacuum_max_workers` und `max_parallel_workers` entspricht. Bei kleinen DB-Instance-Klassen können sich Hintergrund-Workerprozesse auf Anwendungs-Workloads auswirken. Überwachen Sie daher die Leistung Ihrer Datenbank, wenn Sie `max_worker_processes` höher als den Standardwert festlegen. (Der Standardwert ist das Ergebnis von `GREATEST(${DBInstanceVCPU*2},8}`, was bedeutet, dass dies standardmäßig entweder 8 oder doppelt so hoch ist wie das CPU-Äquivalent der DB-Instance-Klasse, je nachdem, welcher Wert größer ist).

   > **Note**
   >
   > Sie können die Parameterwerte in einer benutzerdefinierten DB-Parametergruppe ändern. Die Parameterwerte in einer Standard-DB-Parametergruppe können nicht geändert werden.

7. Wählen Sie Änderungen speichern aus.

8. Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu, damit Ihre Änderungen wirksam werden. Wählen Sie in der Amazon-RDS-Konsole die primäre DB-Instance des Clusters und dann im Menü Actions (Aktionen) die Option Reboot (Neustart) aus.
9. Wenn die Instance verfügbar ist, können Sie wie folgt überprüfen, ob die logische Replikation aktiviert ist.

    a. Verwenden Sie `psql`, um eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL herzustellen.

    ```
    psql --host=your-db-cluster-instance-1.aws-region.rds.amazonaws.com --port=5432
     --username=postgres --password --dbname=labdb
    ```

    b. Stellen Sie mithilfe des folgenden Befehls sicher, dass die logische Replikation aktiviert wurde.

    ```
    labdb=> SHOW rds.logical_replication;
     rds.logical_replication
    -------------------------
     on
    (1 row)
    ```

    c. Überprüfen Sie, dass `wal_level` auf `logical` festgelegt ist.

    ```
    labdb=> SHOW wal_level;
      wal_level
    -----------
     logical
    (1 row)
    ```

Ein Beispiel für die Verwendung der logischen Replikation, um eine Datenbanktabelle mit Änderungen aus einem Quell-DB-Cluster von Aurora PostgreSQL zu synchronisieren, finden Sie unter Beispiel: Verwenden der logischen Replikation mit DB-Clustern von Aurora PostgreSQL.

Deaktivieren der logischen Replikation

Nachdem Sie Ihre Replikationsaufgaben abgeschlossen haben, sollten Sie den Replikationsprozess beenden, Replikationsslots löschen und die logische Replikation deaktivieren. Bevor Sie Slots löschen, vergewissern Sie sich, dass diese nicht mehr benötigt werden. Aktive Replikationsslots können nicht gelöscht werden.

So deaktivieren Sie die logische Replikation

1. Löschen Sie alle Replikations-Slots.

   Wenn Sie alle Replikationsslots löschen möchten, stellen Sie eine Verbindung zum Herausgeber her und führen Sie den folgenden SQL-Befehl aus.

   ```
   SELECT pg_drop_replication_slot(slot_name)
     FROM pg_replication_slots
    WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
   ```

   Die Replikationsslots dürfen nicht aktiv sein, wenn Sie diesen Befehl ausführen.
2. Ändern Sie die dem Herausgeber zugeordnete benutzerdefinierte DB-Cluster-Parametergruppe, wie in Einrichten der logischen Replikation für Ihren DB-Cluster von Aurora PostgreSQL beschrieben. Legen Sie den `rds.logical_replication`-Parameter jedoch auf 0 fest.

   Weitere Informationen zu benutzerdefinierten DB-Parametergruppen finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.
3. Starten Sie den Herausgeber-DB-Cluster von Aurora PostgreSQL neu, damit die Änderung des `rds.logical_replication`-Parameters wirksam wird.

Überwachen des Write-Through-Caches und der logischen Slots für die logische Replikation in Aurora PostgreSQL

Überwachen Sie den Write-Through-Cache für die logische Replikation und verwalten Sie die logischen Slots, um die Leistung Ihres DB-Clusters in Aurora PostgreSQL zu verbessern. Im Folgenden finden Sie weitere Informationen zum Write-Through-Cache und zu den logischen Slots.

Themen

- Überwachen des Write-Through-Caches für die logische Replikation in Aurora PostgreSQL
- Verwalten logischer Slots für Aurora PostgreSQL

Überwachen des Write-Through-Caches für die logische Replikation in Aurora PostgreSQL

Standardmäßig verwenden die Aurora-PostgreSQL-Versionen 14.5, 13.8, 12.12 und 11.17 und höher einen Write-Through-Cache, um die Leistung für die logische Replikation zu verbessern.

Ohne den Write-Through-Cache verwendet Aurora PostgreSQL die Aurora-Speicherschicht bei der Implementierung des nativen logischen PostgreSQL-Replikationsprozesses. Dazu schreibt es WAL-Daten in den Speicher und liest die Daten dann wieder aus dem Speicher, um sie zu dekodieren und an ihre Ziele (Abonnenten) zu senden (zu replizieren). Dies kann zu Engpässen bei der logischen Replikation für DB-Cluster von Aurora PostgreSQL führen.

Der Write-Through-Cache minimiert die Abhängigkeit von der Aurora-Speicherschicht. Statt fortlaufender Schreib- und Lesezugriffe auf diese Schicht nutzt Aurora PostgreSQL einen Puffer, der den logischen WAL-Stream für die Replikation zwischenspeichert und so Festplattenzugriffe reduziert. Dieser Puffer ist der native PostgreSQL-Cache, der in der logischen Replikation verwendet wird. Er wird in den DB-Cluster-Parametern von Aurora PostgreSQL als `rds.logical_wal_cache` identifiziert.

Wenn Sie die logische Replikation mit Ihrem DB-Cluster von Aurora PostgreSQL verwenden (für die Versionen, die den Write-Through-Cache unterstützen), können Sie die Cache-Trefferrate überwachen, um festzustellen, wie gut sie für Ihren Anwendungsfall funktioniert. Stellen Sie dazu mit `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her und verwenden Sie dann die Aurora-Funktion `aurora_stat_logical_wal_cache`, wie im folgenden Beispiel gezeigt.

```
SELECT * FROM aurora_stat_logical_wal_cache();
```

Die Funktion gibt beispielsweise die folgende Ausgabe zurück.

```
name       | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
-----------+------------+-----------+------------+-----------+----------+--------------
test_slot1 | 79183      | 24        | 0          | 24        | 100.00%  | 2022-08-05
 17:39...
test_slot2 |            | 1         | 0          |  1        | 100.00%  | 2022-08-05
 17:34...
(2 rows)
```

Die `last_reset_timestamp`-Werte wurden aus Gründen der Lesbarkeit gekürzt. Weitere Informationen zu dieser Funktion finden Sie unter aurora_stat_logical_wal_cache.

Aurora PostgreSQL bietet die beiden folgenden Funktionen zur Überwachung des Write-Through-Cache.

- Die `aurora_stat_logical_wal_cache`-Funktion – Die Referenzdokumentation finden Sie unter aurora_stat_logical_wal_cache.
- Die `aurora_stat_reset_wal_cache`-Funktion – Die Referenzdokumentation finden Sie unter aurora_stat_reset_wal_cache.

Wenn Sie feststellen, dass die automatisch angepasste WAL-Cache-Größe für Ihre Workloads nicht ausreicht, können Sie den Wert von `rds.logical_wal_cache` manuell ändern. Berücksichtigen Sie dabei Folgendes:

- Wenn der Parameter `rds.logical_replication` deaktiviert ist, wird `rds.logical_wal_cache` auf Null (0) gesetzt.
- Wenn der Parameter `rds.logical_replication` aktiviert ist, hat `rds.logical_wal_cache` einen Standardwert von 16 MB.
- Der Parameter `rds.logical_wal_cache` ist statisch und erfordert einen Neustart der Datenbank-Instance, damit Änderungen wirksam werden. Dieser Parameter ist in Form von 8-KB-Blöcken definiert. Beachten Sie, dass jeder positive Wert unter 32 KB als 32 KB behandelt wird. Weitere Informationen über `wal_buffers` finden Sie unter Write-Ahead-Protokoll in der PostgreSQL-Dokumentation.

Verwalten logischer Slots für Aurora PostgreSQL

Streaming-Aktivitäten werden in der `pg_replication_origin_status`-Ansicht erfasst. Wenn Sie den Inhalt dieser Ansicht anzeigen möchten, können Sie die `pg_show_replication_origin_status()`-Funktion verwenden, wie im Folgenden gezeigt:

```
SELECT * FROM pg_show_replication_origin_status();
```

Sie können eine Liste Ihrer logischen Slots mit der folgenden SQL-Abfrage abrufen.

```
SELECT * FROM pg_replication_slots;
```

Zum Löschen eines logischen Slots können Sie den `pg_drop_replication_slot` mit dem Namen des Slots verwenden, wie im folgenden Befehl gezeigt.

```
SELECT pg_drop_replication_slot('test_slot');
```

Beispiel: Verwenden der logischen Replikation mit DB-Clustern von Aurora PostgreSQL

Das folgende Verfahren zeigt Ihnen, wie Sie die logische Replikation zwischen zwei DB-Clustern von Aurora PostgreSQL starten. Sowohl der Herausgeber als auch der Abonnent müssen für die logische Replikation konfiguriert sein, wie unter Einrichten der logischen Replikation für Ihren DB-Cluster von Aurora PostgreSQL beschrieben.

Der DB-Cluster von Aurora PostgreSQL, der der designierte Herausgeber ist, muss ebenfalls Zugriff auf den Replikationsslot gewähren. Ändern Sie dazu die Sicherheitsgruppe, die der Virtual Public Cloud (VPC) des DB-Clusters von Aurora PostgreSQL zugeordnet ist, basierend auf dem Amazon-VPC-Service. Erlauben Sie eingehenden Zugriff, indem Sie die Sicherheitsgruppe, die der VPC des Abonnenten zugeordnet ist, zur Sicherheitsgruppe des Herausgebers hinzufügen. Weitere Informationen finden Sie unter Kontrollieren des Datenverkehrs zu Ressourcen mithilfe von Sicherheitsgruppen im Benutzerhandbuch von Amazon VPC.

Nachdem diese vorbereitenden Schritte abgeschlossen sind, können Sie die PostgreSQL-Befehle `CREATE PUBLICATION` auf dem Herausgeber und `CREATE SUBSCRIPTION` auf dem Abonnenten verwenden, wie im folgenden Verfahren beschrieben.

So starten Sie den logischen Replikationsprozess zwischen zwei DB-Clustern von Aurora PostgreSQL

Bei diesen Schritten wird davon ausgegangen, dass Ihre DB-Cluster von Aurora PostgreSQL über eine Writer-Instance mit einer Datenbank verfügen, in der die Beispieltabellen erstellt werden können.

1. Auf dem Herausgeber-DB-Cluster von Aurora PostgreSQL

   a. Erstellen Sie mit der folgenden SQL-Anweisung eine Tabelle.

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. Fügen Sie mit der folgenden SQL-Anweisung Daten in die Herausgebertabelle ein.

   ```
   INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
   ```

   c. Stellen Sie sicher, dass Daten in der Tabelle vorhanden sind, indem Sie die folgende SQL-Anweisung verwenden.

```
SELECT count(*) FROM LogicalReplicationTest;
```

d. Erstellen Sie eine Veröffentlichung für diese Tabelle, indem Sie die `CREATE PUBLICATION`-Anweisung wie folgt verwenden.

```
CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
```

2. Auf dem Abonnenten-DB-Cluster von Aurora PostgreSQL

a. Erstellen Sie wie folgt dieselbe `LogicalReplicationTest`-Tabelle auf dem Abonnenten, die Sie auf dem Herausgeber erstellt haben.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

b. Stellen Sie sicher, dass diese Tabelle leer ist.

```
SELECT count(*) FROM LogicalReplicationTest;
```

c. Erstellen Sie ein Abonnement, um die Änderungen vom Herausgeber zu erhalten. Sie müssen die folgenden Details über den Herausgeber-DB-Cluster von Aurora PostgreSQL verwenden.

- host – Die Writer-DB-Instance des Herausgeber-DB-Clusters von Aurora PostgreSQL.
- port – Der Port, den die Writer-DB-Instance überwacht. Der Standardwert für PostgreSQL lautet 5432.
- dbname – Der Name der Datenbank.

```
CREATE SUBSCRIPTION testsub CONNECTION
   'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
 password=password'
   PUBLICATION testpub;
```

> **Note**
>
> Geben Sie aus Sicherheitsgründen ein anderes Passwort als hier angegeben an.

Nachdem das Abonnement erstellt wurde, wird für den Herausgeber ein Slot für die logische Replikation erstellt.

d. Um für dieses Beispiel zu überprüfen, dass das Anfangsdatum für den Abonnenten repliziert wird, verwenden Sie die folgende SQL-Anweisung für die Abonnenten-Datenbank.

```
SELECT count(*) FROM LogicalReplicationTest;
```

Alle weiteren Änderungen am Herausgeber werden beim Abonnenten repliziert.

Durch die logische Replikation wird die Leistung beeinträchtigt. Es wird empfohlen, die logische Replikation zu deaktivieren, nachdem Ihre Replikationsaufgaben abgeschlossen sind.

Beispiel: Logische Replikation mit Aurora PostgreSQL und AWS Database Migration Service

Sie können das AWS Database Migration Service (AWS DMS) verwenden, um eine Datenbank oder einen Teil einer Datenbank zu replizieren. Wird verwendet AWS DMS , um Ihre Daten von einer Aurora PostgreSQL-Datenbank in eine andere Open Source- oder kommerzielle Datenbank zu migrieren. Weitere Informationen AWS DMS dazu finden Sie im AWS Database Migration Service Benutzerhandbuch.

Das folgende Beispiel zeigt, wie die logische Replikation von einer Aurora PostgreSQL-Datenbank als Herausgeber eingerichtet und dann AWS DMS für die Migration verwendet wird. In diesem Beispiel werden der gleiche Herausgeber und Abonnent verwendet, die unter erstellt wurde Beispiel: Verwenden der logischen Replikation mit DB-Clustern von Aurora PostgreSQL.

Um die logische Replikation mit einzurichten AWS DMS, benötigen Sie Details zu Ihrem Herausgeber und Abonnenten von Amazon RDS. Insbesondere benötigen Sie Details zur Writer-DB-Instance des Herausgebers und zur DB-Instance des Abonnenten.

Fordern Sie die folgenden Informationen für die Writer-DB-Instance des Herausgebers an:

- Die ID der Virtual Private Cloud (VPC)
- Die Subnetzgruppe
- Die Availability Zone (AZ)
- Die VPC-Sicherheitsgruppe

- Die ID der DB-Instance

Fordern Sie die folgenden Informationen für die DB-Instance des Herausgebers an:

- Die ID der DB-Instance
- Die Quell-Engine

Zur Verwendung AWS DMS für die logische Replikation mit Aurora PostgreSQL

1. Bereiten Sie die Herausgeberdatenbank für die Arbeit vor. AWS DMS

   Hierzu müssen Sie bei PostgreSQL-Datenbanken der Version 10.x und höher AWS DMS - Wrapper-Funktionen auf die Datenbank des Herausgebers anwenden. Einzelheiten hierzu und weitere Schritte finden Sie in den Anweisungen unter Verwenden von PostgreSQL-Version 10.x und höher als Quelle für AWS DMS im AWS Database Migration Service -Benuterhandbuch.
2. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die AWS DMS Konsole unterhttps://console.aws.amazon.com/dms/v2. Wählen Sie oben rechts dieselbe AWS Region aus, in der sich der Herausgeber und der Abonnent befinden.
3. Erstellen Sie eine AWS DMS Replikationsinstanz.

   Wählen Sie Werte, die mit denen für die Writer-DB-Instance des Herausgebers identisch sind. Dazu gehören die folgenden Einstellungen:

   - Wählen Sie unter VPC dasselbe VPC wie für die Writer-DB-Instance aus.
   - Wählen Sie für Replikations-Subnetzgruppe eine Subnetzgruppe mit denselben Werten wie die Writer-DB-Instance aus. Erstellen Sie bei Bedarf eine Neue.
   - Wählen Sie unter Availability zone (Availability Zone) dieselbe Zone wie für die Writer-DB-Instance aus.
   - Wählen Sie unter VPC Security Group (VPC-Sicherheitsgruppe) dieselbe Gruppe wie für die Writer-DB-Instance aus.
4. Erstellen Sie einen AWS DMS Endpunkt für die Quelle.

   Geben Sie unter Verwendung der folgenden Einstellungen den Herausgeber als Quellendpunkt an:

   - Wählen Sie unter Endpoint type (Endpunkttyp) die Option Source endpoint (Quellendpunkt) aus.

- Wählen Sie Select RDS DB Instance (RDS-DB-Instance auswählen).
- Wählen Sie unter RDS Instance (RDS-Instance) die DB-Kennung der Writer-DB-Instance des Herausgebers aus.
- Wählen Sie unter Source engine (Quellen-Engine) die Option postgres aus.

5. Erstellen Sie einen AWS DMS Endpunkt für das Ziel.

   Geben Sie den Abonnenten unter Verwendung der folgenden Einstellungen als Zielendpunkt an:

   - Wählen Sie unter Endpoint type (Endpunkttyp) die Option Target endpoint (Zielendpunkt) aus.
   - Wählen Sie Select RDS DB Instance (RDS-DB-Instance auswählen).
   - Wählen Sie unter RDS Instance (RDS-Instance) die DB-Kennung der Abonnenten-DB-Instance aus.
   - Wählen Sie einen Wert für Source engine (Quell-Engine) aus. Beispiel: Wenn der Abonnent eine RDS-PostgreSQL-Datenbank ist, wählen Sie postgres. Wenn der Subscriber eine Aurora PostgreSQL-Datenbank ist, wählen Sie aurora-postgresql aus.

6. Erstellen Sie eine AWS DMS Datenbankmigrationsaufgabe.

   Sie geben mit der Datenbankmigrationsaufgabe an, welche Datenbanktabelle migriert werden sollen, um Daten mithilfe des Zielschemas zuzuordnen und um neue Tabellen für die Zieldatenbank zu erstellen. Verwenden Sie zumindest die folgenden Einstellungen für Task configuration (Aufgabenkonfiguration):

   - Wählen Sie unter Replication instance (Replikations-Instance) die Replikations-Instance aus, die Sie in einem früheren Schritt erstellt haben.
   - Wählen Sie unter Source database endpoint (Quelldatenbank-Endpunkt) die Herausgeberquelle aus, die Sie in einem früheren Schritt erstellt haben.
   - Wählen Sie unter Target database endpoint (Zieldatenbank-Endpunkt) das Abonnentenziel aus, das Sie in einem früheren Schritt erstellt haben.

   Die übrigen Aufgabendetails sind von Ihrem Migrationsprojekt abhängig. Weitere Informationen zur Angabe aller Details für DMS-Aufgaben finden Sie im AWS Database Migration Service Benutzerhandbuch unter Arbeiten mit AWS DMS-Aufgaben.

Nachdem die Aufgabe AWS DMS erstellt wurde, beginnt sie mit der Migration der Daten vom Herausgeber zum Abonnenten.

Konfiguration der IAM-Authentifizierung für logische Replikationsverbindungen

Ab Aurora PostgreSQL Version 11 und höher können Sie die AWS Identity and Access Management (IAM) -Authentifizierung für Replikationsverbindungen verwenden. Diese Funktion erhöht die Sicherheit, da Sie den Datenbankzugriff mithilfe von IAM-Rollen anstelle von Passwörtern verwalten können. Sie funktioniert auf Clusterebene und folgt demselben Sicherheitsmodell wie die Standard-IAM-Authentifizierung.

Die IAM-Authentifizierung für Replikationsverbindungen ist eine optionale Funktion. Um sie zu aktivieren, setzen Sie den `rds.iam_auth_for_replication` Parameter 1 in Ihrer DB-Cluster-Parametergruppe auf. Da es sich um einen dynamischen Parameter handelt, muss Ihr DB-Cluster nicht neu gestartet werden, sodass Sie die IAM-Authentifizierung für bestehende Workloads ohne Ausfallzeiten nutzen können. Bevor Sie diese Funktion aktivieren können, müssen Sie die unten Voraussetzungen aufgeführten Voraussetzungen erfüllen.

Voraussetzungen

Um die IAM-Authentifizierung für Replikationsverbindungen verwenden zu können, müssen Sie alle der folgenden Anforderungen erfüllen:

- Ihr Aurora PostgreSQL-DB-Cluster muss Version 11 oder höher sein.
- Auf Ihrem Publisher-Aurora-PostgreSQL-DB-Cluster:
  - Aktivieren Sie die IAM-Datenbankauthentifizierung.

    Weitere Informationen finden Sie unter Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung.
  - Aktivieren Sie die logische Replikation, indem Sie den `rds.logical_replication` Parameter auf 1 setzen.

    Weitere Informationen finden Sie unter Einrichten der logischen Replikation für Ihren DB-Cluster von Aurora PostgreSQL.

  Bei der logischen Replikation ist der Herausgeber der Aurora PostgreSQL-Quell-DB-Cluster, der Daten an Abonnentencluster sendet. Weitere Informationen finden Sie unter Überblick über die logische PostgreSQL-Replikation mit Aurora.

> **Note**
>
> Sowohl die IAM-Authentifizierung als auch die logische Replikation müssen auf Ihrem Publisher-Aurora-PostgreSQL-DB-Cluster aktiviert sein. Wenn eine der beiden Optionen nicht aktiviert ist, können Sie die IAM-Authentifizierung nicht für Replikationsverbindungen verwenden.

Aktivieren der IAM-Authentifizierung für Replikationsverbindungen

Gehen Sie wie folgt vor, um die IAM-Authentifizierung für die Replikationsverbindung zu aktivieren.

1. Stellen Sie sicher, dass Ihr Aurora PostgreSQL-DB-Cluster alle Voraussetzungen für die IAM-Authentifizierung mit Replikationsverbindungen erfüllt. Details hierzu finden Sie unter Voraussetzungen.
2. Konfigurieren Sie den `rds.iam_auth_for_replication` Parameter, indem Sie Ihre DB-Cluster-Parametergruppe ändern:
   - Stellen Sie den Parameter `rds.iam_auth_for_replication` auf 1 ein. Für diesen dynamischen Parameter ist kein Neustart erforderlich.
3. Connect zu Ihrer Datenbank her und weisen Sie Ihrem Replikationsbenutzer die erforderlichen Rollen zu:

   Die folgenden SQL-Befehle gewähren die erforderlichen Rollen, um die IAM-Authentifizierung für Replikationsverbindungen zu aktivieren:

```
-- Grant IAM authentication role
GRANT rds_iam TO replication_user_name;
-- Grant replication privileges
ALTER USER replication_user_name WITH REPLICATION;
```

Nachdem Sie diese Schritte abgeschlossen haben, muss der angegebene Benutzer die IAM-Authentifizierung für Replikationsverbindungen verwenden.

> **Important**
>
> Wenn Sie die Funktion aktivieren, müssen Benutzer mit `rds_replication` Rollen `rds_iam` sowohl als auch die IAM-Authentifizierung für Replikationsverbindungen verwenden. Dies gilt

unabhängig davon, ob die Rollen dem Benutzer direkt zugewiesen oder über andere Rollen vererbt wurden.

Deaktivierung der IAM-Authentifizierung für Replikationsverbindungen

Sie können die IAM-Authentifizierung für Replikationsverbindungen mit einer der folgenden Methoden deaktivieren:

- Setzen Sie den `rds.iam_auth_for_replication` Parameter `0` in Ihrer DB-Cluster-Parametergruppe auf
- Alternativ können Sie eine dieser Funktionen in Ihrem Aurora PostgreSQL-DB-Cluster deaktivieren:
  - Deaktivieren Sie die logische Replikation, indem Sie den Parameter auf `rds.logical_replication` setzen `0`
  - Deaktivieren Sie die IAM-Authentifizierung

Wenn Sie die Funktion deaktivieren, können Replikationsverbindungen Datenbankkennwörter für die Authentifizierung verwenden, sofern sie konfiguriert sind.

> **Note**
>
> Replikationsverbindungen für Benutzer ohne die `rds_iam` Rolle können die Kennwortauthentifizierung verwenden, auch wenn die Funktion aktiviert ist.

Einschränkungen und Überlegungen

Die folgenden Einschränkungen und Überlegungen gelten für die Verwendung der IAM-Authentifizierung für Replikationsverbindungen.

- Die IAM-Authentifizierung für Replikationsverbindungen ist nur für Aurora PostgreSQL Versionen 11 und höher verfügbar.
- Der Herausgeber muss die IAM-Authentifizierung für Replikationsverbindungen unterstützen.
- Das IAM-Authentifizierungstoken läuft standardmäßig nach 15 Minuten ab. Möglicherweise müssen Sie lang andauernde Replikationsverbindungen aktualisieren, bevor das Token abläuft.

Lokale Schreibweiterleitung in Aurora PostgreSQL

Lokale (clusterinterne) Schreibweiterleitung ermöglicht Ihren Anwendungen, Lese-/Schreibtransaktionen direkt auf einem Aurora-Replikat durchzuführen. Die Schreibbefehle werden dann zum Commit an die Writer-DB-Instance weitergeleitet. Sie können die lokale Schreibweiterleitung für Anwendungen verwenden, bei denen gelegentlich Schreibvorgänge auftreten und die Lesen-nach-Schreiben-Konsistenz erfordern. Dies ist die Fähigkeit, den letzten Schreibvorgang in einer Transaktion zu lesen.

Ohne Schreibweiterleitung müssen Ihre Anwendungen den gesamten Lese- und Schreibverkehr vollständig aufteilen und zwei Gruppen von Datenbankverbindungen aufrechterhalten, um den Datenverkehr an den richtigen Endpunkt zu senden. Lesereplikate erhalten asynchron Updates von der Writer-Instance. Da die Replikationsverzögerung zwischen Lesereplikaten unterschiedlich sein kann, ist es außerdem schwierig, eine globale Lesekonsistenz für alle Replikate zu erreichen. Die Transaktion aller Lesevorgänge, die eine Lesen-nach-Schreiben-Konsistenz erfordern, muss auf der Writer-Datenbank-Instance erfolgen. Alternativ müssen Sie eine komplexe benutzerdefinierte Anwendungslogik entwickeln, um aus Gründen der Skalierbarkeit mehrere Lesereplikate nutzen zu können und gleichzeitig die Konsistenz sicherzustellen.

Durch die Schreibweiterleitung müssen diese Transaktionen nicht aufgeteilt oder ausschließlich an die Writer-Instance gesendet werden. Sie müssen auch keine komplexe Anwendungslogik entwickeln, um Lesen-nach-Schreiben-Konsistenz zu erreichen.

Die lokale Schreibweiterleitung ist in jeder Region verfügbar, in der Aurora PostgreSQL verfügbar ist. Sie wird in folgenden Aurora-PostgreSQL-Versionen unterstützt:

- 16.4 und höhere 16-Versionen
- 15.8 und höhere 15-Versionen
- 14.13 und höhere 14-Versionen

Die lokale Schreibweiterleitung wird verwendet, um Schreibvorgänge von Replikaten innerhalb der Region weiterzuleiten. Informationen zur Weiterleitung von Schreibvorgängen aus einem globalen Replikat finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

Themen

- Einschränkungen und Überlegungen zur lokalen Schreibweiterleitung in Aurora PostgreSQL

- Konfiguration von Aurora PostgreSQL für die lokale Schreibweiterleitung
- Arbeiten mit der lokalen Schreibweiterleitung für Aurora PostgreSQL
- Überwachen der lokalen Schreibweiterleitung in Aurora PostgreSQL

Einschränkungen und Überlegungen zur lokalen Schreibweiterleitung in Aurora PostgreSQL

Für die lokale Schreibweiterleitung in Aurora PostgreSQL gelten aktuell die folgenden Einschränkungen:

- Die lokale Schreibweiterleitung wird für RDS-Proxy nicht unterstützt.
- Bestimmte Anweisungen sind nicht zulässig oder können veraltete Ergebnisse erzeugen, wenn Sie sie in Aurora PostgreSQL mit Schreibweiterleitung verwenden. Darüber hinaus werden benutzerdefinierte Funktionen und benutzerdefinierte Prozeduren nicht unterstützt. Daher ist die Einstellung `EnableLocalWriteForwarding` für DB-Cluster standardmäßig deaktiviert. Stellen Sie vor einer Aktivierung sicher, dass der Anwendungscode von keiner dieser Einschränkungen betroffen ist.
- Die folgenden Arten von SQL-Anweisungen werden bei Schreibweiterleitung nicht unterstützt:

> **Note**
>
> Diese Anweisungen können von Ihnen implizit in Ihrer Anwendung verwendet oder vom PostgreSQL-Protokoll abgeleitet werden. Beispielsweise kann die PL/SQL-Ausnahmebehandlung zur Verwendung von SAVEPOINT führen, was keine unterstützte Anweisung ist.

  - `ANALYZE`
  - `CLUSTER`
  - `COPY`
  - Cursor – Cursor werden nicht unterstützt. Stellen Sie daher sicher, dass Sie sie schließen, bevor Sie die lokale Schreibweiterleitung verwenden.
  - DDL-Anweisungen (Data Definition Language)
  - `GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL`
  - `LISTEN / NOTIFY`

- LOCK
- SAVEPOINT
- SELECT INTO
- SET CONSTRAINTS
- Sequenzaktualisierungen: nextval(), setval()
- TRUNCATE
- Zwei-Phasen-Commit-Befehle: PREPARE TRANSACTION, COMMIT PREPARED, ROLLBACK PREPARED
- Benutzerdefinierte Funktionen und benutzerdefinierte Prozeduren.
- VACUUM

Sie können die folgenden SQL-Anweisungen mit Schreibweiterleitung verwenden:

- Eine DML-Anweisung kann aus mehreren Teilen bestehen, z. B. einer INSERT ... SELECT-Anweisung oder einer DELETE ... WHERE-Anweisung. In diesem Fall wird die gesamte Anweisung an die Writer-DB-Instance weitergeleitet und dort ausgeführt.
- Data Manipulation Language (DM)-Anweisungen wie INSERT, wieDELETE und UPDATE.
- EXPLAIN-Anweisungen mit den Anweisungen in dieser Liste.
- PREPARE- und EXECUTE-Anweisungen.
- SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }-Anweisungen.

Konfiguration von Aurora PostgreSQL für die lokale Schreibweiterleitung

Mithilfe der Informationen in den folgenden Abschnitten können Sie die lokale Schreibweiterleitung für Ihren Amazon-Aurora-PostgreSQL-DB-Cluster aktivieren, Konsistenzstufen konfigurieren und Transaktionen mit Schreibweiterleitung verwalten.

Aktivieren der lokalen Schreibweiterleitung

Standardmäßig ist die lokale Schreibweiterleitung für Aurora-PostgreSQL-DB-Cluster nicht aktiviert. Sie aktivieren die lokale Schreibweiterleitung auf Cluster-Ebene, nicht auf Instance-Ebene.

Konsole

Wählen Sie in der AWS-Managementkonsole das Kontrollkästchen Lokale Schreibweiterleitung aktivieren unter Read-Replica-Schreibweiterleitung aktivieren aus, wenn Sie einen DB-Cluster erstellen oder ändern.

AWS CLI

Wenn Sie die lokale Schreibweiterleitung über die AWS CLI aktivieren möchten, verwenden Sie die Option `--enable-local-write-forwarding`. Diese Option funktioniert, wenn Sie über den Befehl `create-db-cluster` einen neuen DB-Cluster erstellen. Sie funktioniert auch, wenn Sie einen vorhandenen DB-Cluster über den Befehl `modify-db-cluster` ändern. Sie können die lokale Schreibweiterleitung deaktivieren, indem Sie die Option `--no-enable-local-write-forwarding` mit denselben CLI-Befehlen verwenden.

Im folgenden Beispiel wird ein Aurora-PostgreSQL-DB-Cluster mit aktivierter lokaler Schreibweiterleitung erstellt.

```
aws rds create-db-cluster \
--db-cluster-identifier write-forwarding-test-cluster \
--enable-local-write-forwarding \
--engine aurora-postgresql \
--engine-version 16.4 \
--master-username myuser \
--master-user-password mypassword \
--backup-retention 1
```

Anschließend erstellen Sie Writer- und Reader-DB-Instances, sodass Sie die Schreibweiterleitung verwenden können. Weitere Informationen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

RDS-API

Um die lokale Schreibweiterleitung über die Amazon-RDS-API zu aktivieren, legen Sie den Parameter `EnableLocalWriteForwarding` auf `true` fest. Dieser Parameter funktioniert, wenn Sie über die Operation `CreateDBCluster` einen neuen DB-Cluster erstellen. Sie funktioniert auch, wenn Sie einen vorhandenen DB-Cluster über die Operation `ModifyDBCluster` ändern. Sie können die lokale Schreibweiterleitung deaktivieren, indem Sie den Parameter `EnableLocalWriteForwarding` auf `false` festlegen.

Aktivieren der lokalen Schreibweiterleitung für Datenbanksitzungen

Der Parameter `apg_write_forward.consistency_mode` ist ein DB-Parameter und ein DB-Cluster-Parameter, der die Schreibweiterleitung ermöglicht. Sie können `SESSION`, `EVENTUAL`, `GLOBAL` oder `OFF` als Lesekonsistenzstufe angeben. Weitere Informationen über Konsistenzebenen finden Sie unter Konsistenz und Isolation für die lokale Schreibweiterleitung in Aurora PostgreSQL.

Für diesen Parameter gelten die folgenden Regeln:

- Der Standardwert ist `SESSION`.
- Die lokale Schreibweiterleitung ist nur verfügbar, wenn `apg_write_forward.consistency_mode` auf `EVENTUAL`, `SESSION` oder `GLOBAL` festgelegt ist. Dieser Parameter ist nur in Reader-Instances von DB-Clustern relevant, für die die lokale Schreibweiterleitung aktiviert ist.
- Wenn Sie den Wert auf `OFF` festlegen, wird die lokale Schreibweiterleitung in der Sitzung deaktiviert.

Konsistenz und Isolation für die lokale Schreibweiterleitung in Aurora PostgreSQL

Sie können den Grad der Lesekonsistenz in einem Lesereplikat steuern. Sie können die Lesekonsistenzstufe anpassen, um sicherzustellen, dass alle aus Ihrer Sitzung weitergeleiteten Schreiboperationen im Lesereplikat vor nachfolgenden Abfragen angezeigt werden. Sie können diese Einstellung auch verwenden, um sicherzustellen, dass bei Abfragen des Lesereplikats stets die aktuellen Updates der Writer-DB-Instance angezeigt werden. Dies gilt auch für diejenigen, die von anderen Sitzungen oder anderen Clustern übermittelt wurden. Um diese Art von Verhalten für Ihre Anwendung anzugeben, wählen Sie den entsprechenden Wert für den Sitzungsebenen-Parameter `apg_write_forward.consistency_mode`. Der Parameter `apg_write_forward.consistency_mode` wirkt sich nur auf Lesereplikate aus, in denen die lokale Schreibweiterleitung aktiviert ist.

> **Note**
>
> Für den Parameter `apg_write_forward.consistency_mode` können Sie die Werte `SESSION`, `EVENTUAL`, `GLOBAL` oder `OFF` angeben. Standardmäßig ist der Wert auf `SESSION` festgelegt. Wenn Sie den Wert auf `OFF` festlegen, wird die Schreibweiterleitung deaktiviert.

Wenn Sie das Konsistenzniveau erhöhen, verbringt Ihre Anwendung mehr Zeit mit dem Warten auf die Propagierung von Änderungen zu Lesereplikaten. Sie können das Verhältnis zwischen geringerer Latenz und der Gewährleistung festlegen, dass vor der Ausführung Ihrer Abfragen Änderungen an anderen Speicherorten vollständig verfügbar sind.

Bei jeder verfügbaren Konsistenzmodus-Einstellung hat dies folgende Auswirkungen:

- `SESSION` – Für eine Sitzung auf einem Lesereplikat, das die lokale Schreibweiterleitung verwendet, werden die Ergebnisse aller in dieser Sitzung ausgeführten Änderungen angezeigt. Die Änderungen werden unabhängig davon angezeigt, ob die Transaktion übergeben wurde. Wenn notwendig, wartet die Abfrage auf die Replikation der Ergebnisse weitergeleiteter Schreiboperationen zur aktuellen Reader-DB-Instance. Sie wartet nicht auf aktualisierte Ergebnisse aus Schreiboperationen, die in anderen Sitzungen innerhalb des aktuellen DB-Clusters ausgeführt werden.
- `EVENTUAL` – Einer Sitzung auf einem Lesereplikat, das die lokale Schreibweiterleitung verwendet, werden möglicherweise Daten angezeigt, die aufgrund von Replikationsverzögerungen leicht veraltet sind. Ergebnisse von Schreibvorgängen in derselben Sitzung sind erst sichtbar, wenn die Schreiboperation auf der Writer-DB-Instance ausgeführt und auf dem Lesereplikat repliziert wurde. Die Abfrage wartet nicht, bis die aktualisierten Ergebnisse verfügbar sind. Daher könnte sie die älteren Daten oder die aktualisierten Daten abrufen, abhängig vom Zeitpunkt der Anweisungen und der Größe der Replikationsverzögerung.
- `GLOBAL` – In einer Sitzung auf einem Lesereplikat werden Änderungen angezeigt, die von dieser Sitzung vorgenommen wurden. Außerdem werden ihr alle übergebenen Änderungen sowohl aus der Schreib-DB-Instance als auch aus anderen Lesereplikaten angezeigt. Jede Abfrage kann für einen bestimmten Zeitraum warten, abhängig von der Sitzungsverzögerung. Die Abfrage wird fortgesetzt, wenn das Lesereplikat mit allen übergebenen Daten aus der Writer-DB-Instance mit Stand zu dem Zeitpunkt, an dem die Abfrage gestartet wurde, aktualisiert ist.

  > **Note**
  >
  > Der globale Konsistenzmodus wirkt sich auf die Latenz von Abfragen aus, die innerhalb einer Sitzung ausgeführt werden. Es gibt eine Wartezeit, auch wenn die Sitzung keine Schreibabfragen gesendet hat.

- `OFF` – Die lokale Schreibweiterleitung ist deaktiviert.

In Sitzungen, die Schreibweiterleitung verwenden, können Sie nur die Isolationsstufen `REPEATABLE READ` und `READ COMMITTED` verwenden. Die `SERIALIZABLE`-Isolationsstufe wird jedoch nicht unterstützt.

Weitere Informationen zu allen mit der Schreibweiterleitung verbundenen Parametern finden Sie unter Standardparametereinstellungen für die Schreibweiterleitung.

Transaktionszugriffsmodi mit Schreibweiterleitung

Wenn der Transaktionszugriffsmodus schreibgeschützt ist, wird die lokale Schreibweiterleitung nicht verwendet. Sie können den Zugriffsmodus nur dann auf Lese- und Schreibzugriff festlegen, wenn Sie mit einem DB-Cluster und einer Sitzung verbunden sind, in der die lokale Schreibweiterleitung aktiviert ist.

Weitere Informationen zu den Transaktionszugriffsmodi finden Sie unter SET TRANSACTION.

Arbeiten mit der lokalen Schreibweiterleitung für Aurora PostgreSQL

Mit den Informationen in den folgenden Abschnitten können Sie überprüfen, ob für einen Datenbank-Cluster die lokale Schreibweiterleitung aktiviert ist. Außerdem finden Sie Überlegungen zur Kompatibilität und Informationen zu den konfigurierbaren Parametern und der Einrichtung der Authentifizierung. So können Sie die lokale Schreibweiterleitungsfunktion in Aurora PostgreSQL effektiv nutzen.

> **Note**
>
> Wenn eine Writer-Instance in einem Cluster, der die lokale Schreibweiterleitung verwendet, neu gestartet wird, werden alle aktiven, weitergeleiteten Transaktionen und Abfragen auf Reader-Instances, die die lokale Schreibweiterleitung verwenden, automatisch geschlossen. Nachdem die Writer-Instance wieder verfügbar ist, können Sie diese Transaktionen erneut versuchen.

Überprüfen, ob die lokale Schreibweiterleitung für einen DB-Cluster aktiviert ist

Wenn Sie bestimmen möchten, ob Sie die lokale Schreibweiterleitung in einem DB-Cluster verwenden können, stellen Sie sicher, dass das `LocalWriteForwardingStatus`-Attribut für den Cluster auf `enabled` festgelegt ist.

In der AWS-Managementkonsole sehen Sie auf der Registerkarte Konfiguration der Detailseite des Clusters den StatusAktiviert für Lokale Lesereplikat-Schreibweiterleitung.

Um den Status der Einstellung für die lokale Schreibweiterleitung für alle Cluster anzuzeigen, führen Sie den folgenden AWS CLI-Befehl aus.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
{
"LocalWriteForwardingStatus": "enabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-1"
},
{
"LocalWriteForwardingStatus": "disabled",
"DBClusterIdentifier": "write-forwarding-test-cluster-2"
},
{
"LocalWriteForwardingStatus": "requested",
"DBClusterIdentifier": "test-global-cluster-2"
},
{
"LocalWriteForwardingStatus": "null",
"DBClusterIdentifier": "aurora-postgresql-v2-cluster"
}
]
```

Für einen DB-Cluster sind folgende Werte `LocalWriteForwardingStatus` zulässig:

- `disabled` – Die lokale Schreibweiterleitung ist deaktiviert.
- `disabling` – Die lokale Schreibweiterleitung wird gerade deaktiviert.
- `enabled` – Die lokale Schreibweiterleitung ist aktiviert.
- `enabling` – Die lokale Schreibweiterleitung wird gerade aktiviert.
- `null` – Die lokale Schreibweiterleitung ist für diesen DB-Cluster nicht verfügbar.
- `requested` – Die lokale Schreibweiterleitung wurde angefordert, ist aber noch nicht aktiv.

Standardparametereinstellungen für die Schreibweiterleitung

Die Aurora-Cluster-Parametergruppen enthalten Einstellungen für die lokale Schreibweiterleitung. Da es sich um Cluster-Parameter handelt, haben alle DB-Instances in allen Clustern die gleichen Werte für diese Variablen. Details zu diesen Parametern sind in der folgenden Tabelle zusammengefasst, mit Nutzungshinweisen unterhalb der Tabelle.

Parameter	Scope	Typ	Standardwert	Zulässige Werte
`apg_write_forward.connect_timeout`	Sitzung	Sekunden	30	0 – 2147483647
`apg_write_forward.consistency_mode`	Sitzung	enum	Sitzung	`SESSION`, `EVENTUAL`, `GLOBAL`, und `OFF`
`apg_write_forward.idle_in_transaction_session_timeout`	Sitzung	Millisekunden	86400000	0 – 2147483647
`apg_write_forward.idle_session_timeout`	Sitzung	Millisekunden	300000	0 – 2147483647
`apg_write_forward.max_forwarding_connections_percent`	Global	int	25	1-100

Der `apg_write_forward.max_forwarding_connections_percent`-Parameter definiert die Obergrenze der Datenbankverbindungsslots, die zum Umgang mit von Readern weitergeleiteten Abfragen verwendet werden können. Er wird als Prozentsatz der Einstellung `max_connections` für die Writer-DB-Instance ausgedrückt. Wenn beispielsweise `max_connections` `800` und `apg_write_forward.max_forwarding_connections_percent` `10` ist, dann lässt der Writer maximal 80 weitergeleitete Sitzungen gleichzeitig zu. Diese Verbindungen stammen aus demselben,

von der Einstellung `max_connections` verwalteten Verbindungspool. Diese Einstellung gilt nur für die Writer-DB-Instance, wenn für den Cluster die lokale Schreibweiterleitung aktiviert ist.

Verwenden Sie die folgenden Einstellungen, um Anforderungen für die lokale Schreibweiterleitung zu steuern:

- `apg_write_forward.consistency_mode` – Ein Parameter auf Sitzungsebene, der den Grad der Lesekonsistenz auf dem Lesereplikat steuert. Gültige Werte sind `SESSION`, `EVENTUAL`, `GLOBAL` oder `OFF`. Standardmäßig ist der Wert auf `SESSION` festgelegt. Wenn Sie den Wert auf `OFF` festlegen, wird die lokale Schreibweiterleitung in der Sitzung deaktiviert. Weitere Informationen über Konsistenzebenen finden Sie unter Konsistenz und Isolation für die lokale Schreibweiterleitung in Aurora PostgreSQL. Dieser Parameter ist nur in Reader-Instances relevant, für die die lokale Schreibweiterleitung aktiviert ist.
- `apg_write_forward.connect_timeout` – Die maximale Anzahl von Sekunden, die das Lesereplikat beim Herstellen einer Verbindung mit der Writer-DB-Instance wartet, bevor der Versuch beendet wird. Ein Wert von `0` bedeutet, dass auf unbestimmte Zeit gewartet werden soll.
- `apg_write_forward.idle_in_transaction_session_timeout` – Die Anzahl der Millisekunden, die die Writer-DB-Instance auf Aktivitäten bei einer Verbindung wartet, die von einem Lesereplikat mit einer offenen Transaktion weitergeleitet wurde, bevor die Verbindung beendet wird. Wenn die Sitzung über diesen Zeitraum hinaus inaktiv ist, bricht Aurora die Sitzung ab. Durch einen Wert des Typs `0` wird der Timout deaktiviert.
- `apg_write_forward.idle_session_timeout` – Die Anzahl der Millisekunden, die die Writer-DB-Instance auf Aktivitäten bei einer Verbindung wartet, die von einem Lesereplikat weitergeleitet wurde, bevor sie beendet wird. Wenn die Sitzung über diesen Zeitraum hinaus inaktiv bleibt, beendet Aurora die Sitzung. Durch einen Wert des Typs `0` wird der Timout deaktiviert.

rdswriteforwarduser

`rdswriteforwarduser` ist ein Benutzer, der verwendet wird, um eine Verbindung zwischen dem Lesereplikat und der Writer-DB-Instance herzustellen.

> **Note**
>
> `rdswriteforwarduser` erbt die CONNECT-Berechtigungen für Kundendatenbanken über die PUBLIC-Rolle. Wenn die Berechtigungen für die PUBLIC-Rolle entzogen werden, müssen

Sie CONNECT-Berechtigungen für die Datenbanken gewähren, an die Sie Schreibvorgänge weiterleiten müssen.

Überwachen der lokalen Schreibweiterleitung in Aurora PostgreSQL

Mithilfe der Informationen in den folgenden Abschnitten können Sie die lokale Schreibweiterleitung in Aurora-PostgreSQL-Clustern überwachen, einschließlich relevanter CloudWatch-Metriken und Warteereignisse, um die Leistung zu verfolgen und potenzielle Probleme zu identifizieren.

Amazon-CloudWatch-Metriken und Aurora-PostgreSQL-Statusvariablen für die Schreibweiterleitung

Die folgenden Amazon CloudWatch-Metriken gelten für die Writer-DB-Instances, wenn Sie die Schreibweiterleitung auf einem oder mehreren Lesereplikaten verwenden.

CloudWatch-Metrik	Einheiten und Beschreibung
`AuroraLocal ForwardingWriterDM LThroughput`	Anzahl pro Sekunde Anzahl der weitergel eiteten DML-Anweisungen, die pro Sekunde von dieser Writer-DB-Instance verarbeitet werden.
`AuroraLocalForwardingWriter OpenSessions`	Anzahl Anzahl der offenen Sitzungen auf dieser Writer-DB-Instance, die weitergeleitete Anfragen verarbeiten.
`AuroraLocal ForwardingWriterTo talSessions`	Anzahl Anzahl der weitergeleiteten Sitzungen auf dieser Writer-DB-Instance.

Die folgenden CloudWatch-Metriken gelten für jedes Lesereplikat. Diese Metriken werden auf jeder Reader-DB-Instance im DB-Cluster mit aktivierter lokaler Schreibweiterleitung gemessen.

CloudWatch-Metrik	Einheit und Beschreibung
`AuroraForwardingReplicaCommitThroughput`	Anzahl pro Sekunde Anzahl der Commits in Sitzungen, die von diesem Replikat pro Sekunde weitergeleitet werden.
`AuroraForwardingReplicaDMLLatency`	Millisekunden Durchschnittliche Reaktionszeit weitergeleiteter DML-Anweisungen in Millisekunden auf Replikaten.
`AuroraForwardingReplicaDMLThroughput`	Anzahl pro Sekunde Anzahl der weitergeleiteten DML-Anweisungen, die in diesem Replikat pro Sekunde verarbeitet werden.
`AuroraForwardingReplicaErrorSessionsLimit`	Anzahl Anzahl der Sitzungen, die von der Writer-DB-Instance zurückgewiesen wurden, weil das Limit für die maximale Zahl von Verbindungen oder die maximalen Verbindungen für die Schreibweiterleitung erreicht wurde.
`AuroraForwardingReplicaOpenSessions`	Anzahl Die Anzahl der Sitzungen, die die lokale Schreibweiterleitung für eine Replikat-Instance verwenden.
`AuroraForwardingReplicaReadWaitLatency`	Millisekunden Durchschnittliche Wartezeit in Millisekunden, die das Replikat darauf wartet, mit der LSN der Writer-DB-Instance konsistent zu sein. Das Ausmaß, in dem die Reader-DB-Instance vor der Verarbeitung einer Abfrage wartet, ist von der Einstellung `apg_write_forward.consistency_mode` abhängig. Weitere Informationen zu dieser Einstellung finden Sie unter the section called “Konfigurationsparameter für die Schreibweiterleitung in Aurora PostgreSQL”.

Warteereignisse für die lokale Schreibweiterleitung in Aurora PostgreSQL

Amazon Aurora generiert die folgenden Warteereignisse, wenn Sie die Schreibweiterleitung mit Aurora PostgreSQL verwenden.

Themen

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

Das `IPC:AuroraWriteForwardConnect`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem Lesereplikat darauf wartet, dass eine Verbindung zur Writer-DB-Instance geöffnet wird.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Dieses Ereignis nimmt zu, wenn die Anzahl der Verbindungsversuche vom Lesereplikat zum Writer-Knoten zunimmt.

Aktionen

Reduzieren Sie die Anzahl gleichzeitiger Verbindungen von einem Lesereplikat zum Writer-Knoten.

IPC:AuroraWriteForwardConsistencyPoint

Das `IPC:AuroraWriteForwardConsistencyPoint`-Ereignis beschreibt, wie lange eine Abfrage von einem Knoten auf dem Lesereplikat darauf wartet, dass die Ergebnisse weitergeleiteter Schreiboperationen in die aktuelle Region repliziert werden. Dieses Ereignis wird nur generiert, wenn der Parameter `apg_write_forward.consistency_mode` auf Sitzungsebene auf einen der folgenden Werte gesetzt ist:

- `SESSION` – Abfragen auf einem Lesereplikat warten auf die Ergebnisse aller in dieser Sitzung vorgenommenen Änderungen.

- `GLOBAL` – Abfragen auf einem Lesereplikat warten auf die Ergebnisse der von dieser Sitzung vorgenommenen Änderungen sowie auf alle festgeschriebenen Änderungen sowohl aus der Writer-DB-Instance als auch aus dem Lesereplikat.

Informationen zum Einstellen des Parameters `apg_write_forward.consistency_mode` finden Sie unter the section called “Konfigurationsparameter für die Schreibweiterleitung in Aurora PostgreSQL”.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten gehören:

- Erhöhte Replikatverzögerung, gemessen anhand der Amazon CloudWatch-Metrik `ReplicaLag`. Weitere Informationen zu dieser Metrik finden Sie unter Überwachung einer Aurora PostgreSQL-Replikation.
- Erhöhte Auslastung der Writer-DB-Instance oder des Lesereplikats.

Aktionen

Ändern Sie Ihren Konsistenzmodus je nach den Anforderungen Ihrer Anwendung.

IPC:AuroraWriteForwardExecute

Das `IPC:AuroraWriteForwardExecute`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem Lesereplikat darauf wartet, dass eine weitergeleitete Abfrage abgeschlossen wird und Ergebnisse vom Writer-Knoten des DB-Clusters abgerufen werden.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Abrufen einer großen Anzahl von Zeilen aus dem Writer-Knoten.
- Eine erhöhte Netzwerklatenz zwischen dem Writer-Knoten und dem Lesereplikat erhöht die Zeit, die das Lesereplikat benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Auslastung des Lesereplikats kann die Übertragung der Abfrageanforderung vom Lesereplikat zum Writer-Knoten verzögern.
- Eine erhöhte Auslastung des Writer-Knotens kann die Übertragung der Daten vom Writer-Knoten zum Lesereplikat verzögern.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

- Optimieren Sie Abfragen, um nur die erforderlichen Daten abzurufen.
- Optimieren Sie DML-Operationen (Data Manipulation Language), um nur die erforderlichen Daten zu ändern.
- Wenn das Lesereplikat oder der Writer-Knoten durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, dies auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

Das `IPC:AuroraWriteForwardGetGlobalConsistencyPoint`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem Lesereplikat, der den Konsistenzmodus GLOBAL verwendet, darauf wartet, den globalen Konsistenzpunkt vom Writer-Knoten zu erhalten, bevor er eine Abfrage ausführt.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem Lesereplikat und dem Writer-Knoten verlängert die Zeit, die das Lesereplikat benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Auslastung des Lesereplikats kann die Übertragung der Abfrageanforderung vom Lesereplikat zum Writer-Knoten verzögern.
- Eine erhöhte Auslastung des Writer-Knotens kann die Übertragung der Daten vom Writer-Knoten zum Lesereplikat verzögern.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

- Ändern Sie Ihren Konsistenzmodus je nach den Anforderungen Ihrer Anwendung.
- Wenn das Lesereplikat oder der Writer-Knoten durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, dies auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardXactAbort

Das `IPC:AuroraWriteForwardXactAbort`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem Lesereplikat auf das Ergebnis einer Remote-Bereinigungsabfrage wartet. Bereinigungsabfragen werden ausgegeben, um den Prozess nach dem Abbruch einer per Schreibweiterleitung weitergeleiteten Transaktion wieder in den entsprechenden Zustand zu versetzen. Amazon Aurora führt diese entweder aus, weil ein Fehler gefunden wurde oder weil ein Benutzer einen expliziten `ABORT`-Befehl ausgegeben oder eine laufende Abfrage abgebrochen hat.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem Lesereplikat und dem Writer-Knoten verlängert die Zeit, die das Lesereplikat benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Auslastung des Lesereplikats kann die Übertragung der Abfrageanforderung für die Bereinigung vom Lesereplikat zum Writer-Knoten verzögern.
- Eine erhöhte Auslastung des Writer-Knotens kann die Übertragung der Daten vom Writer-Knoten zum Lesereplikat verzögern.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

- Untersuchen Sie die Ursache für die abgebrochene Transaktion.
- Wenn das Lesereplikat oder die Writer-DB-Instance durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, dies auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardXactCommit

Das `IPC:AuroraWriteForwardXactCommit`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem Lesereplikat auf das Ergebnis eines weitergeleiteten Commit-Transaktionsbefehls wartet.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem Lesereplikat und dem Writer-Knoten verlängert die Zeit, die das Lesereplikat benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Auslastung des Lesereplikats kann die Übertragung der Abfrageanforderung vom Lesereplikat zum Writer-Knoten verzögern.
- Eine erhöhte Auslastung des Writer-Knotens kann die Übertragung der Daten vom Writer-Knoten zum Lesereplikat verzögern.

Aktionen

Wenn das Lesereplikat oder der Writer-Knoten durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, dies auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardXactStart

Das `IPC:AuroraWriteForwardXactStart`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem Lesereplikat auf das Ergebnis eines weitergeleiteten Start-Transaktionsbefehls wartet.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem Lesereplikat und dem Writer-Knoten verlängert die Zeit, die das Lesereplikat benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Auslastung des Lesereplikats kann die Übertragung der Abfrageanforderung vom Lesereplikat zum Writer-Knoten verzögern.
- Eine erhöhte Auslastung des Writer-Knotens kann die Übertragung der Daten vom Writer-Knoten zum Lesereplikat verzögern.

Aktionen

Wenn das Lesereplikat oder der Writer-Knoten durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, dies auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

Verwenden von Aurora PostgreSQL als Wissensdatenbank für Amazon Bedrock

Sie können einen DB-Cluster von Aurora PostgreSQL als Wissensdatenbank für Amazon Bedrock verwenden. Weitere Informationen finden Sie unter Erstellen eines Vektorspeichers in Amazon Aurora. Eine Wissensdatenbank nimmt automatisch unstrukturierte Textdaten, die in einem Amazon-S3-Bucket gespeichert sind, konvertiert sie in Textblöcke und Vektoren und speichert sie in einer PostgreSQL-Datenbank. Mit den Anwendungen mit generativer KI können Sie Agenten für Amazon Bedrock verwenden, um die in der Wissensdatenbank gespeicherten Daten abzufragen und die Ergebnisse dieser Abfragen zu verwenden, um die Antworten zu erweitern, die von grundlegenden Modellen bereitgestellt werden. Dieser Workflow wird Retrieval Augmented Generation (RAG) genannt. Weitere Informationen zu RAG finden Sie unter Retrieval Augmented Generation (RAG).

Ausführliche Informationen zur Verwendung von Aurora PostgreSQL zum Erstellen von Anwendungen mit generativer KI mithilfe von RAG finden Sie in diesem Blogbeitrag.

Themen

- Voraussetzungen
- Vorbereiten von Aurora PostgreSQL zur Verwendung als Wissensdatenbank für Amazon Bedrock
- Erstellen einer Wissensdatenbank in der Bedrock-Konsole
- Schnellerstellen einer Aurora-PostgreSQL-Wissensdatenbank für Amazon Bedrock

Voraussetzungen

Machen Sie sich mit den folgenden Voraussetzungen vertraut, um den Aurora-PostgreSQL-Cluster als Wissensdatenbank für Amazon Bedrock zu verwenden. Auf oberster Ebene müssen Sie die folgenden Services für die Verwendung mit Bedrock konfigurieren:

- Der DB-Cluster von Amazon Aurora PostgreSQL wurde in einer der folgenden Versionen erstellt:
  - 16.1 und alle höheren Versionen
  - 15.4 und höhere Versionen
  - 14.9 und höhere Versionen
  - 13.12 und höhere Versionen
  - 12.16 und höhere Versionen

> **Note**
>
> Sie müssen die `pgvector`-Erweiterung in Ihrer Zieldatenbank aktivieren und Version 0.5.0 oder höher verwenden. Weitere Informationen finden Sie unter pgvector v0.5.0 mit HNSW-Indizierung.

- RDS-Daten-API
- Ein in AWS Secrets Manager verwalteter Benutzer. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager.

Vorbereiten von Aurora PostgreSQL zur Verwendung als Wissensdatenbank für Amazon Bedrock

Folgen Sie den Schritten in den folgenden Abschnitten, um Aurora PostgreSQL für die Verwendung als Wissensdatenbank für Amazon Bedrock vorzubereiten.

Erstellen und Konfigurieren von Aurora PostgreSQL

Zur Konfiguration von Amazon Bedrock mit einem DB-Cluster von Aurora PostgreSQL müssen Sie zunächst einen DB-Cluster von Aurora PostgreSQL erstellen und sich die wichtigen Felder für dessen Konfiguration mit Amazon Bedrock notieren. Weitere Informationen zum Erstellen eines DB-Clusters von Aurora PostgreSQL finden Sie unter Erstellen eines DB-Clusters von Aurora PostgreSQL und Herstellen einer Verbindung.

- Aktivieren Sie die Daten-API, während Sie den DB-Cluster von Aurora PostgreSQL erstellen. Weitere Informationen zu den unterstützten Versionen finden Sie unter Verwenden der Daten-API von Amazon RDS.
- Notieren Sie sich unbedingt den Amazon-Ressourcennamen (ARN) Ihres DB-Clusters von Aurora PostgreSQL. Sie benötigen ihn, um den DB-Cluster für die Verwendung mit Amazon Bedrock zu konfigurieren. Weitere Informationen finden Sie unter Amazon-Ressourcennamen (ARNs).

Verbinden mit einer Datenbank und Installieren von pgvector

Sie können mit jedem der Verbindungsdienstprogramme eine Verbindung zu Aurora PostgreSQL herstellen. Weitere Informationen zu diesen Dienstprogrammen finden Sie unter Herstellen einer Verbindung mit einem Amazon-Aurora-PostgreSQL-DB-Cluster. Zur Ausführung der Abfragen

können Sie auch den Abfrage-Editor der RDS-Konsole verwenden. Sie benötigen einen Aurora-DB-Cluster mit aktivierter RDS-Daten-API, um den Abfrage-Editor verwenden zu können.

1. Melden Sie sich mit Ihrem Master-Benutzer bei der Datenbank an und richten Sie pgvector ein. Verwenden Sie den folgenden Befehl, wenn die Erweiterung nicht installiert ist:

   ```
   CREATE EXTENSION IF NOT EXISTS vector;
   ```

   Verwenden Sie `pgvector` 0.5.0 und höhere Versionen, die die HNSW-Indizierung unterstützen. Weitere Informationen finden Sie unter pgvector v0.5.0 mit HNSW-Indizierung.
2. Verwenden Sie den folgenden Befehl, um die installierte Version des `pg_vector` zu überprüfen:

   ```
   SELECT extversion FROM pg_extension WHERE extname='vector';
   ```

Einrichten von Datenbankobjekten und -berechtigungen

1. Erstellen Sie ein bestimmtes Schema, mit dem Bedrock die Daten abfragen kann. Verwenden Sie den folgenden Befehl, um ein Schema zu erstellen:

   ```
   CREATE SCHEMA bedrock_integration;
   ```
2. Erstellen Sie eine neue Rolle, mit der Bedrock die Datenbank abfragen kann. Verwenden Sie den folgenden Befehl, um eine neue Rolle zu erstellen:

   ```
   CREATE ROLE bedrock_user WITH PASSWORD 'password' LOGIN;
   ```

   > **Note**
   >
   > Notieren Sie sich dieses Passwort, da Sie es später brauchen, um ein Secrets-Manager-Passwort zu erstellen.

   Bei Nutzung des `psql`-Clients verwenden Sie die folgenden Befehle, um eine neue Rolle zu erstellen:

   ```
   CREATE ROLE bedrock_user LOGIN;
   \PASSWORD password;
   ```

3. Erteilen Sie die `bedrock_user`-Berechtigungen zur Verwaltung des Schemas `bedrock_integration`. Dies bietet die Möglichkeit, Tabellen oder Indizes innerhalb des Schemas zu erstellen.

```
GRANT ALL ON SCHEMA bedrock_integration to bedrock_user;
```

4. Melden Sie sich als `bedrock_user` an und erstellen Sie eine Tabelle im `bedrock_integration schema`.

```
CREATE TABLE bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding
 vector(n), chunks text, metadata json, custom_metadata jsonb);
```

Dieser Befehl erstellt die Tabelle `bedrock_kb` im Schema `bedrock_integration` mit Titan-Einbettungen.

Ersetzen Sie n im Datentyp `vector(n)` durch die entsprechende Dimension für das von Ihnen verwendete Einbettungsmodell. Verwenden Sie die folgenden Empfehlungen, um Ihre Dimensionen auszuwählen:

- Verwenden Sie für das Modell Titan v2 `vector(1024)`, `vector(512)` oder `vector (256)`. Weitere Informationen finden Sie unter Amazon Titan Embeddings Text.
- Verwenden Sie für das Modell Titan v1.2 `vector(1536)`. Weitere Informationen finden Sie unter Amazon Titan Multimodal Embeddings G1.
- Verwenden Sie für das Cohere-Embed-Modell `vector(1024)`. Weitere Informationen finden Sie unter Cohere-Embed-Modelle.
- Verwenden Sie `vector(1024)` für das Cohere Embed Multilingual v3.

Die ersten vier Spalten sind Pflichtfelder. Für die Bearbeitung von Metadaten schreibt Bedrock Daten aus Ihren Metadatendateien in die Spalte `custom_metadata`. Wir empfehlen, diese Spalte zu erstellen, wenn Sie Metadaten und Filterung verwenden möchten. Wenn Sie keine Spalte `custom_metadata` erstellen, fügen Sie einzelne Spalten für jedes Metadatenattribut in Ihrer Tabelle hinzu, bevor Sie mit der Erfassung beginnen. Weitere Informationen finden Sie unter Konfigurieren und Anpassen von Abfragen und Antwortgenerierung.

5. Gehen Sie wie folgt vor, um die erforderlichen Indizes zu erstellen, die Bedrock zur Abfrage Ihrer Daten verwendet:

- Erstellen Sie einen Index mit dem Kosinus-Operator, den Bedrock verwenden kann, um die Daten abzufragen.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops);
```

- Wir empfehlen Ihnen, den Wert von `ef_construction` für `pgvector` 0.6.0 und höhere Versionen, die die parallele Indexerstellung verwenden, auf 256 zu setzen.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops) WITH (ef_construction=256);
```

- Erstellen Sie einen Index, den Bedrock zur Abfrage der Textdaten verwenden kann.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (to_tsvector('simple',
 chunks));
```

- Wenn Sie eine Spalte für benutzerdefinierte Metadaten erstellt haben, erstellen Sie einen Index, den Bedrock zur Abfrage der Metadaten verwenden kann.

```
CREATE INDEX ON bedrock_integration.bedrock_kb USING gin (custom_metadata);
```

Erstellen eines Secrets in Secrets Manager

Mit Secrets Manager können Sie Ihre Aurora-Anmeldeinformationen speichern, sodass sie sicher an Anwendungen übertragen werden können. Wenn Sie bei der Erstellung des DB-Clusters von Aurora PostgreSQL nicht die Option AWS Secrets Manager ausgewählt haben, können Sie jetzt ein Secret erstellen. Weitere Informationen zum Erstellen eines AWS Secrets Manager-Datenbank-Secrets finden Sie unter Datenbank-Secret von AWS Secrets Manager.

Erstellen einer Wissensdatenbank in der Bedrock-Konsole

Während Sie Aurora PostgreSQL für die Verwendung als Vektorspeicher für eine Wissensdatenbank vorbereiten, müssen Sie die folgenden Informationen sammeln, die Sie der Amazon-Bedrock-Konsole zur Verfügung stellen müssen.

- ARN des DB-Clusters von Amazon Aurora – Der ARN Ihres DB-Clusters
- Secret-ARN – Der ARN des Schlüssels von AWS Secrets Manager für Ihren DB-Cluster

- Datenbankname – Der Name Ihrer Datenbank. Sie können beispielsweise die Standarddatenbank *postgres* verwenden.
- Tabellenname – Wir empfehlen Ihnen, bei der Erstellung der Tabelle einen schemaqualifizierten Namen anzugeben. Verwenden Sie dazu einen Befehl, der dem folgenden ähnelt:

```
CREATE TABLE bedrock_integration.bedrock_kb;
```

  Dieser Befehl erstellt die Tabelle `bedrock_kb` im Schema `bedrock_integration`.
- Achten Sie beim Erstellen der Tabelle darauf, sie mit den angegebenen Spalten und Datentypen zu konfigurieren. Sie können Ihre bevorzugten Spaltennamen anstelle der in der Tabelle aufgeführten verwenden. Denken Sie daran, sich die Namen zu notieren, die Sie bei der Einrichtung der Wissensdatenbank als Referenz ausgewählt haben.

Spaltenname	Datentyp	Beschreibung
id	UUID-Primärschlüssel	Enthält eindeutige Bezeichner für jeden Datensatz.
Blöcke	Text	Enthält die Rohtextblöcke aus Ihren Datenquellen.
Einbettung	Vektor	Enthält die Vektorein bettungen der Datenquellen.
Metadaten	JSON	Enthält Metadaten, die für die Quellenzuweisung und die Datenerfassung und -abfrage erforderlich sind
custom_metadata	JSONB	(Optional) Definiert die Zielspalte, in die Amazon Bedrock Metadatendetails aus Ihren Datenquellen schreibt

Mit diesen Details können Sie jetzt eine Wissensdatenbank in der Bedrock-Konsole erstellen. Ausführlichere Informationen zum Einrichten eines Vektorindex und zum Erstellen von

Wissensdatenbank-Informationen finden Sie unter Erstellen eines Vektorspeichers in Amazon Aurora und Erstellen eines Vektorspeichers in Amazon Aurora.

Nachdem Sie Aurora als Ihre Wissensdatenbank hinzugefügt haben, können Sie jetzt Ihre Datenquellen für Suchen und Abfragen aufnehmen. Weitere Informationen finden Sie unter Aufnehmen Ihrer Datenquellen in die Wissensdatenbank.

Schnellerstellen einer Aurora-PostgreSQL-Wissensdatenbank für Amazon Bedrock

Der Retrieval Augmented Generation (RAG)-Workflow von Amazon Bedrock stützt sich auf Vektordaten, die in einer Aurora-PostgreSQL-Datenbank gespeichert sind, um den Inhaltsabruf zu ermöglichen. Bisher war die Einrichtung von Aurora PostgreSQL als Vektordatenspeicher für Bedrock-Wissensdatenbanken ein mehrstufiger Prozess, der zahlreiche manuelle Aktionen auf verschiedenen Benutzeroberflächen erforderte. Dies machte es für Datenwissenschaftler und Entwickler schwierig, Aurora für ihre Bedrock-Projekte zu nutzen.

Zur Verbesserung der Benutzererfahrung hat AWS eine neue CloudFormation-basierte Schnellerstellungsoption entwickelt, die den Einrichtungsprozess vereinfacht. Mit der Aurora-Schnellerstellung können Sie jetzt mit einem einzigen Klick einen vorkonfigurierten DB-Cluster von Aurora PostgreSQL als Vektorspeicher für Ihre Amazon-Bedrock-Wissensdatenbanken bereitstellen.

Themen

- Unterstützte Regionen und Aurora-PostgreSQL-Versionen
- Grundlagen des Schnellerstellungsprozesses
- Vorteile der Verwendung der Aurora-Schnellerstellung
- Einschränkungen beim Aurora-Schnellerstellungsprozess

Unterstützte Regionen und Aurora-PostgreSQL-Versionen

Die Aurora-Schnellerstellungsoption ist in allen AWS-Regionen verfügbar, die Amazon-Bedrock-Wissensdatenbanken unterstützen. Standardmäßig wird ein DB-Cluster von Aurora PostgreSQL mit Version 15.7 erstellt. Weitere Informationen zu unterstützten Regionen finden Sie unter Unterstützte Modelle und Regionen für Amazon-Bedrock-Wissensdatenbanken.

Grundlagen des Schnellerstellungsprozesses

Der Prozess der Schnellerstellung stellt automatisch die folgenden Ressourcen bereit, um eine Datenbank von Amazon Aurora PostgreSQL als Vektordatenspeicher für Ihre Amazon-Bedrock-Wissensdatenbank einzurichten:

Einen DB-Cluster von Aurora PostgreSQL in Ihrem Konto, konfiguriert mit Standardeinstellungen

- ACUs (Aurora Capacity Units) sind auf einen Wert von 0 bis 16 eingestellt. Dadurch kann Ihr Vektorspeicher auf Null herunterskalieren, wenn er nicht verwendet wird, was Rechenkosten spart. Die ACUs können später in der Amazon-RDS-Konsole angepasst werden.
- Der HNSW-Index (Hierarchical Navigable Small World) verwendet die euklidische Entfernung als Ähnlichkeitsmaß für die in Aurora gespeicherten Bedrock-Vektoreinbettungen.
- Die DB-Instance ist eine Serverless-v2-Instance.
- Der Cluster ist mit der Standard-VPC und den Standard-Subnetzen verknüpft und die RDS-Daten-API ist aktiviert.
- Die Anmeldeinformationen des Cluster-Administrators werden von AWS Secrets Manager verwaltet.

Neben den Standardeinstellungen werden die folgenden Einstellungen für Sie eingerichtet. Während Sie den Prozess durchlaufen, werden Ihnen Bildschirme angezeigt, in denen der Workflow erklärt wird.

- Seeding des Aurora-Clusters mit den erforderlichen Datenbankobjekten:
  - Erstellen Sie die pgvector-Erweiterung, das Schema, die Rolle und die Tabellen, die für die Bedrock-Wissensdatenbank erforderlich sind.
  - Registrieren Sie einen Datenbankbenutzer mit eingeschränkten Berechtigungen, damit Bedrock mit dem Cluster interagieren kann.
- Während des gesamten Prozesses der Ressourcenbereitstellung wird ein Fortschrittsbanner angezeigt, mit dem Sie den Status der folgenden Unterereignisse verfolgen können:
  - Erstellung des Aurora-Clusters
  - Seeding des Aurora-Clusters
  - Erstellung der Wissensdatenbank

  Das Banner bleibt sichtbar, bis die Wissensdatenbank vollständig erstellt ist, auch wenn Sie die Seite verlassen und zurückkehren.

- Sie können im Fortschrittsbanner auf `View details` klicken, um den Status der einzelnen Schritte zu sehen. Weitere Informationen zu Ereignissen während der Erstellung der Wissensdatenbank erhalten Sie, wenn Sie auf dem Bildschirm zur Detailanzeige den CloudFormation-Link auswählen. Sobald der Prozess abgeschlossen ist, ist Ihre neue Bedrock-Wissensdatenbank einsatzbereit.
- Die Stack-IDs für alle Ressourcen der Schnellerstellung finden Sie in den Tags der Bedrock-Wissensdatenbank, falls Sie darauf verweisen müssen.

Eine Bedrock-Wissensdatenbank mit der Konfiguration für den neu bereitgestellten Aurora-Cluster, während der Vektorspeicher erstellt wird

Vorteile der Verwendung der Aurora-Schnellerstellung

- Der auf CloudFormation basierende Prozess der Schnellerstellung reduziert den Zeit- und Komplexitätsaufwand für die Verwendung von Aurora als Vektorspeicher erheblich.
- Aurora bietet hervorragende Leistung, Vektorskalierbarkeit und Kostenvorteile einschließlich der Möglichkeit, bei Nichtnutzung auf null Rechenkosten zu skalieren.
- Der Prozess der Schnellerstellung optimiert das Gesamterlebnis und ermöglicht es Ihnen, Ihre Bedrock-Wissensdatenbanken mit Aurora einfach zu erstellen und zu konfigurieren.
- Kunden können die CloudFormation-Vorlage als Grundlage verwenden, um die Bereitstellung mit ihren eigenen Konfigurationen anzupassen.

Einschränkungen beim Aurora-Schnellerstellungsprozess

- Mit der Aurora-Schnellerstellungsoption wird der DB-Cluster mit Standardkonfigurationen bereitgestellt. Diese Standardeinstellungen entsprechen jedoch möglicherweise nicht Ihren spezifischen Anforderungen oder Ihrem beabsichtigten Anwendungsfall. Die Schnellerstellung bietet keine Optionen zum Ändern der Konfigurationen während des Bereitstellungsprozesses. Die Konfigurationen werden automatisch festgelegt, um die Bereitstellung zu optimieren. Wenn Sie die Konfiguration des Aurora-DB-Clusters anpassen müssen, können Sie dies nach der ersten Bereitstellung durch die Schnellerstellung in der Amazon-RDS-Konsole tun.
- Der Schnellerstellungsablauf vereinfacht zwar den Einrichtungsprozess, die Erstellung des Aurora-DB-Clusters dauert jedoch immer noch etwa 10 Minuten, genau wie bei einer manuellen Bereitstellung. Dies ist auf den Zeitaufwand zurückzuführen, der für die Bereitstellung der Aurora-Infrastruktur benötigt wird.

- Die Schnellerstellungsoption ist für Experimente und das Quick Setup konzipiert. Die mit der Schnellerstellung erstellten Ressourcen sind möglicherweise nicht für die Produktion geeignet und Sie können sie nicht direkt zu einer Produktionsumgebung in Ihrer VPC migrieren.

Integrieren von Amazon Aurora PostgreSQL in anderen AWS-Services

Amazon Aurora ist auch in anderen AWS-Services integriert, damit Sie Ihren Aurora PostgreSQL-DB-Cluster erweitern können, um zusätzliche Funktionen in der AWS Cloud zu verwenden. Ihr Aurora-PostgreSQL-DB-Cluster kann AWS-Services verwenden, um Folgendes durchzuführen:

- Performance Ihrer Aurora PostgreSQL-DB-Instances mit Amazon-RDS-Performance-Insights schnell ermitteln, anzeigen und beurteilen. Performance Insights lässt sich auf vorhandene Amazon-RDS-Überwachungsfunktionen erweitern, damit Sie die Performance Ihrer Datenbank darstellen und mögliche Probleme analysieren können. Mit dem Performance-Insights-Dashboard können Sie die Datenbankauslastung visualisieren und die Auslastung nach Wartezeiten, SQL-Anweisungen, Hosts oder Benutzern filtern. Weitere Informationen zu Performance Insights finden Sie unter Überwachung mit Performance Insights auf .
- Konfigurieren Ihres Aurora-PostgreSQL-DB-Clusters, sodass Protokolldaten in Amazon CloudWatch Logs veröffentlicht werden. CloudWatch Logs bietet hoch dauerhaften Speicher für Ihre Protokolldatensätze. Mit CloudWatch Logs können Sie Echtzeitanalysen der Protokolldaten durchführen, und mit CloudWatch können Sie Alarme und Metriken erstellen. Weitere Informationen finden Sie unter Veröffentlichen von Aurora-PostgreSQL-Protokollen in Amazon CloudWatch Logs.
- Importieren Sie Daten von einem Amazon S3-Bucket zu einem Aurora PostgreSQL DB-Cluster oder exportieren Sie Daten von einem Aurora PostgreSQL DB-Cluster zu einem Amazon S3-Bucket. Weitere Informationen erhalten Sie unter Importieren von Amazon S3 in einen Aurora-PostgreSQL-DB-Cluster und Exportieren von Daten aus einem/einer Aurora PostgreSQL-DB-Cluster zu Amazon S3.
- Hinzufügen von auf Machine Learning basierenden Vorhersagen zu Datenbankanwendungen unter Verwendung der SQL-Sprache. Aurora-Machine-Learning ist eine hoch optimierte Integration zwischen der Aurora-Datenbank und den AWS Machine Learning (ML)-Services SageMaker AI und Amazon Comprehend. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Machine Learning mit Aurora PostgreSQL.

- Rufen Sie AWS Lambda Funktionen von einem Aurora PostgreSQL DB-Cluster auf. Verwenden Sie dazu die `aws_lambda` PostgreSQL-Erweiterung, die mit Aurora PostgreSQL bereitgestellt wird. Weitere Informationen finden Sie unter Aufrufen einer AWS Lambda Funktion aus einem Aurora PostgreSQL-DB-Cluster (PostgreSQL-DB-Instance).
- Integrieren Sie Abfragen von Amazon Redshift und Aurora PostgreSQL. Weitere Informationen finden Sie im Amazon Redshift Database Developer Guide unter Getting started with using federated queries to PostgreSQL.

Importieren von Amazon S3 in einen Aurora-PostgreSQL-DB-Cluster

Sie können Daten, die mit Amazon Simple Storage Service gespeichert wurden, in eine Tabelle auf einer Aurora PostgreSQL DB-Cluster-Instance importieren. Um dies zu tun, installieren Sie zuerst die `aws_s3`-Erweiterung von Aurora PostgreSQL . Diese Erweiterung stellt die Funktionen bereit, die Sie zum Importieren von einem Amazon S3 Bucket verwenden. Ein Bucket ist ein Amazon S3 Container für Objekte und Dateien. Die Daten können sich in einer Datei mit kommagetrennten Werten (CSV), einer Textdatei oder einer komprimierten Datei (GZIP) befinden. Im Folgenden erfahren Sie, wie Sie die Erweiterung installieren und Daten aus Amazon S3 in eine Tabelle importieren.

Um von Simple Storage Service (Amazon S3) in zu importieren, muss Ihre Datenbank PostgreSQL Version 10.7 oder höher verwenden. Aurora PostgreSQL

Wenn Sie keine Daten in Amazon S3 gespeichert haben, müssen Sie zunächst einen Bucket erstellen und die Daten speichern. Weitere Informationen finden Sie in den folgenden Themen im Benutzerhandbuch zum Amazon Simple Storage Service.

- Erstellen Sie einen Bucket
- Hinzufügen eines Objekts zu einem Bucket

Das kontoübergreifende Importieren aus Amazon S3 wird unterstützt. Weitere Informationen finden Sie unter Gewähren kontoübergreifender Berechtigungen im Benutzerhandbuch zu Amazon Simple Storage Service.

Sie können den vom Kunden verwalteten Schlüssel für die Verschlüsselung verwenden, wenn Sie Daten aus S3 importieren. Weitere Informationen finden Sie unter In AWS KMS gespeicherte KMS-Schlüssel im Benutzerhandbuch zu Amazon Simple Storage Service.

> **Note**
>
> Das Importieren von Daten aus Amazon S3 wird für Aurora Serverless v1 nicht unterstützt. Es wird für Aurora Serverless v2 unterstützt.

Themen

- Installieren der Erweiterung aws_s3
- Übersicht über den Import von Daten aus Amazon S3-Daten
- Einrichten des Zugriffs auf einen Amazon-S3-Bucket
- Importieren von Daten aus Amazon S3 in Ihren Aurora PostgreSQL DB-Cluster
- Funktionsreferenz

Installieren der Erweiterung aws_s3

Bevor Sie Amazon S3 mit Ihrem Aurora-PostgreSQL-DB-Cluster verwenden können müssen Sie die `aws_s3`-Erweiterung installieren. Diese Erweiterung bietet Funktionen zum Importieren von Daten aus einem Amazon S3. Sie bietet auch Funktionen zum Exportieren von Daten aus einer Instance eines Aurora-PostgreSQL-DB-Clusters zu einem Amazon-S3-Bucket. Weitere Informationen finden Sie unter Exportieren von Daten aus einem/einer Aurora PostgreSQL-DB-Cluster zu Amazon S3. Die Erweiterung `aws_s3` hängt von einigen Hilfsfunktionen in der Erweiterung `aws_commons` ab, die bei Bedarf automatisch installiert wird.

So installieren Sie die Erweiterung **`aws_s3`**

1. Verwenden Sie psql (oder pgAdmin), um eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL als Benutzer mit `rds_superuser`-Berechtigungen herzustellen. Wenn Sie beim Einrichten den Standardnamen beibehalten haben, stellen Sie eine Verbindung als `postgres` her.

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Führen Sie den folgenden Befehl aus, um die Erweiterung zu installieren.

   ```
   postgres=> CREATE EXTENSION aws_s3 CASCADE;
   NOTICE: installing required extension "aws_commons"
   ```

```
CREATE EXTENSION
```

3. Wenn Sie überprüfen möchten, ob die Erweiterung installiert wurde, können Sie psql-Metabefehl `\dx` verwenden.

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                 Description
-------------+---------+------------+---------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

Die Funktionen zum Importieren von Daten aus Amazon S3 und Exportieren von Daten nach Amazon S3 sind jetzt verfügbar.

Übersicht über den Import von Daten aus Amazon S3-Daten

S3-Daten in Aurora PostgreSQL importieren

Sammeln Sie zunächst die Details, die Sie der Funktion zur Verfügung stellen müssen. Dazu gehören der Name der Tabelle auf Ihrer Aurora PostgreSQL DB-Cluster Instance, sowie der Bucket-Name, der Dateipfad, der Dateityp und der AWS-Region Speicherort der Amazon S3-Daten. Weitere Informationen finden Sie unter Kopieren von Objekten im Benutzerhandbuch zu Amazon Simple Storage Service.

> **Note**
>
> Der mehrteilige Datenimport aus Amazon S3 wird derzeit nicht unterstützt.

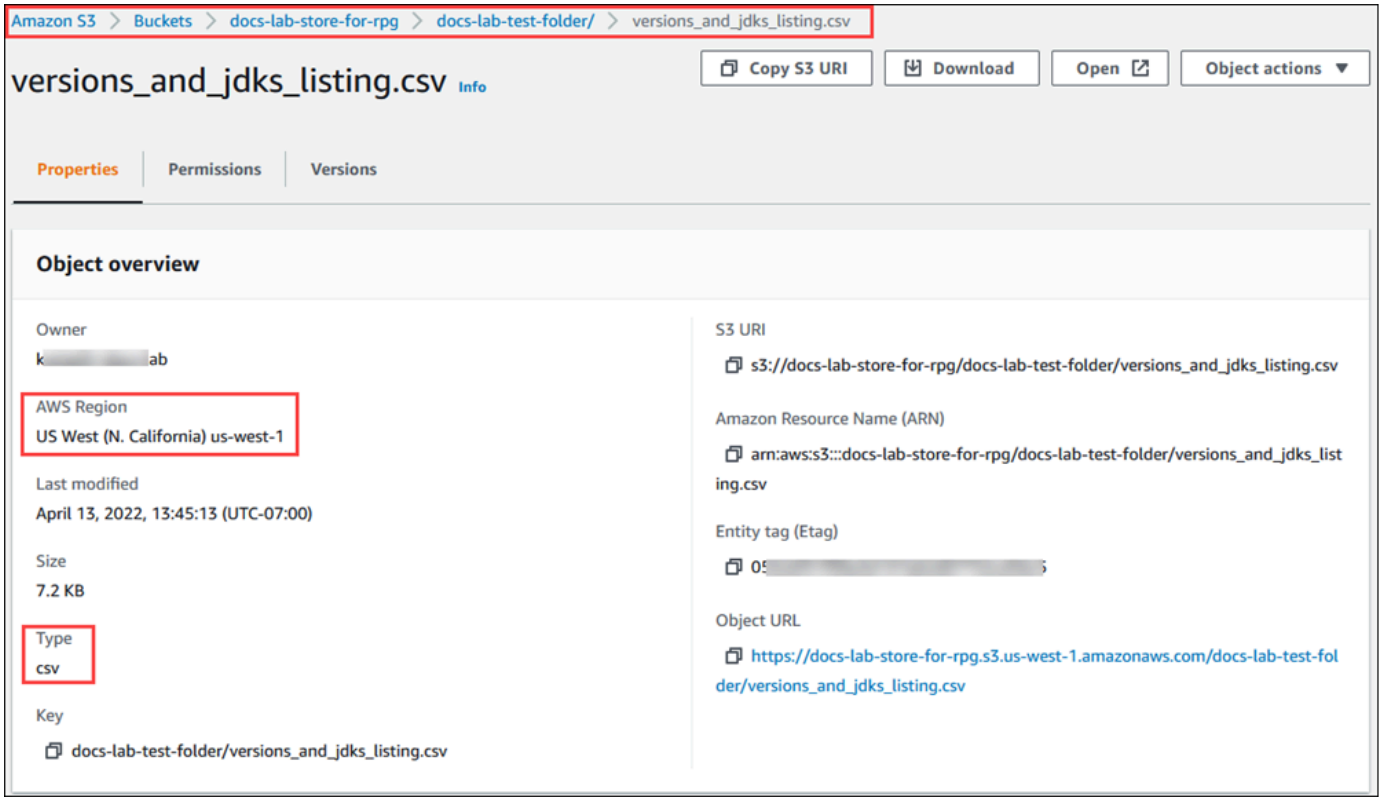
1. Ermittelt den Namen der Tabelle, in die die `aws_s3.table_import_from_s3`-Funktion die Daten importieren soll. Mit dem folgenden Befehl wird beispielsweise eine Tabelle `t1` erstellt, die in späteren Schritten verwendet werden kann.

```
postgres=> CREATE TABLE t1
    (col1 varchar(80),
    col2 varchar(80),
```

```
    col3 varchar(80));
```

2. Rufen Sie die Details zum Amazon-S3-Bucket und die zu importierenden Daten ab. Öffnen Sie dazu die Amazon S3-Konsole unter https://console.aws.amazon.com/s3/ und wählen Sie Buckets. Suchen Sie den Bucket, der Ihre Daten enthält, in der Liste. Wählen Sie den Bucket aus, öffnen Sie die Seite Objektübersicht und wählen Sie dann Properties (Eigenschaften).

   Notieren Sie sich den Bucket-Namen, den Pfad, die AWS-Region und den Dateityp. Sie benötigen den Amazon-Ressourcenname (ARN) später, um den Zugriff auf Amazon S3 über eine IAM-Rolle einzurichten. Weitere Informationen finden Sie unter Einrichten des Zugriffs auf einen Amazon-S3-Bucket. In der folgenden Abbildung sehen Sie ein Beispiel.



3. Sie können den Pfad zu den Daten im Amazon-S3-Bucket überprüfen, indem Sie den Befehl AWS CLI verwenden `aws s3 cp`. Wenn die Informationen korrekt sind, lädt dieser Befehl eine Kopie der Amazon S3-Datei herunter.

```
aws s3 cp s3://amzn-s3-demo-bucket/sample_file_path ./
```

4. Richten Sie Berechtigungen auf Ihrem Aurora-PostgreSQL-DB-Cluster ein, um den Zugriff auf die Datei im Amazon-S3-Bucket zu gestatten. Dazu verwenden Sie entweder eine AWS Identity and Access Management-(IAM)-Rolle oder Sicherheitsanmeldeinformationen. Weitere Informationen finden Sie unter Einrichten des Zugriffs auf einen Amazon-S3-Bucket.

5. Geben Sie den Pfad und andere gesammelte Amazon S3-Objektdetails (siehe Schritt 2) an die `create_s3_uri`-Funktion zum Erstellen eines Amazon S3-URI-Objekts. Weitere Informationen zu dieser Funktion finden Sie unter aws_commons.create_s3_uri. Es folgt ein Beispiel für die Erstellung dieses Objekts während einer psql-Sitzung.

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

   Im nächsten Schritt übergeben Sie dieses Objekt (`aws_commons._s3_uri_1`) an die `aws_s3.table_import_from_s3`-Funktion, um die Daten in die Tabelle zu importieren.

6. Rufen Sie die `aws_s3.table_import_from_s3`-Funktion zum Importieren der Daten aus Amazon S3 in Ihre Tabelle auf. Referenz-Informationen finden Sie unter aws_s3.table_import_from_s3. Beispiele finden Sie unter Importieren von Daten aus Amazon S3 in Ihren Aurora PostgreSQL DB-Cluster.

Einrichten des Zugriffs auf einen Amazon-S3-Bucket

Um Daten aus einer Amazon S3-Datei zu importieren, erteilen Sie dem/der Aurora PostgreSQL-DB-Cluster die Berechtigung, auf den Amazon S3-Bucket zuzugreifen, in dem sich die Datei befindet. Sie können den Zugriff auf einen Amazon S3-Bucket auf zwei Arten erlaubt, wie in den folgenden Themen beschrieben.

Themen

- Verwenden einer IAM-Rolle für den Zugriff auf einen Amazon S3-Bucket
- Verwenden von Sicherheitsanmeldeinformationen für den Zugriff auf einen Amazon S3-Bucket
- Fehlerbehebung beim Zugriff auf Amazon S3

Verwenden einer IAM-Rolle für den Zugriff auf einen Amazon S3-Bucket

Bevor Sie Daten aus einer Amazon S3-Datei laden, geben Sie Ihrer Aurora PostgreSQL DB-Cluster die Berechtigung, auf den Amazon S3-Bucket der Datei zuzugreifen. Auf diese Weise müssen Sie keine zusätzlichen Anmeldeinformationen verwalten oder im aws_s3.table_import_from_s3-Funktionsaufruf angeben.

Erstellen Sie dazu eine IAM-Richtlinie, die den Zugriff auf den Amazon S3-Bucket ermöglicht. Erstellen Sie eine IAM-Rolle und hängen Sie die Richtlinie an die Rolle an. Weisen Sie dann die IAM-Rolle Ihrer DB Cluster zu.

> **Note**
>
> Sie können einem Aurora Serverless v1-DB-Cluster keine IAM-Rolle zuordnen, daher gelten die folgenden Schritte nicht.

Einem Aurora-PostgreSQL-DB-Cluster über eine IAM-Rolle Zugriff auf Amazon S3 gewähren

1. Erstellen Sie eine IAM-Richtlinie.

   Diese Richtlinie enthält die Bucket- und Objektberechtigungen, die Ihrer Aurora PostgreSQL DB-Cluster den Zugriff auf Amazon S3 ermöglichen.

   Nehmen Sie die folgenden erforderlichen Aktionen in die Richtlinie auf, um die Übertragung von Dateien von einem Amazon S3-Bucket nach Aurora PostgreSQL zu ermöglichen:

   - `s3:GetObject`
   - `s3:ListBucket`

   Nehmen Sie die folgenden Ressourcen in die Richtlinie auf, um den Amazon S3-Bucket und Objekte im Bucket zu identifizieren. Dies zeigt das Amazon Resource Name (ARN) Format für den Zugriff auf Amazon S3 an.

   - arn:aws:s3::: *`amzn-s3-demo-bucket`*
   - arn:aws:s3::: /* *`amzn-s3-demo-bucket`*

   Weitere Informationen zum Erstellen einer IAM-Richtlinie für Aurora PostgreSQL finden Sie unter Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff. Siehe auch Tutorial: Erstellen und Anfügen Ihrer ersten vom Kunden verwalteten Richtlinie im IAM-Benutzerhandbuch.

   Der folgende AWS CLI Befehl erstellt eine IAM-Richtlinie mit diesen Optionen. `rds-s3-import-policy` Sie gewährt Zugriff auf einen Bucket namens *`amzn-s3-demo-bucket`*.

> **Note**
>
> Notieren Sie sich den Amazon-Ressourcennamen (ARN) der Richtlinie, der vom Befehl zurückgegeben wurde. Sie benötigen den ARN in einem nachfolgenden Schritt, in dem Sie die Richtlinie an eine IAM-Rolle anhängen.

Example

Für Linux, macOS oder Unix:

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

Für Windows:

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
```

```
        "Action": [
          "s3:GetObject",
          "s3:ListBucket"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket",
          "arn:aws:s3:::amzn-s3-demo-bucket/*"
        ]
      }
    ]
  }'
```

2. Erstellen Sie eine IAM-Rolle.

   Sie tun dies, damit Aurora PostgreSQL in Ihrem Namen diese IAM-Rolle übernehmen kann, um auf Ihre Amazon S3-Buckets zuzugreifen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen IAM-Benutzer im IAM-Benutzerhandbuch.

   Wir empfehlen die Verwendung von `aws:SourceArn` und `aws:SourceAccount` globaler Bedingungskontext-Schlüssel in ressourcenbasierten Richtlinien, um die Berechtigungen des Dienstes auf eine bestimmte Ressource zu beschränken. Dies ist der effektivste Weg, um sich vor dem verwirrtes Stellvertreterproblem zu schützen.

   Wenn Sie sowohl globale Kontextschlüssel nutzen und der `aws:SourceArn`-Wert enthält die Konto-ID, muss der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn`-Wert die gleiche Konto-ID verwenden, wenn er in der gleichen Richtlinienanweisung verwendet wird.

   - Verwenden von `aws:SourceArn` wenn Sie einen serviceübergreifenden Zugriff für eine einzelne Ressource wünschen.
   - Verwenden von `aws:SourceAccount` wenn Sie zulassen möchten, dass eine Ressource in diesem Konto mit der betriebsübergreifenden Verwendung verknüpft wird.

   Verwenden Sie in der Richtlinie den `aws:SourceArn` globalen Kontextschlüssel mit dem vollständigen ARN der Ressource. Das folgende Beispiel zeigt, wie Sie dazu den AWS CLI Befehl verwenden, um eine Rolle mit dem Namen `rds-s3-import-role` zu erstellen.

   Example

   Für Linux, macOS oder Unix:

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
       }
     ]
   }'
```

Für Windows:

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
```

```
      }
    ]
  }'
```

3. Fügen Sie die erstellte IAM-Richtlinie der IAM-Rolle an, die Sie erstellt haben.

   Mit dem folgenden AWS CLI Befehl wird die im vorherigen Schritt erstellte Richtlinie der Rolle `rds-s3-import-role` Replace *`your-policy-arn`* mit dem Richtlinien-ARN zugeordnet, den Sie in einem früheren Schritt notiert haben.

   Example

   Für Linux, macOS oder Unix:

   ```
   aws iam attach-role-policy \
      --policy-arn your-policy-arn \
      --role-name rds-s3-import-role
   ```

   Für Windows:

   ```
   aws iam attach-role-policy ^
      --policy-arn your-policy-arn ^
      --role-name rds-s3-import-role
   ```

4. Fügen Sie die IAM-Rolle der DB Cluster hinzu.

   Dazu verwenden Sie das AWS-Managementkonsole oder AWS CLI, wie im Folgenden beschrieben.

Konsole

So fügen Sie eine IAM-Rolle für eine PostgreSQL DB-Cluster- über die Konsole hinzu:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie den Namen der PostgreSQL DB-Cluster- aus, um ihre Details anzuzeigen.
3. Wählen Sie auf der Registerkarte Connectivity & Security im Bereich Manage IAM roles (IAM-Rollen verwalten) die Rolle aus, die unter Add IAM roles (IAM-Rollen hinzufügen) zu diesen Cluster- hinzugefügt werden soll.
4. Wählen Sie unter Feature (Funktion) die Option s3Import aus.

5. Wählen Sie Rolle hinzufügen.

AWS CLI

So fügen Sie eine IAM-Rolle für einen PostgreSQL-DB-Cluster mithilfe der CLI hinzu:

- Verwenden Sie den folgenden Befehl, um die Rolle mit dem Namen `my-db-cluster` zum PostgreSQL DB-Cluster hinzuzufügen. Ersetzen Sie *`your-role-arn`* durch den Rollen-ARN, den Sie im vorherigen Schritt notiert haben. Verwenden Sie `s3Import` für den Wert der `--feature-name`-Option.

  Example

  Für Linux, macOS oder Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Import \
   --role-arn your-role-arn   \
   --region your-region
```

  Für Windows:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Import ^
   --role-arn your-role-arn ^
   --region your-region
```

RDS-API

Um eine IAM-Rolle für eine mithilfe der Amazon RDS-API hinzuzufügen, rufen Sie den Vorgang auf. AddRoleToDBCluster

Verwenden von Sicherheitsanmeldeinformationen für den Zugriff auf einen Amazon S3-Bucket

Wenn Sie es vorziehen, können Sie Sicherheitsanmeldeinformationen verwenden, um den Zugriff auf einen Amazon S3-Bucket zu ermöglichen, anstatt den Zugriff mit einer IAM-Rolle zu ermöglichen. Dazu geben Sie die `credentials`-Parameter im aws_s3.table_import_from_s3-Funktionsaufruf an.

Der `credentials` Parameter ist eine Struktur vom Typ`aws_commons._aws_credentials_1`, die Anmeldeinformationen enthält. AWS Verwenden Sie die Funktion aws_commons.create_aws_credentials, um den Zugriffs- und den Geheimschlüssel in einer `aws_commons._aws_credentials_1`-Struktur festzulegen, wie nachfolgend dargestellt.

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

Nachdem Sie die `aws_commons._aws_credentials_1` -Struktur erstellt haben, verwenden Sie die Funktion aws_s3.table_import_from_s3 mit dem Parameter `credentials`, um die Daten zu importieren, wie nachfolgend gezeigt.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

Sie können auch den Funktionsaufruf aws_commons.create_aws_credentials in den Funktionsaufruf `aws_s3.table_import_from_s3` einbinden.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Fehlerbehebung beim Zugriff auf Amazon S3

Wenn Sie beim Versuch, Daten aus Amazon S3 zu importieren, auf Verbindungsprobleme stoßen, finden Sie im Folgenden Empfehlungen:

- Fehlerbehebung für Amazon Aurora-Identität und -Zugriff
- Fehlerbehebung bei Amazon S3 im Entwicklerhandbuch für Amazon Simple Storage Service
- Fehlerbehebung bei Amazon S3 und IAM im IAM-Benutzerhandbuch

Importieren von Daten aus Amazon S3 in Ihren Aurora PostgreSQL DB-Cluster

Sie importieren Daten aus Ihrem Amazon-S3-Bucket mithilfe der `table_import_from_s3`-Funktion der aws_s3-Erweiterung. Referenz-Informationen finden Sie unter aws_s3.table_import_from_s3.

> **Note**
>
> Die folgenden Beispiele verwenden die IAM-Rollen-Methode, um den Zugriff auf den Amazon-S3-Bucket zu ermöglichen. Daher enthalten die `aws_s3.table_import_from_s3`-Funktionsaufrufe keine Berechtigungsnachweisparameter.

Nachfolgend ist ein typisches Beispiel aufgeführt.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't1',
   '',
   '(format csv)',
   :'s3_uri'
);
```

Es werden folgende Parameter verwendet:

- `t1` – Der Name für die Tabelle in der PostgreSQL DB Cluster, in die die Daten kopiert werden.
- `''` – Eine optionale Liste mit Spalten in der Datenbanktabelle. Mithilfe dieses Parameters können Sie angeben, welche Spalten der S3-Daten in die Tabellenspalten übernommen werden sollen. Wenn keine Spalten angegeben sind, werden alle Spalten in die Tabelle kopiert. Ein Beispiel zum Verwenden einer Spaltenliste finden Sie unter Importieren einer Amazon S3-Datei, die ein benutzerdefiniertes Trennzeichen verwendet.
- `(format csv)` – PostgreSQL COPY-Argumente. Der Kopiervorgang verwendet die Argumente und das Format des PostgreSQL-Befehls COPY, um die Daten zu importieren. Zu den Optionen für das Format gehören kommagetrennte Werte (CSV), Text und Binärwerte. Der Standard ist Text.
- `s3_uri` – Eine Struktur mit den Informationen zum Identifizieren der Amazon S3-Datei. Ein Beispiel für die Verwendung der Funktion aws_commons.create_s3_uri zum Erstellen einer `s3_uri`-Struktur finden Sie unter Übersicht über den Import von Daten aus Amazon S3-Daten.

Weitere Informationen zu dieser Funktion finden Sie unter aws_s3.table_import_from_s3.

Die Funktion gibt `aws_s3.table_import_from_s3` zurück. Weitere Informationen zum Angeben von anderen Dateien für den Import aus einem Amazon S3-Bucket finden Sie in einem der folgenden Beispiele.

> **Note**
>
> Beim Importieren einer Datei mit 0 Byte tritt ein Fehler auf.

Themen

- Importieren einer Amazon S3-Datei, die ein benutzerdefiniertes Trennzeichen verwendet
- Importieren einer Amazon S3-komprimierten Datei (gzip)
- Importieren einer kodierten Amazon S3-Datei

Importieren einer Amazon S3-Datei, die ein benutzerdefiniertes Trennzeichen verwendet

Das folgende Beispiel zeigt, wie man eine Datei importiert, die ein benutzerdefiniertes Trennzeichen verwendet. Außerdem wird veranschaulicht, wie mit dem `column_list`-Parameter der Funktion aws_s3.table_import_from_s3 kontrolliert wird, wo die Daten in der Datenbanktabelle platziert werden.

Für dieses Beispiel wird angenommen, dass die folgenden Informationen in durch Pipe-Zeichen getrennte Spalten in der Amazon S3-Datei angeordnet sind.

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

So importieren Sie eine Datei, die ein benutzerdefiniertes Trennzeichen verwendet:

1. Erstellen Sie eine Tabelle in der Datenbank für die importierten Daten.

   ```
   postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
   ```

2. Verwenden Sie die folgende Form der Funktion aws_s3.table_import_from_s3, um Daten aus der Amazon S3-Datei zu importieren.

Zur Angabe der Datei können Sie auch den Funktionsaufruf aws_commons.create_s3_uri in den Funktionsaufruf `aws_s3.table_import_from_s3` einbinden.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   'test',
   'a,b,d,e',
   'DELIMITER ''|''',
   aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'pipeDelimitedSampleFile', 'us-
east-2')
);
```

Die Daten befinden sich nun in den folgenden Spalten der Tabelle.

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

Importieren einer Amazon S3-komprimierten Datei (gzip)

Das folgende Beispiel zeigt, wie eine mit gzip komprimierte Datei aus Amazon S3 importiert wird. Die Datei, die Sie importieren, muss die folgenden Amazon-S3-Metadaten aufweisen:

- Schlüssel: `Content-Encoding`
- Wert: `gzip`

Wenn Sie die Datei mit der AWS-Managementkonsole hochladen, werden die Metadaten normalerweise vom System angewendet. Informationen zum Hochladen von Dateien in Amazon S3 mit der AWS-Managementkonsole, der AWS CLI oder der API finden Sie unter Hochladen von Objekten im Benutzerhandbuch für Amazon Simple Storage Service.

Weitere Informationen zu Amazon-S3-Metadaten und zu vom System bereitgestellten Metadaten finden Sie unter Bearbeiten von Objektmetadaten in der Amazon-S3-Konsole im Benutzerhandbuch für Amazon Simple Storage Service.

Importieren Sie die gzip-Datei folgendermaßen in Ihrer Aurora PostgreSQL DB-Cluster-.

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
 'amzn-s3-demo-bucket', 'test-data.gz', 'us-east-2'
);
```

Importieren einer kodierten Amazon S3-Datei

Das folgende Beispiel zeigt, wie eine Datei aus Amazon S3 mit Windows-1252-Kodierung importiert wird.

```
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'SampleFile', 'us-east-2')
);
```

Funktionsreferenz

Funktionen

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

Importiert Amazon S3-Daten in eine Aurora PostgreSQL-Tabelle. Die Erweiterung `aws_s3` stellt die Funktion `aws_s3.table_import_from_s3` bereit. Der Rückgabewert ist Text.

Syntax

Die erforderlichen Parameter sind `table_name`, `column_list` und `options`. Diese Parameter identifizieren die Datenbanktabelle und geben an, wie die Daten in die Tabelle kopiert werden.

Sie können auch die folgenden Parameter verwenden:

- Die zu importierende Amazon S3-Datei wird mit dem Parameter `s3_info` übergeben. Wenn Sie diesen Parameter verwenden, wird der Zugriff auf Amazon S3 von einer IAM-Rolle für die PostgreSQL DB-Cluster- bereitgestellt.

```
aws_s3.table_import_from_s3 (
```

```
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- Die Anmeldeinformationen für den Zugriff auf Amazon S3 werden mit dem Parameter `credentials` übergeben. Mit diesem Parameter verwenden Sie keine IAM-Rolle.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
   credentials aws_commons._aws_credentials_1
)
```

Parameter

table_name

Eine erforderliche Textzeichenfolge mit dem Namen der PostgreSQL-Datenbanktabelle, in die die Daten importiert werden sollen.

column_list

Eine erforderliche Textzeichenfolge mit einer optionalen Liste der Tabellenspalten der PostgreSQL-Datenbank, in die die Daten kopiert werden sollen. Wenn die Zeichenfolge leer ist, werden alle Spalten der Tabelle verwendet. Ein Beispiel finden Sie unter Importieren einer Amazon S3-Datei, die ein benutzerdefiniertes Trennzeichen verwendet.

options

Eine erforderliche Textzeichenfolge mit Argumenten für den PostgreSQL `COPY`-Befehl. Diese Parameter legen fest, wie die Daten in die PostgreSQL-Tabelle kopiert werden. Weitere Informationen finden Sie in der PostgreSQL COPY-Dokumentation.

s3_info

Ein zusammengesetzter `aws_commons._s3_uri_1`-Typ mit den folgenden Informationen zum S3-Objekt:

- `bucket` – Der Name des Amazon S3-Buckets, der die Datei enthält.

- `file_path` – Der Amazon S3-Dateiname einschließlich des Pfads der Datei.
- `region`Die AWS-Region, in der sich die Datei befindet. Eine Liste der AWS-Regionsnamen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

Anmeldedaten

Ein zusammengesetzter `aws_commons._aws_credentials_1`-Typ mit den folgenden Anmeldeinformationen, die für den Importvorgang verwendet werden sollen:

- Zugriffsschlüssel
- Geheimschlüssel
- Sitzungs-Token

Hinweise zum Erstellen einer zusammengesetzten `aws_commons._aws_credentials_1`-Struktur finden Sie unter aws_commons.create_aws_credentials.

Alternative Syntax

Zum Testen können Sie statt der Parameter `s3_info` und `credentials` eine erweiterte Gruppe von Parametern verwenden. Nachfolgend sind weitere Syntaxvariationen für die Funktion `aws_s3.table_import_from_s3` aufgeführt:

- Statt den Parameter `s3_info` zum Identifizieren einer Amazon S3-Datei zu verwenden, nutzen Sie die Kombination aus den Parametern `bucket`, `file_path` und `region`. Mit dieser Form der Funktion wird der Zugriff auf Amazon S3 mit einer IAM-Rolle für die PostgreSQL-DB-Instance bereitgestellt.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- Statt den Parameter `credentials` zum Angeben einer Amazon S3-Datei zu verwenden, nutzen Sie die Kombination aus den Parametern `access_key`, `session_key` und `session_token`.

```
aws_s3.table_import_from_s3 (
```

```
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text,
   access_key text,
   secret_key text,
   session_token text
)
```

Alternative Parameter

bucket

Eine Textzeichenfolge mit den Namen des Amazon S3-Buckets, der die Datei enthält.

file_path

Eine Textzeichenfolge, die den Amazon S3-Dateinamen einschließlich des Pfades der Datei enthält.

Region

Eine Textzeichenfolge, die den AWS-Region-Speicherort der Datei kennzeichnet. Eine Liste der AWS-Region-Namen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

access_key

Eine Textzeichenfolge mit dem Zugriffsschlüssel, der für den Importvorgang verwendet werden soll. Der Standardwert ist „NULL“.

secret_key

Eine Textzeichenfolge mit dem Geheimschlüssel, der für den Importvorgang verwendet werden soll. Der Standardwert ist „NULL“.

session_token

(Optional) Eine Textzeichenfolge mit dem Sitzungsschlüssel, der für den Importvorgang verwendet werden soll. Der Standardwert ist „NULL“.

aws_commons.create_s3_uri

Erstellt eine `aws_commons._s3_uri_1`-Struktur für die Amazon S3-Dateiinformationen. Die Ergebnisse der Funktion `aws_commons.create_s3_uri` werden im Parameter `s3_info` der Funktion aws_s3.table_import_from_s3 verwendet.

Syntax

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

Parameter

bucket

Eine erforderliche Textzeichenfolge mit dem Namen des Amazon S3-Buckets für die Datei.

file_path

Eine erforderliche Textzeichenfolge, die den Amazon S3-Dateinamen einschließlich des Pfads der Datei enthält.

Region

Eine erforderliche Textzeichenfolge mit der AWS-Region, in der sich die Datei befindet. Eine Liste der AWS-Region-Namen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

aws_commons.create_aws_credentials

Legt einen Zugriffs- und einen Geheimschlüssel in einer `aws_commons._aws_credentials_1`-Struktur fest. Die Ergebnisse der Funktion `aws_commons.create_aws_credentials` werden im Parameter `credentials` der Funktion aws_s3.table_import_from_s3 verwendet.

Syntax

```
aws_commons.create_aws_credentials(
   access_key text,
   secret_key text,
```

```
   session_token text
)
```

Parameter

access_key

Eine erforderliche Textzeichenfolge mit dem Zugriffsschlüssel, der zum Importieren einer Amazon S3-Datei verwendet werden soll. Der Standardwert ist „NULL“.

secret_key

Eine erforderliche Textzeichenfolge mit dem Geheimschlüssel, der zum Importieren einer Amazon S3-Datei verwendet werden soll. Der Standardwert ist „NULL“.

session_token

Eine erforderliche Textzeichenfolge mit dem Sitzungs-Token, der zum Importieren einer Amazon S3-Datei verwendet werden soll. Der Standardwert ist „NULL“. Wenn Sie ein optionales `session_token` angeben, können Sie temporäre Anmeldeinformationen verwenden.

Exportieren von Daten aus einem/einer Aurora PostgreSQL-DB-Cluster zu Amazon S3

Sie können Daten aus einem Aurora-PostgreSQL-DB-Cluster abfragen und direkt in Dateien exportieren, die in einem Amazon-S3-Bucket gespeichert sind. Dazu installieren Sie zuerst die Erweiterung von Aurora PostgreSQL `aws_s3`. Diese Erweiterung stellt die Funktionen bereit, die Sie zum Exportieren von Abfrageergebnissen nach Amazon S3 verwenden. Im Folgenden erfahren Sie, wie Sie die Erweiterung installieren und Daten nach Amazon S3 exportieren.

Sie können nur von einer bereitgestellten oder einer DB-Instance von Aurora Serverless v2 aus exportieren. Diese Schritte werden für Aurora Serverless v1 nicht unterstützt.

> **Note**
>
> Kontoübergreifender Export nach Amazon S3 wird nicht unterstützt.

Alle derzeit verfügbaren Aurora-PostgreSQL-Versionen unterstützen den Export von Daten nach Amazon Simple Storage Service. Ausführliche Versionsinformationen finden Sie unter

Aktualisierungen von Amazon Aurora PostgreSQL im Abschnitt Versionshinweise für Aurora PostgreSQL.

Wenn Sie keinen Bucket für Ihren Export eingerichtet haben, lesen Sie die folgenden Themen im Benutzerhandbuch von Amazon Simple Storage Service.

- Einrichten von Amazon S3
- Erstellen eines Buckets

Standardmäßig verwenden die von Aurora PostgreSQL nach Amazon S3 exportierten Daten eine serverseitige Verschlüsselung mit Von AWS verwalteter Schlüssel. Sie können alternativ einen vom Kunden verwalteten Schlüssel verwenden, den Sie bereits erstellt haben. Wenn Sie eine Bucket-Verschlüsselung verwenden, muss der Amazon-S3-Bucket mit einem AWS Key Management Service-Schlüssel (AWS KMS) (SSE-KMS) verschlüsselt sein. Derzeit werden Buckets, die mit Schlüsseln verschlüsselt sind, die von Amazon S3 (SSE-S3) verwaltet werden, nicht unterstützt.

> **Note**
>
> Sie können DB- und DB-Cluster-Snapshot-Daten in Amazon S3 speichern, indem Sie die AWS-Managementkonsole, AWS CLI oder Amazon RDS API verwenden. Weitere Informationen finden Sie unter Exportieren von Daten von DB-Cluster-Snapshots nach Amazon S3.

Themen

- Installieren der Erweiterung aws_s3
- Übersicht über das Exportieren von Daten zu Amazon S3
- Angeben des Amazon S3-Dateipfads für den Export
- Einrichten des Zugriffs auf einen Amazon-S3-Bucket
- Exportieren von Abfragedaten mithilfe der Funktion aws_s3.query_export_to_s3
- Funktionsreferenz
- Fehlerbehebung beim Zugriff auf Amazon S3

Installieren der Erweiterung aws_s3

Bevor Sie Amazon Simple Storage Service mit Ihrem DB-Cluster voon Aurora PostgreSQL verwenden können, müssen Sie die Erweiterung `aws_s3` installieren. Diese Erweiterung bietet Funktionen zum Exportieren von Daten aus der Writer-Instance eines DB-Clusters von Aurora PostgreSQL in einen Amazon-S3-Bucket Sie stellt außerdem Funktionen zum Importieren von Daten aus Amazon S3 bereit. Weitere Informationen finden Sie unter Importieren von Amazon S3 in einen Aurora-PostgreSQL-DB-Cluster. Die Erweiterung `aws_s3` hängt von einigen Hilfsfunktionen in der Erweiterung `aws_commons` ab, die bei Bedarf automatisch installiert wird.

So installieren Sie die Erweiterung **`aws_s3`**

1. Verwenden Sie psql (oder pgAdmin), um eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL als Benutzer mit `rds_superuser`-Berechtigungen herzustellen. Wenn Sie beim Einrichten den Standardnamen beibehalten haben, stellen Sie eine Verbindung als `postgres` her.

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Führen Sie den folgenden Befehl aus, um die Erweiterung zu installieren.

   ```
   postgres=> CREATE EXTENSION aws_s3 CASCADE;
   NOTICE: installing required extension "aws_commons"
   CREATE EXTENSION
   ```

3. Wenn Sie überprüfen möchten, ob die Erweiterung installiert wurde, können Sie psql-Metabefehl `\dx` verwenden.

   ```
   postgres=> \dx
          List of installed extensions
       Name     | Version |   Schema   |                Description
   -------------+---------+------------+-------------------------------------------
    aws_commons | 1.2     | public     | Common data types across AWS services
    aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
    plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
   (3 rows)
   ```

Die Funktionen zum Importieren von Daten aus Amazon S3 und Exportieren von Daten nach Amazon S3 sind jetzt verfügbar.

Überprüfen, ob Ihre Version von Aurora PostgreSQL Exporte nach Amazon S3 unterstützt

Sie können überprüfen, ob Ihre Version von Aurora PostgreSQL den Export nach Amazon S3 unterstützt, indem Sie den Befehl `describe-db-engine-versions` verwenden. Im folgenden Beispiel wird überprüft, ob Version 10.14 nach Amazon S3 exportieren kann.

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

Wenn die Ausgabe die Zeichenfolge `"s3Export"` enthält, unterstützt die Engine Amazon S3-Exporte. Ansonsten unterstützt die Engine sie nicht.

Übersicht über das Exportieren von Daten zu Amazon S3

Verwenden Sie das folgende Verfahren, um in einer Aurora PostgreSQL Datenbank gespeicherte Daten in einen Amazon S3 Bucket zu exportieren.

So exportieren Sie Aurora PostgreSQL Daten nach S3

1. Identifizieren Sie einen Amazon S3-Dateipfad, der zum Exportieren von Daten verwendet werden soll. Weitere Informationen zu diesem Prozess finden Sie unter Angeben des Amazon S3-Dateipfads für den Export.
2. Erteilen Sie die Berechtigung für den Zugriff auf den Amazon S3-Bucket.

   Um Daten in eine Amazon S3-Datei zu exportieren, erteilen Sie dem/der Aurora PostgreSQL-DB-Cluster die Berechtigung, auf den Amazon S3-Bucket zuzugreifen, den der Export für die Speicherung verwendet. Dazu gehören die folgenden Schritte:

   1. Erstellen Sie eine IAM-Richtlinie, die Zugriff auf einen Amazon S3-Bucket bietet, in den Sie exportieren möchten.
   2. Erstellen Sie eine IAM-Rolle.
   3. Fügen Sie die erstellte Richtlinie an die erstellte Rolle an.
   4. Fügen Sie diese IAM-Rolle zu Ihrem DB-Cluster Ihrer hinzu.

   Weitere Informationen zu diesem Prozess finden Sie unter Einrichten des Zugriffs auf einen Amazon-S3-Bucket.

3. Identifizieren Sie eine Datenbankabfrage, um die Daten abzurufen. Exportieren Sie die Abfragedaten, indem Sie die Funktion `aws_s3.query_export_to_s3` aufrufen.

   Nachdem Sie die vorangegangenen Vorbereitungsaufgaben abgeschlossen haben, verwenden Sie die aws_s3.query_export_to_s3-Funktion, um Abfrageergebnisse in Amazon S3 zu exportieren. Weitere Informationen zu diesem Prozess finden Sie unter Exportieren von Abfragedaten mithilfe der Funktion aws_s3.query_export_to_s3.

Angeben des Amazon S3-Dateipfads für den Export

Geben Sie die folgenden Informationen an, um den Speicherort in Amazon S3 zu identifizieren, in den Sie Daten exportieren möchten:

- Bucket-Name – Ein Bucket ist ein Container für Amazon S3-Objekte oder -Dateien.

  Weitere Informationen zum Speichern von Daten mit Amazon S3 finden Sie unter Erstellen eines Buckets und Arbeiten mit Objekten im Benutzerhandbuch für Amazon Simple Storage Service.
- Dateipfad – Der Dateipfad gibt an, wo der Export im Amazon S3-Bucket gespeichert wird. Der Dateipfad besteht aus Folgendem:
  - Ein optionales Pfadpräfix, das einen Pfad für virtuelle Ordner identifiziert.
  - Ein Dateipräfix, das eine oder mehrere Dateien identifiziert, die gespeichert werden sollen. Größere Exporte werden in mehreren Dateien gespeichert, jeweils mit einer maximalen Größe von ca. 6 GB. Die zusätzlichen Dateinamen haben das gleiche Dateipräfix, aber mit `_part`*XX* angefügt. *XX* stellt 2, dann 3 usw. dar.

  Beispielsweise ist ein Dateipfad mit einem `exports`-Ordner und einem `query-1-export`-Dateipräfix `/exports/query-1-export`.
- AWS-Region (optional) – Die AWS-Region, in der sich der Amazon-S3-Bucket befindet. Wenn Sie keinen Wert für die AWS-Region angeben, speichert Aurora Ihre Dateien in Amazon S3 in derselben AWS-Region wie der/die exportierende DB-Cluster.

> **Note**
>
> Derzeit muss die AWS-Region mit der Region des/der exportierenden DB-Cluster übereinstimmen.

Eine Liste der AWS-Regionsnamen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

Um die Amazon S3-Dateiinformationen darüber zu speichern, wo der Export gespeichert werden soll, können Sie die aws_commons.create_s3_uri-Funktion verwenden, um eine zusammengesetzte `aws_commons._s3_uri_1`-Struktur wie folgt zu erstellen.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

Sie geben diesen `s3_uri_1`-Wert später als Parameter im Aufruf der aws_s3.query_export_to_s3-Funktion an. Beispiele finden Sie unter Exportieren von Abfragedaten mithilfe der Funktion aws_s3.query_export_to_s3.

Einrichten des Zugriffs auf einen Amazon-S3-Bucket

Um Daten zu Amazon S3 zu exportieren, erteilen Sie Ihrem PostgreSQL-DB-Cluster die Berechtigung, auf den Amazon S3-Bucket zuzugreifen, in den die Dateien aufgenommen werden sollen.

Führen Sie dazu die folgenden Schritte aus.

So erteilen Sie einem PostgreSQL-DB-Cluster Zugriff auf Amazon S3 über eine IAM-Rolle

1. Erstellen Sie eine IAM-Richtlinie.

   Diese Richtlinie enthält die Bucket- und Objektberechtigungen, die IhremPostgreSQL-DB-Cluster den Zugriff auf Amazon S3 ermöglichen.

   Führen Sie beim Erstellen dieser Richtlinie die folgenden Schritte aus:

   a. Nehmen Sie die folgenden erforderlichen Aktionen in die Richtlinie auf, um die Übertragung von Dateien aus Ihrem PostgreSQL-DB-Cluster in einen Amazon S3-Bucket zu gestatten:

      - `s3:PutObject`
      - `s3:AbortMultipartUpload`

b. Geben Sie den Amazon-Ressourcennamen (ARN) ein, der den Amazon S3-Bucket und die Objekte im Bucket identifiziert. Das ARN-Format für den Zugriff auf Amazon S3 lautet: `arn:aws:s3:::amzn-s3-demo-bucket/*`

Weitere Informationen zum Erstellen einer IAM-Richtlinie für Aurora PostgreSQL finden Sie unter Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff. Siehe auch Tutorial: Erstellen und Anfügen Ihrer ersten vom Kunden verwalteten Richtlinie im IAM-Benutzerhandbuch.

Mit dem folgenden AWS CLI Befehl wird eine IAM-Richtlinie `rds-s3-export-policy` mit diesen Optionen erstellt. Sie gewährt Zugriff auf einen Bucket namens *amzn-s3-demo-bucket*.

> **Warning**
>
> Es wird empfohlen, die Datenbank innerhalb einer privaten VPC einzurichten, deren Endpunktrichtlinien für den Zugriff auf bestimmte Buckets konfiguriert sind. Weitere Informationen finden Sie unter Verwenden von Endpunktrichtlinien für Amazon S3 im Amazon VPC Benutzerhandbuch.
> Es wird dringend empfohlen, keine Richtlinie mit Zugriff auf alle Ressourcen zu erstellen. Dieser Zugriff kann eine Bedrohung für die Datensicherheit darstellen.
> Wenn Sie eine Richtlinie erstellen, die `S3:PutObject` Zugriff auf alle Ressourcen mit `"Resource":"*"` gewährt, kann ein Benutzer mit Exportberechtigungen Daten in alle Buckets in Ihrem Konto exportieren. Darüber hinaus kann der Benutzer Daten in jeden öffentlich beschreibbaren Bucket in Ihrer AWS -Region exportieren.

Notieren Sie nach dem Erstellen der Richtlinie den Amazon-Ressourcennamen (ARN) der Richtlinie. Sie benötigen den ARN für einen nachfolgenden Schritt, in dem Sie die Richtlinie an eine IAM-Rolle anhängen.

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
           "s3:PutObject*",
           "s3:ListBucket",
```

```
          "s3:GetObject*",
          "s3:DeleteObject*",
          "s3:GetBucketLocation",
          "s3:AbortMultipartUpload"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket/*"
        ]
      }
    ]
  }'
```

2. Erstellen Sie eine IAM-Rolle.

   Sie tun dies, damit Aurora PostgreSQL in Ihrem Namen diese IAM-Rolle übernehmen kann, um auf Ihre Amazon S3-Buckets zuzugreifen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen IAM-Benutzer im IAM-Benutzerhandbuch.

   Wir empfehlen die Verwendung von `aws:SourceArn` und `aws:SourceAccount` globaler Bedingungskontext-Schlüssel in ressourcenbasierten Richtlinien, um die Berechtigungen des Dienstes auf eine bestimmte Ressource zu beschränken. Dies ist der effektivste Weg, um sich vor dem verwirrtes Stellvertreterproblem zu schützen.

   Wenn Sie sowohl globale Kontextschlüssel nutzen und der `aws:SourceArn`-Wert enthält die Konto-ID, muss der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn`-Wert die gleiche Konto-ID verwenden, wenn er in der gleichen Richtlinienanweisung verwendet wird.

   - Verwenden von `aws:SourceArn` wenn Sie einen serviceübergreifenden Zugriff für eine einzelne Ressource wünschen.
   - Verwenden von `aws:SourceAccount` wenn Sie zulassen möchten, dass eine Ressource in diesem Konto mit der betriebsübergreifenden Verwendung verknüpft wird.

   Verwenden Sie in der Richtlinie den `aws:SourceArn` globalen Kontextschlüssel mit dem vollständigen ARN der Ressource. Das folgende Beispiel zeigt, wie Sie dazu den AWS CLI Befehl verwenden, um eine Rolle mit dem Namen `rds-s3-export-role` zu erstellen.

   Example

   Für Linux, macOS oder Unix:

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
             }
       }
     ]
   }'
```

Für Windows:

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:dbname"
                }
             }
```

```
      }
    ]
}'
```

3. Fügen Sie die erstellte IAM-Richtlinie der IAM-Rolle an, die Sie erstellt haben.

   Mit dem folgenden AWS CLI Befehl wird die zuvor erstellte Richtlinie an die Rolle `rds-s3-export-role`. Replace *`your-policy-arn`* mit dem Richtlinien-ARN angehängt, den Sie in einem früheren Schritt notiert haben.

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-export-role
   ```

4. Fügen Sie die IAM-Rolle der DB Cluster hinzu. Sie tun dies, indem Sie das AWS-Managementkonsole oder verwenden AWS CLI, wie im Folgenden beschrieben.

Konsole

So fügen Sie eine IAM-Rolle für eine PostgreSQL DB-Cluster- über die Konsole hinzu:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie den Namen der PostgreSQL DB-Cluster- aus, um ihre Details anzuzeigen.
3. Wählen Sie auf der Registerkarte Connectivity & security (Konnektivität und Sicherheit) im Bereich Manage IAM roles (IAM-Rollen verwalten) die Rolle aus, die unter Add IAM roles to this instance (IAM-Rollen zu dieser Instance hinzufügen) hinzugefügt werden soll.
4. Wählen Sie unter Funktion die Option s3Export aus.
5. Wählen Sie Rolle hinzufügen.

AWS CLI

So fügen Sie eine IAM-Rolle für einen PostgreSQL-DB-Cluster mithilfe der CLI hinzu:

- Verwenden Sie den folgenden Befehl, um die Rolle mit dem Namen `my-db-cluster` zum PostgreSQL DB-Cluster hinzuzufügen. Ersetzen Sie *`your-role-arn`* durch den Rollen-ARN, den Sie im vorherigen Schritt notiert haben. Verwenden Sie `s3Export` für den Wert der `--feature-name`-Option.

Example

Für Linux, macOS oder Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Export \
   --role-arn your-role-arn   \
   --region your-region
```

Für Windows:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Export ^
   --role-arn your-role-arn ^
   --region your-region
```

Exportieren von Abfragedaten mithilfe der Funktion aws_s3.query_export_to_s3

Exportieren Sie Ihre PostgreSQL-Daten nach Amazon S3, indem Sie die aws_s3.query_export_to_s3-Funktion aufrufen.

Themen

- Voraussetzungen
- Aufruf von aws_s3.query_export_to_s3
- Exportieren in eine CSV-Datei, die ein benutzerdefiniertes Trennzeichen verwendet
- Exportieren in eine Binärdatei mit Codierung

Voraussetzungen

Bevor Sie die `aws_s3.query_export_to_s3`-Funktion verwenden, müssen Sie die folgenden Voraussetzungen erfüllen:

- Installieren Sie die erforderlichen PostgreSQL-Erweiterungen, wie unter beschriebe Übersicht über das Exportieren von Daten zu Amazon S3.

- Legen Sie fest, wohin die Daten nach Amazon S3 exportiert werden sollen, wie unter beschriebe Angeben des Amazon S3-Dateipfads für den Export.
- Stellen Sie sicher, dass die Exportzugriff auf Amazon S3 hat (wie unter Einrichten des Zugriffs auf einen Amazon-S3-Bucket beschrieben).

In den folgenden Beispielen wird eine Datenbanktabelle namens verwende `sample_table`. In diesen Beispielen werden die Daten in einen Bucket namens *`amzn-s3-demo-bucket`* exportiert. Die Beispieltabelle und die Daten werden mit den folgenden SQL-Anweisungen in psql erstellt.

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

Aufruf von aws_s3.query_export_to_s3

Im Folgenden werden die grundlegenden Möglichkeiten zum Aufrufen der aws_s3.query_export_to_s3-Funktion dargestellt.

In diesen Beispielen wird die Variable `s3_uri_1` zum Identifizieren einer Struktur verwendet, die die Informationen zur Identifizierung der Amazon S3-Datei enthält. Verwenden Sie die Funktion aws_commons.create_s3_uri, um die Struktur zu erstellen.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

Obwohl die Parameter für die folgenden zwei `aws_s3.query_export_to_s3`-Funktionsaufrufe unterschiedlich sind, sind die Ergebnisse für diese Beispiele identisch. Alle Zeilen der `sample_table`-Tabelle werden in einen Bucket namens *`amzn-s3-demo-bucket`* exportiert.

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

Die Parameter werden wie folgt beschrieben:

- `'SELECT * FROM sample_table'` – Der erste Parameter ist eine erforderliche Textzeichenfolge, die eine SQL-Abfrage enthält. Die PostgreSQL-Engine führt diese Abfrage aus. Die Ergebnisse der Abfrage werden in den S3-Bucket kopiert, der in anderen Parametern identifiziert wurde.
- `:'s3_uri_1'` – Dieser Parameter ist eine Struktur, die die Datei Amazon S3 identifiziert. In diesem Beispiel wird die zuvor erstellte Struktur anhand einer Variablen identifiziert. Sie können die Struktur stattdessen erstellen, indem Sie den Funktionsaufruf `aws_commons.create_s3_uri` wie folgt inline in den Funktionsaufruf `aws_s3.query_export_to_s3` einschließen.

```
SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
   aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'sample-filepath', 'us-west-2')
);
```

- `options :='format text'` – Der `options`-Parameter ist eine optionale Textzeichenfolge, die `COPY`-PostgreSQL-Argumente enthält. Beim Kopiervorgang werden die Argumente und das Format des PostgreSQL COPY-Befehls verwendet.

Wenn die angegebene Datei nicht im Amazon S3-Bucket vorhanden ist, wird sie erstellt. Wenn die Datei bereits vorhanden ist, wird sie überschrieben. Die Syntax für den Zugriff auf die exportierten Daten in Amazon S3 ist die folgende.

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

Größere Exporte werden in mehreren Dateien gespeichert, jeweils mit einer maximalen Größe von ca. 6 GB. Die zusätzlichen Dateinamen haben das gleiche Dateipräfix, aber mit `_part`*XX* angefügt. *XX* stellt 2, dann 3 usw. dar. Angenommen, Sie geben den Pfad, in dem Sie Datendateien speichern, wie folgt an.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
```

Wenn der Export drei Datendateien anlegen muss, enthält der Amazon S3-Bucket die folgenden Datendateien.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part2
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part3
```

Die vollständige Referenz für diese Funktion und weitere Aufrufmöglichkeiten finden Sie unter aws_s3.query_export_to_s3. Weitere Informationen zum Zugriff auf Dateien in Amazon S3 finden Sie unter View an object im Amazon Simple Storage Service User Guide.

Exportieren in eine CSV-Datei, die ein benutzerdefiniertes Trennzeichen verwendet

Das folgende Beispiel zeigt, wie die aws_s3.query_export_to_s3-Funktion zum Exportieren von Daten in eine Datei aufgerufen wird, die ein benutzerdefiniertes Trennzeichen verwendet. Im Beispiel werden Argumente des Befehls PostgreSQL COPY verwendet, um das CSV-Dateiformat (durch Kommas getrennte Werte) und ein Doppelpunkt (:)-Trennzeichen anzugeben.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

Exportieren in eine Binärdatei mit Codierung

Das folgende Beispiel zeigt, wie die aws_s3.query_export_to_s3-Funktion zum Exportieren von Daten in eine Binärdatei mit Windows-1253-Codierung aufgerufen wird.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

Funktionsreferenz

Funktionen

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

aws_s3.query_export_to_s3

Exportiert ein PostgreSQL-Abfrageergebnis in einen Amazon S3-Bucket. Die Erweiterung `aws_s3` stellt die Funktion `aws_s3.query_export_to_s3` bereit.

Die zwei erforderlichen Parameter sind `query` und `s3_info`. Diese definieren die zu exportierende Abfrage und identifizieren den Amazon S3-Bucket, in den exportiert werden soll. Ein optionaler Parameter namens `options` ermöglicht die Definition verschiedener Exportparameter. Beispiele für die Verwendung der `aws_s3.query_export_to_s3`-Funktion finden Sie unter Exportieren von Abfragedaten mithilfe der Funktion aws_s3.query_export_to_s3.

Syntax

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
    kms_key text
)
```

Eingabeparameter

query

Eine erforderliche Textzeichenfolge, die eine SQL-Abfrage enthält, die von der PostgreSQL-Engine ausgeführt wird. Die Ergebnisse dieser Abfrage werden in einen S3-Bucket kopiert, der im `s3_info`-Parameter identifiziert wurde.

s3_info

Ein zusammengesetzter `aws_commons._s3_uri_1`-Typ mit den folgenden Informationen zum S3-Objekt:

- `bucket` – Der Name des Amazon S3-Buckets, der die Datei enthalten soll.
- `file_path` – Der Amazon S3-Dateiname und der -Pfad.
- `region` – Die AWS-Region, in der sich der Bucket befindet. Eine Liste der AWS-Regionsnamen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

  Derzeit muss dieser Wert die gleiche AWS-Region wie der/die exportierende DB-Cluster sein. Der Standard ist die AWS-Region der exportierenden DB-Cluster .

Informationen zum Erstellen einer zusammengesetzten `aws_commons._s3_uri_1`-Struktur finden Sie in der aws_commons.create_s3_uri-Funktion.

options

Eine optionale Textzeichenfolge mit Argumenten für den PostgreSQL `COPY`-Befehl. Diese Argumente geben an, wie die Daten beim Exportieren kopiert werden sollen. Weitere Informationen finden Sie in der PostgreSQL COPY-Dokumentation.

kms_key text

Eine optionale Textzeichenfolge, die den vom Kunden verwalteten KMS-Schlüssel des S3-Buckets enthält, in den die Daten exportiert werden sollen.

Alternative Eingabeparameter

Zum Testen können Sie statt des Parameters `s3_info` eine erweiterte Gruppe von Parametern verwenden. Nachfolgend sind weitere Syntaxvariationen für die Funktion `aws_s3.query_export_to_s3` aufgeführt.

Statt den Parameter `s3_info` zum Identifizieren einer Amazon S3-Datei zu verwenden, nutzen Sie die Kombination aus den Parametern `bucket`, `file_path` und `region`.

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

Eine erforderliche Textzeichenfolge, die eine SQL-Abfrage enthält, die von der PostgreSQL-Engine ausgeführt wird. Die Ergebnisse dieser Abfrage werden in einen S3-Bucket kopiert, der im `s3_info`-Parameter identifiziert wurde.

bucket

Eine erforderliche Textzeichenfolge mit dem Namen des Amazon S3-Buckets, der die Datei enthält.

file_path

Eine erforderliche Textzeichenfolge, die den Amazon S3-Dateinamen einschließlich des Pfads der Datei enthält.

Region

Eine optionale Textzeichenfolge mit der AWS-Region, in der sich der Bucket befindet. Eine Liste der AWS-Regionsnamen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

Derzeit muss dieser Wert die gleiche AWS-Region wie der/die exportierende DB-Cluster sein. Der Standard ist die AWS-Region der exportierenden DB-Cluster .

options

Eine optionale Textzeichenfolge mit Argumenten für den PostgreSQL COPY-Befehl. Diese Argumente geben an, wie die Daten beim Exportieren kopiert werden sollen. Weitere Informationen finden Sie in der PostgreSQL COPY-Dokumentation.

kms_key text

Eine optionale Textzeichenfolge, die den vom Kunden verwalteten KMS-Schlüssel des S3-Buckets enthält, in den die Daten exportiert werden sollen.

Ausgabeparameter

```
aws_s3.query_export_to_s3(
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

Die Anzahl der Tabellenzeilen, die für die angegebene Abfrage erfolgreich in Amazon S3 hochgeladen wurden.

files_uploaded

Die Anzahl der in Amazon S3 hochgeladenen Dateien. Dateien werden in Größen von ca. 6 GB erstellt. Jeder weiteren erstellten Datei wird `_part`*XX* an den Namen angehängt. *XX* stellt 2, dann 3 usw. nach Bedarf dar.

bytes_uploaded

Die Gesamtanzahl der in Amazon S3 hochgeladenen Bytes.

Beispiele

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-demo-bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

Erstellt eine `aws_commons._s3_uri_1`-Struktur für die Amazon S3-Dateiinformationen. Die Ergebnisse der Funktion `aws_commons.create_s3_uri` werden im Parameter `s3_info` der Funktion aws_s3.query_export_to_s3 verwendet. Ein Beispiel für die Verwendung der `aws_commons.create_s3_uri`-Funktion finden Sie unter Angeben des Amazon S3-Dateipfads für den Export.

Syntax

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

Eingabeparameter

bucket

Eine erforderliche Textzeichenfolge mit dem Namen des Amazon S3-Buckets für die Datei.

file_path

Eine erforderliche Textzeichenfolge, die den Amazon S3-Dateinamen einschließlich des Pfads der Datei enthält.

Region

Eine erforderliche Textzeichenfolge mit der AWS-Region, in der sich die Datei befindet. Eine Liste der AWS-Regionsnamen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

Fehlerbehebung beim Zugriff auf Amazon S3

Wenn beim Versuch, Daten nach Amazon S3 zu exportieren, Verbindungsprobleme auftreten, bestätigen Sie zunächst, dass die Regeln für den ausgehenden Zugriff für die mit Ihrer DB-Instance verknüpfte VPC-Sicherheitsgruppe Netzwerkkonnektivität zulassen. Insbesondere muss die Sicherheitsgruppe über eine Regel verfügen, die es der DB-Instance erlaubt, TCP-Datenverkehr über Port 443 und an eine beliebige IPv4-Adresse (0.0.0.0/0) zu senden. Weitere Informationen

finden Sie unter Ermöglichen des Zugriffs auf den DB-Cluster in der VPC durch Erstellen einer Sicherheitsgruppe.

Empfehlungen finden Sie im Folgenden:

- Fehlerbehebung für Amazon Aurora-Identität und -Zugriff
- Fehlerbehebung bei Amazon S3 im Entwicklerhandbuch für Amazon Simple Storage Service
- Fehlerbehebung bei Amazon S3 und IAM im IAM-Benutzerhandbuch

Aufrufen einer AWS Lambda Funktion aus einem Aurora PostgreSQL-DB-Cluster (PostgreSQL-DB-Instance)

AWS Lambda ist ein ereignisgesteuerter Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Es ist für die Verwendung mit vielen AWS Diensten verfügbar, einschließlich Aurora PostgreSQL PostgreSQL. Sie können beispielsweise Lambda-Funktionen verwenden, um Ereignisbenachrichtigungen aus einer Datenbank zu verarbeiten oder Daten aus Dateien zu laden, wann immer eine neue Datei in Amazon S3 hochgeladen wird. Weitere Informationen zu Lambda finden Sie unter Was ist AWS Lambda? im AWS Lambda Entwicklerhandbuch.

> **Note**
>
> Das Aufrufen von AWS Lambda Funktionen wird in Aurora PostgreSQL 11.9 und höher (einschließlich) unterstützt. Aurora Serverless v2

Im Folgenden finden Sie Zusammenfassungen der notwendigen Schritte.

Weitere Informationen über Lambda-Funktionen finden Sie unter Erste Schritte mit Lambda und Grundlagen von AWS Lambda im AWS Lambda -Entwicklerhandbuch.

Themen

- Schritt 1: Konfigurieren Sie Ihren Aurora PostgreSQL DB-Cluster für ausgehende Verbindungen zu AWS Lambda
- Schritt 2: Konfigurieren Sie IAM für Ihren Aurora PostgreSQL-DB-Cluster, und AWS Lambda
- Schritt 3: Installieren der aws_lambda-Erweiterung für einen Aurora-PostgreSQL-DB-Cluster

- Schritt 4: Verwenden von Lambda-Hilfsfunktionen mit Ihrem Aurora-PostgreSQL-DB-Cluster (optional)
- Schritt 5: Aufrufen einer Lambda-Funktion von Ihrem Aurora-PostgreSQL-DB-Cluster
- Schritt 6: Erteilen der Berechtigung, Lambda-Funktionen aufzurufen, für andere Benutzer
- Beispiele: Aufrufen von Lambda-Funktionen von Ihrem Aurora-PostgreSQL-DB-Cluster
- Fehlermeldungen von Lambda-Funktionen
- Referenz für AWS Lambda-Funktionen und -Parameter

Schritt 1: Konfigurieren Sie Ihren Aurora PostgreSQL DB-Cluster für ausgehende Verbindungen zu AWS Lambda

Lambda-Funktionen werden immer in einer Amazon-VPC ausgeführt, die dem AWS Lambda Service gehört. Lambda wendet Netzwerkzugriffs- und Sicherheitsregeln auf diese VPC an und pflegt und überwacht die VPC automatisch. Ihr DB-Cluster von Aurora-PostgreSQL muss Netzwerkdatenverkehr an die VPC des Lambda-Services senden. Wie Sie dies konfigurieren, hängt davon ab, ob die primäre DB-Instance Ihres Aurora-DB-Clusters öffentlich oder privat ist.

- Öffentlicher Aurora PostgreSQL-DB-Cluster — Die primäre eines DB-Clusters ist öffentlich, wenn sie sich in einem öffentlichen Subnetz auf Ihrer VPC befindet und wenn die Eigenschaft "" der Instance lautet. PubliclyAccessible `true` Um den Wert dieser Eigenschaft zu ermitteln, können Sie den Befehl verwenden. describe-db-instances AWS CLI Alternativ können Sie über die AWS-Managementkonsole die Registerkarte Connectivity & security (Konnektivität und Sicherheit) öffnen und prüfen, ob Publicly accessible (Öffentlich zugänglich) auf Yes (Ja) eingestellt ist. Wenn Sie überprüfen möchten, ob sich die Instance im öffentlichen Subnetz Ihrer VPC befindet, können Sie die AWS-Managementkonsole oder die AWS CLI verwenden.

  Um den Zugriff auf Lambda einzurichten, verwenden Sie AWS-Managementkonsole oder, AWS CLI um eine ausgehende Regel für die Sicherheitsgruppe Ihrer VPC zu erstellen. Die ausgehende Regel gibt an, dass TCP Port 443 verwenden kann, um Pakete an beliebige IPv4 Adressen (0.0.0.0/0) zu senden.
- Private Aurora PostgreSQL DB-Cluster — In diesem Fall befindet sich die Eigenschaft "PubliclyAccessible" der Instance in einem privaten Subnetz `false` oder sie befindet sich in einem privaten Subnetz. Damit die Instance mit Lambda arbeiten kann, können Sie ein Network Address Translation (NAT)-Gateway verwenden. Weitere Informationen finden Sie unter NAT-Gateways. Sie können Ihre VPC auch mit einem VPC-Endpunkt für Lambda konfigurieren. Weitere Informationen finden Sie unter VPC-Endpunkte im Amazon-VPC-Benutzerhandbuch. Der Endpunkt

gibt Antworten auf Aufrufe Ihrer Lambda-Funktionen von Ihrem DB-Cluster von Aurora PostgreSQL zurück.

Ihre VPC kann jetzt auf Netzwerkebene mit der AWS Lambda VPC interagieren. Als Nächstes konfigurieren Sie die Berechtigungen mithilfe von IAM.

Schritt 2: Konfigurieren Sie IAM für Ihren Aurora PostgreSQL-DB-Cluster, und AWS Lambda

Das Aufrufen von Lambda-Funktionen von Ihrem Aurora-PostgreSQL-DB-Cluster erfordert bestimmte Berechtigungen. Um die erforderlichen Berechtigungen zu konfigurieren, empfehlen wir Ihnen, eine IAM-Richtlinie zu erstellen, die es ermöglicht, Lambda-Funktionen aufzurufen, diese Richtlinie einer Rolle zuzuweisen und die Rolle dann auf Ihren DB-Cluster anzuwenden. Dieser Ansatz gewährt dem DB-Cluster Berechtigungen zum Aufrufen der angegebenen Lambda-Funktion in Ihrem Namen. Nachfolgend wird beschrieben, wie Sie dazu die AWS CLI verwenden können.

IAM-Berechtigungen für die Verwendung Ihrer Cluster mit Lambda konfigurieren

1. Verwenden Sie den AWS CLI Befehl create-policy, um eine IAM-Richtlinie zu erstellen, die es Ihrer Aurora ermöglicht, die angegebene Lambda-Funktion aufzurufen. (Die Anweisungs-ID (Sid) ist eine optionale Beschreibung für Ihre Richtlinienanweisung und hat keine Auswirkungen auf die Verwendung.) Diese Richtlinie gewährt Ihrem Aurora-DB-Cluster die Mindestberechtigungen zum Aufrufen der angegebenen Lambda-Funktion.

   ```
   aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
       "Version": "2012-10-17",
       "Statement": [
           {
           "Sid": "AllowAccessToExampleFunction",
           "Effect": "Allow",
           "Action": "lambda:InvokeFunction",
           "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
           }
       ]
   }'
   ```

   Alternativ können Sie die vordefinierte `AWSLambdaRole`-Richtlinie verwenden, mit der Sie alle Ihre Lambda-Funktionen aufrufen können. Weitere Informationen finden Sie unter Identitätsbasierte IAM-Richtlinien für Lambda.

2. Verwenden Sie den AWS CLI Befehl create-role, um eine IAM-Rolle zu erstellen, die die Richtlinie zur Laufzeit annehmen kann.

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. Wenden Sie die Richtlinie mithilfe des Befehls auf die Rolle an. attach-role-policy AWS CLI

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/rds/add-role-to-db-cluster.html AWS CLI Dieser letzte Schritt erlaubt den Datenbankbenutzern Ihres DB-Clusters, Lambda-Funktionen aufzurufen.

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

Wenn die VPC- und die IAM-Konfigurationen abgeschlossen sind, können Sie jetzt die `aws_lambda`-Erweiterung installieren. (Beachten Sie, dass Sie die Erweiterung jederzeit installieren können, bis Sie jedoch die richtige VPC-Unterstützung und IAM-Berechtigungen eingerichtet haben, hat die `aws_lambda`-Erweiterung keinerlei Auswirkungen auf die Funktionen Ihrer Aurora-PostgreSQL-DB-Cluster.)

Schritt 3: Installieren der **aws_lambda**-Erweiterung für einen Aurora-PostgreSQL-DB-Cluster

Diese Erweiterung bietet Ihrem Aurora-PostgreSQL-DB-Cluster die Möglichkeit, Lambda-Funktionen von PostgreSQL aus aufzurufen.

aws_lambda-Erweiterung in einem Aurora-PostgreSQL-DB-Cluster installieren

Verwenden Sie die PostgreSQL-`psql`-Befehlszeile oder das pgAdmin-Tool, um Ihren Aurora-PostgreSQL-DB-Cluster zu verbinden.

1. Verbinden Sie Ihren Aurora-PostgreSQL-DB-Cluster als Benutzer mit `rds_superuser`-Berechtigungen. Im Beispiel wird der `postgres`-Standardbenutzer dargestellt.

   ```
   psql -h cluster-instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p
    5432
   ```

2. Installieren Sie die `aws_lambda`-Erweiterung. Die `aws_commons`-Erweiterung ist auch erforderlich. Sie bietet Hilfsfunktionen für `aws_lambda` und viele andere Aurora-Erweiterungen für PostgreSQL. Wenn Sie noch nicht auf Ihrem Aurora-PostgreSQL-DB-Cluster installiert ist, wird sie mit `aws_lambda` wie folgt installiert.

   ```
   CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
   NOTICE:  installing required extension "aws_commons"
   CREATE EXTENSION
   ```

Die `aws_lambda`-Erweiterung wird in Ihrer primären, auf Ihrem Aurora-PostgreSQL-DB-Cluster befindlichen DB-Instance installiert. Sie können jetzt praktische Strukturen für den Aufruf Ihrer Lambda-Funktionen erstellen.

Schritt 4: Verwenden von Lambda-Hilfsfunktionen mit Ihrem Aurora-PostgreSQL-DB-Cluster (optional)

Sie können die Hilfsfunktionen in der `aws_commons`-Erweiterung verwenden, um Entitäten vorzubereiten, die sich einfacher über PostgreSQL aufrufen lassen. Hierzu sind die folgenden Informationen zu Ihren Lambda-Funktionen erforderlich:

- Funktionsname – Der Name, der Amazon-Ressourcenname (ARN), die Version oder der Alias der Lambda-Funktion. Die in Schritt 2: Konfigurieren von IAM für Ihren Cluster und Lambda

erstellte IAM-Richtlinie benötigt den ARN, daher empfehlen wir Ihnen, den ARN Ihrer Funktion zu verwenden.

- AWS Region — (Optional) Die AWS Region, in der sich die Lambda-Funktion befindet, wenn sie sich nicht in derselben Region wie Ihre Aurora befindet.

Um die Daten zum Lambda-Funktionsnamen zu speichern, verwenden Sie die aws_commons.create_lambda_function_arn-Funktion. Diese Hilfsfunktion erstellt eine zusammengesetzte `aws_commons._lambda_function_arn_1`-Struktur mit den Details, die von der Aufruffunktion benötigt werden. Im Folgenden finden Sie drei alternative Ansätze zum Einrichten dieser zusammengesetzten Struktur.

```
SELECT aws_commons.create_lambda_function_arn(
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

Jeder dieser Werte kann in Aufrufen der aws_lambda.invoke-Funktion verwendet werden. Beispiele finden Sie unter Schritt 5: Aufrufen einer Lambda-Funktion von Ihrem Aurora-PostgreSQL-DB-Cluster.

Schritt 5: Aufrufen einer Lambda-Funktion von Ihrem Aurora-PostgreSQL-DB-Cluster

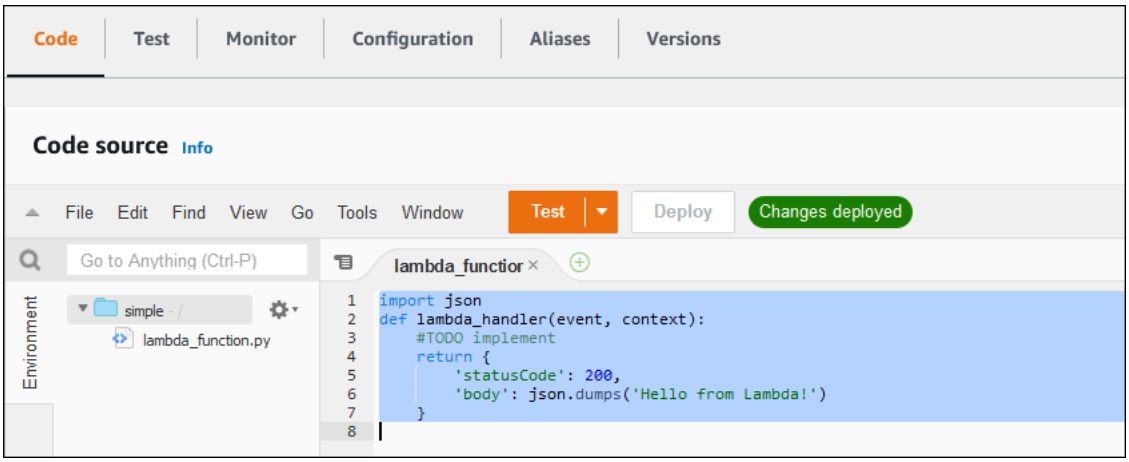
Die `aws_lambda.invoke`-Funktion verhält sich synchron oder asynchron, je nach `invocation_type`. Die beiden Alternativen für diesen Parameter sind `RequestResponse` (der Standardwert) und `Event`, wie folgt.

- **`RequestResponse`** – Dieser Aufrufstyp ist synchron. Dies ist das Standardverhalten, wenn der Aufruf erfolgt, ohne einen Aufruftyp anzugeben. Die Antwort-Nutzlast beinhaltet die Ergebnisse der

`aws_lambda.invoke`-Funktion. Verwenden Sie diesen Aufruftyp, wenn Ihr Workflow Ergebnisse von der Lambda-Funktion erhalten muss, bevor er fortfährt.

- **`Event`** – Dieser Aufrufstyp ist asynchron. Die Antwort umfasst keine Nutzlast, die Ergebnisse enthält. Verwenden Sie diesen Aufruftyp, wenn Ihr Workflow kein Ergebnis der Lambda-Funktion benötigt, um die Verarbeitung fortzusetzen.

Als einfacher Test Ihres Setups können Sie mittels `psql` eine Verbindung mit Ihrer DB-Instance herstellen und eine Beispielfunktion über die Befehlszeile aufrufen. Angenommen, Sie haben eine der Grundfunktionen Ihres Lambda-Services eingerichtet, z. B. die einfache Python-Funktion, die im folgenden Screenshot gezeigt wird.



Beispielfunktion aufrufen

1. Stellen Sie per `psql` oder pgAdmin eine Verbindung zu Ihrer primären DB-Instance her.

   ```
   psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
   ```

2. Rufen Sie die Funktion mit ihrem ARN auf.

   ```
   SELECT * from
    aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
   region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
    Postgres!"}'::json );
   ```

   Die Antwort sieht wie folgt aus.

   ```
   status_code |                          payload                          |
    executed_version | log_result
   ```

```
-------------+------------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
     |
(1 row)
```

Schlägt der Aufruf fehl, siehe Fehlermeldungen von Lambda-Funktionen .

Schritt 6: Erteilen der Berechtigung, Lambda-Funktionen aufzurufen, für andere Benutzer

An dieser Stelle in den Prozeduren können nur Sie als `rds_superuser` Ihre Lambda-Funktionen aufrufen. Damit andere Benutzer alle von Ihnen erstellten Funktionen aufrufen können, müssen Sie ihnen die entsprechende Berechtigung erteilen.

Gewähren der Berechtigung zum Aufrufen von Lambda-Funktionen

1. Stellen Sie per `psql` oder pgAdmin eine Verbindung zu Ihrer primären DB-Instance her.

   ```
   psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
   ```

2. Führen Sie die folgenden SQL-Befehle aus:

   ```
   postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
   GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
   ```

Beispiele: Aufrufen von Lambda-Funktionen von Ihrem Aurora-PostgreSQL-DB-Cluster

Nachfolgend finden Sie einige Beispiele für den Aufruf der aws_lambda.invoke-Funktion. Die meisten Beispiele verwenden die Verbundstruktur `aws_lambda_arn_1`, die Sie in Schritt 4: Verwenden von Lambda-Hilfsfunktionen mit Ihrem Aurora-PostgreSQL-DB-Cluster (optional) erstellen, um die Übergabe der Funktionsdetails zu vereinfachen. Ein Beispiel für einen asynchronen Aufruf finden Sie unter Beispiel: Asynchroner Ereignisaufruf (Event) von Lambda-Funktionen. Alle anderen aufgelisteten Beispiele verwenden einen synchronen Aufruf.

Weitere Informationen zu Lambda-Aufruftypen finden Sie unter Aufrufen von Lambda-Funktionen im AWS Lambda-Entwicklerhandbuch. Mehr über `aws_lambda_arn_1` erfahren Sie unter aws_commons.create_lambda_function_arn.

Liste mit Beispielen

- Beispiel: Synchroner Aufruf (RequestResponse) von Lambda-Funktionen
- Beispiel: Asynchroner Ereignisaufruf (Event) von Lambda-Funktionen
- Beispiel: Erfassen des Lambda-Ausführungsprotokolls in einer Funktionsantwort
- Beispiel: Einschließen von Client-Kontext in einer Lambda-Funktion
- Beispiel: Aufrufen einer bestimmten Version einer Lambda-Funktion

Beispiel: Synchroner Aufruf (RequestResponse) von Lambda-Funktionen

Es folgen zwei Beispiele für einen synchronen Lambda-Funktionsaufruf. Die Ergebnisse dieser `aws_lambda.invoke`-Funktionen sind identisch.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

Die Parameter werden wie folgt beschrieben:

- `:'aws_lambda_arn_1'` – Dieser Parameter identifiziert die zusammengesetzte Struktur, die in Schritt 4: Verwenden von Lambda-Hilfsfunktionen mit Ihrem Aurora-PostgreSQL-DB-Cluster (optional) erstellt wurde, mit der `aws_commons.create_lambda_function_arn`-Hilfsfunktion. Sie können diese Struktur wie folgt auch in Ihren `aws_lambda.invoke`-Aufruf einbinden.

  ```
  SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
   'aws-region'),
  '{"body": "Hello from Postgres!"}'::json
  );
  ```

- `'{"body": "Hello from PostgreSQL!"}'::json` – Die JSON-Nutzlast, die an die Lambda-Funktion übergeben werden soll.
- `'RequestResponse'` – Der Lambda Aufruftyp.

Beispiel: Asynchroner Ereignisaufruf (Event) von Lambda-Funktionen

Es folgt ein Beispiel für einen asynchronen Lambda-Funktionsaufruf. Der `Event`-Aufruftyp plant den Lambda-Funktionsaufruf mit der angegebenen Eingabe-Nutzlast und sofort Rückgabe. Verwenden Sie den `Event`-Aufruftyp in bestimmten Workflows, die nicht von den Ergebnissen der Lambda-Funktion abhängen.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

Beispiel: Erfassen des Lambda-Ausführungsprotokolls in einer Funktionsantwort

Sie können die letzten 4 KB des Ausführungsprotokolls in die Funktionsantwort aufnehmen, indem Sie den Parameter `log_type` in Ihrem `aws_lambda.invoke`-Funktionsaufruf verwenden. Standardmäßig ist dieser Parameter auf `None` festgelegt, aber Sie können `Tail` angeben, um die Ergebnisse des Lambda-Ausführungsprotokolls in der Antwort zu erfassen, wie nachfolgend gezeigt.

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

Legen Sie den Parameter aws_lambda.invoke der `log_type`-Funktion auf `Tail`, um das Ausführungsprotokoll in die Antwort aufzunehmen. Der Standardwert für diesen `log_type`-Parameter ist `None`.

Das `log_result`, was zurückgegeben wird, ist eine `base64` codierte Zeichenfolge. Sie können den Inhalt mit einer Kombination der PostgreSQL-Funktionen `decode` und `convert_from` dekodieren.

Mehr über `log_type` erfahren Sie unter aws_lambda.invoke.

Beispiel: Einschließen von Client-Kontext in einer Lambda-Funktion

Die `aws_lambda.invoke`-Funktion hat einen `context`-Parameter, den Sie verwenden können, um Informationen getrennt von der Nutzlast zu übergeben, wie nachfolgend gezeigt.

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

Um den Clientkontext einzuschließen, verwenden Sie ein JSON-Objekt für den Parameter aws_lambda.invoke der `context`-Funktion.

Weitere Informationen zum `context`-Parameter finden Sie in der aws_lambda.invoke-Referenz.

Beispiel: Aufrufen einer bestimmten Version einer Lambda-Funktion

Sie können eine bestimmte Version einer Lambda-Funktion angeben, indem Sie den `qualifier`-Parameter mit dem `aws_lambda.invoke`-Aufruf einschließen. Im Folgenden finden Sie ein Beispiel, das diese Funktion mit '*custom_version*' als Alias für die Version erfüllt.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

Sie können auch einen Lambda-Funktionsqualifizierer wie folgt mit den Daten zum Funktionsnamen angeben.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

Weitere Informationen zu `qualifier` und anderen Parametern finden Sie in der aws_lambda.invoke-Referenz.

Fehlermeldungen von Lambda-Funktionen

In der folgenden Liste finden Sie Informationen zu Fehlermeldungen sowie mögliche Ursachen und Lösungen.

- VPC-Konfigurationsprobleme

  Probleme mit der VPC-Konfiguration können beim Versuch, eine Verbindung herzustellen, die folgenden Fehlermeldungen auslösen:

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  Eine häufige Ursache für diesen Fehler ist eine falsch konfigurierte VPC-Sicherheitsgruppe. Stellen Sie sicher, dass eine Regel Port 443 Ihrer VPC-Sicherheitsgruppe für ausgehenden TCP-Datenverkehr öffnet, damit Ihre VPC eine Verbindung zur Lambda-VPC herstellen kann.
- Fehlende Berechtigungen, die zum Aufrufen von Lambda-Funktionen erforderlich sind

Wenn eine der folgenden Fehlermeldungen angezeigt wird, verfügt der Benutzer (Rolle), der die Funktion aufruft, nicht über die entsprechenden Berechtigungen.

```
ERROR:  permission denied for schema aws_lambda
```

```
ERROR:  permission denied for function invoke
```

Ein Benutzer (Rolle) muss bestimmte Berechtigungen erhalten, um Lambda-Funktionen aufrufen zu können. Weitere Informationen finden Sie unter Schritt 6: Erteilen der Berechtigung, Lambda-Funktionen aufzurufen, für andere Benutzer.

- Unsachgemäße Handhabung von Fehlern in Ihren Lambda-Funktionen

  Wenn eine Lambda-Funktion während der Anforderungsverarbeitung eine Ausnahme auslöst, `aws_lambda.invoke` schlägt dies mit einem PostgreSQL-Fehler wie folgt fehl.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

  Kümmern Sie sich unbedingt um Fehler in Ihren Lambda-Funktionen oder in Ihrer PostgreSQL-Anwendung.

Referenz für AWS Lambda-Funktionen und -Parameter

Im Folgenden finden Sie die Referenz für die Funktionen, die zum Aufrufen von Lambda-Funktionen mit Aurora PostgreSQL verwendet werden müssen.

Funktionen und Parameter

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- aws_lambda-Parameter

aws_lambda.invoke

Führt eine Lambda-Funktion für eine Aurora PostgreSQL DB-Cluster-RDS aus.

Weitere Informationen zum Aufrufen von Lambda-Funktionen finden Sie unter Invoke (Aufrufen) auch im AWS Lambda-Entwicklerhandbuch.

Syntax

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
```

```
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

Eingabeparameter

function_name

Der spezifizierte Name der Lambda-Funktion. Der Wert kann der Funktionsname, ein ARN oder ein partieller ARN sein. Eine Auflistung möglicher Formate finden Sie unter Lambda Funktionsnamenformate im AWS Lambda-Entwicklerhandbuch.

Nutzlast

Die Eingabe für die Funktion Lambda. Das Format kann JSON oder JSONB sein. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation zu JSON Types.

region

(Optional) Die Lambda-Region für die Funktion. Standardmäßig verwendet Aurora die AWS-Region aus dem vollständigen ARN in der `function_name` oder die Aurora PostgreSQL -DB-Instance-Region. Wenn dieser Region-Wert mit dem im `function_name` ARN angegebenen Wert in Konflikt steht, wird ein Fehler ausgelöst.

invocation_type

Die Aufruftyp der Lambda-Funktion. Bei -Wert ist die Groß- und Kleinschreibung zu beachten. Die folgenden Werte sind möglich:

- `RequestResponse` – Der Standardwert. Diese Art des Aufrufens für eine Lambda-Funktion ist synchron und gibt eine Antwortnutzlast im Ergebnis zurück. Verwenden Sie den `RequestResponse` Aufruftyp, wenn Ihr Workflow vom sofortigen Erhalt des Lambda-Funktionsergebnisses abhängt.
- `Event` – Diese Art des Aufrufs für eine Lambda-Funktion ist asynchron und wird sofort ohne Rückgabe einer Nutzlast zurückgegeben. Verwenden Sie den `Event`-Aufruftyp, wenn Sie keine Ergebnisse der Lambda-Funktion benötigen, bevor Ihr Workflow weitergeht.
- `DryRun` – Diese Art des Aufrufs testet den Zugriff, ohne die Lambda-Funktion auszuführen.

log_typ

Der Typ des Lambda-Protokolls, das im Ausgabeparameter `log_result` ausgegeben werden soll. Bei -Wert ist die Groß- und Kleinschreibung zu beachten. Die folgenden Werte sind möglich:

- Tail – Der zurückgegebene Ausgabeparameter `log_result` enthält die letzten 4 KB des Ausführungsprotokolls.
- Keiner – Es werden keine Lambda-Protokollinformationen zurückgegeben.

context

Client-Kontext im JSON- oder JSONB-Format. Zu verwendende Felder sind dann `custom` und `env`.

Qualifier

Ein Qualifier, der die aufzurufende Version einer Lambda-Funktion spezifiziert. Wenn dieser Wert mit einem im `function_name` ARN angegebenen Wert in Konflikt steht, wird ein Fehler ausgelöst.

Ausgabeparameter

status_code

Ein HTTP-Status-Antwortcode. Weitere Informationen finden Sie unter Lambda Antwortelemente aufrufen im AWS Lambda-Entwicklerhandbuch.

Nutzlast

Die von der ausgeführten Lambda-Funktion zurückgegebenen Daten. Das Format ist in JSON oder JSONB.

executed_version

Die Version der Lambda-Funktion, die ausgeführt wurde.

log_resultat

Die Ausführungsprotokollinformationen werden zurückgegeben, wenn der Wert `log_type` beim Aufruf der Lambda-Funktion `Tail` beträgt. Das Ergebnis enthält die letzten 4 KB des in Base64 codierten Ausführungsprotokolls.

aws_commons.create_lambda_function_arn

Erstellt eine `aws_commons._lambda_function_arn_1`-Struktur für Daten zum Lambda Funktionsnamen. Sie können die Ergebnisse der `aws_commons.create_lambda_function_arn`-Funktion im Parameter `function_name` der aws_lambda.invoke-Funktion aws_lambda.invoke verwenden.

Syntax

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

Eingabeparameter

function_name

Eine erforderliche Textzeichenfolge mit dem Lambda-Funktionsnamen. Der Wert kann ein Funktionsname, ein partieller ARN oder ein vollständiger ARN sein.

region

Eine optionale Textzeichenfolge mit der AWS-Region, in der sich die Lambda-Funktion befindet. Eine Liste der -Regionsnamen und der zugehörigen Werte finden Sie unter Regionen und Availability Zones.

aws_lambda-Parameter

In der Tabelle finden Sie Parameter, die der `aws_lambda`-Funktion zugeordnet sind.

Parameter	Beschreibung
`aws_lambda.connect_timeout_ms`	Dies ist ein dynamischer Parameter, der die maximale Wartezeit beim Herstellen einer Verbindung zu AWS Lambda festlegt. Der Standardwert ist `1000`. Die zulässigen Werte für diesen Parameter sind 1 bis 90000.
`aws_lambda.request_timeout_ms`	Dies ist ein dynamischer Parameter, der die maximale Wartezeit beim Warten auf eine Antwort von AWS Lambda festlegt. Der Standardwert ist `3000`. Die zulässigen Werte für diesen Parameter sind 1 bis 90000.
`aws_lambda.endpoint_override`	Gibt den Endpunkt an, der für die Verbindung mit AWS Lambda verwendet werden kann. Eine leere Zeichenfolge wählt den AWS-Lambda-Standardendpunkt für die Region aus. Sie müssen die Datenbank für diese Änderung des statischen Parameters neu starten, damit Änderungen wirksam werden.

Veröffentlichen von Aurora-PostgreSQL-Protokollen in Amazon CloudWatch Logs

Sie können Ihren Aurora-PostgreSQL-DB-Cluster so konfigurieren, dass Protokolldaten regelmäßig in Amazon CloudWatch Logs exportiert werden. Wenn Sie dies tun, werden Ereignisse aus dem PostgreSQL-Protokoll Ihres Aurora-PostgreSQL-DB-Clusters automatisch als Amazon CloudWatch Logs auf Amazon CloudWatch veröffentlicht. In CloudWatch finden Sie die exportierten Protokolldaten in einer Protokollgruppe für Ihren Aurora-PostgreSQL-DB-Cluster. Die Protokollgruppe enthält einen oder mehrere Protokollstreams, die die Ereignisse aus dem PostgreSQL-Protokoll von jeder Instance im Cluster enthalten.

Durch die Veröffentlichung der Protokolle in CloudWatch Logs können Sie die PostgreSQL-Protokolldatensätze in einem Speicher mit hoher Beständigkeit aufbewahren. Mit den in CloudWatch Logs verfügbaren Protokolldaten können Sie den Betrieb Ihres Clusters auswerten und verbessern.

Sie können CloudWatch also verwenden, um Alarme zu erstellen und Metriken anzuzeigen. Weitere Informationen hierzu finden Sie unter Überwachung von Protokollereignissen in Amazon CloudWatch.

> **Note**
>
> Das Veröffentlichen Ihrer PostgreSQL-Protokolle in CloudWatch Logs verbraucht Speicherplatz, und für diesen Speicher fallen Gebühren an. Löschen Sie CloudWatch Logs, die nicht mehr benötigt werden.

Das Deaktivieren der Option zum Exportieren von Protokollen für einen vorhandenen Aurora PostgreSQL DB-Cluster wirkt sich nicht auf Daten aus, die bereits in CloudWatch Logs gespeichert sind. Bestehende Protokolle bleiben in CloudWatch Logs verfügbar, basierend auf Ihren Protokollaufbewahrungseinstellungen. Weitere Informationen zu CloudWatch Logs finden Sie unter Was ist CloudWatch Logs?

Aurora PostgreSQL unterstützt die Veröffentlichung von Protokollen in CloudWatch Logs für die folgenden Versionen.

- 16.1 und alle höheren Versionen
- 15.2 und höhere 15-Versionen
- 14.3 und höhere 14-Versionen
- 13.3 und höhere 13 Versionen
- 12.8 und höhere 12 Versionen
- 11.9 und höher 11 Versionen

Informationen zum Aktivieren der Option zum Veröffentlichen von Protokollen in CloudWatch Logs, zum Überwachen von Protokollereignissen in CloudWatch Logs und zum Analysieren von Protokollen mit CloudWatch Logs Insights finden Sie in den folgenden Themen.

Themen

- Aktivieren der Option zum Veröffentlichen von Protokollen in Amazon CloudWatch
- Überwachung von Protokollereignissen in Amazon CloudWatch
- Analysieren von Aurora-PostgreSQL-Protokollen mit CloudWatch Logs Insights

Aktivieren der Option zum Veröffentlichen von Protokollen in Amazon CloudWatch

Um das PostgreSQL-DB-Cluster in CloudWatch Logs zu veröffentlichen, wählen Sie die Option Log export (Protokollexport) für den Cluster. Sie können die Log-Export-Einstellung beim Erstellen Ihres Aurora-PostgreSQL-DB-Clusters auswählen. Sie können den Cluster auch später ändern. Wenn Sie einen vorhandenen Cluster ändern, werden seine PostgreSQL-Protokolle von jeder Instance ab diesem Zeitpunkt im CloudWatch-Cluster veröffentlicht. Für Aurora PostgreSQL ist das PostgreSQL-Protokoll (`postgresql.log`) das einzige Protokoll, das auf Amazon CloudWatch veröffentlicht wird.

Sie können das AWS-Managementkonsole, das AWS CLI oder die RDS-API verwenden, um die Protokollexportfunktion für Ihren Aurora PostgreSQL DB-Cluster zu aktivieren.

Konsole

Wählen Sie die Option Protokollexporte, um mit der Veröffentlichung der PostgreSQL-Protokolle aus Ihrem Aurora PostgreSQL DB-Cluster in CloudWatch Logs zu beginnen.

So aktivieren Sie die Funktion zum Exportieren von Protokollen über die Konsole

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Aurora-PostgreSQL-DB-Cluster, für den die Protokolldaten in CloudWatch Logs veröffentlicht werden sollen.
4. Wählen Sie Ändern aus.
5. Wählen Sie im Abschnitt Log exports (Protokollexporte) die Option PostgreSQL log aus.
6. Wählen Sie Continue (Weiter) und dann auf der Übersichtsseite Modify cluster (Cluster ändern).

AWS CLI

Sie können die Option zum Exportieren von Protokollen aktivieren, um die Veröffentlichung von Aurora-PostgreSQL-Protokollen in Amazon CloudWatch Logs mit AWS CLI zu starten. Hierzu führen Sie den Befehl modify-db-cluster AWS CLI mit den folgenden Optionen aus:

- `--db-cluster-identifier` – Die DB-Cluster-Kennung.
- `--cloudwatch-logs-export-configuration` – Die Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster eingestellt werden sollen.

Sie können Aurora-PostgreSQL-Protokolle auch veröffentlichen, indem Sie einen der folgenden AWS CLI-Befehle ausführen:

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Führen Sie einen dieser AWS CLI-Befehle mit den folgenden Optionen aus:

- `--db-cluster-identifier` – Die DB-Cluster-Kennung.
- `--engine`— Die Datenbank-Engine.
- `--enable-cloudwatch-logs-exports`— Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster aktiviert werden sollen.

Je nach ausgeführtem AWS CLI-Befehl müssen möglicherweise noch weitere Optionen angegeben werden.

Example

Der folgende Befehl erstellt einen Aurora-PostgreSQL-DB-Cluster zum Veröffentlichen von Protokolldateien in CloudWatch Logs.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --engine aurora-postgresql \
    --enable-cloudwatch-logs-exports postgresql
```

Für Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --engine aurora-postgresql ^
    --enable-cloudwatch-logs-exports postgresql
```

Example

Der folgende Befehl modifiziert einen vorhandenen Aurora-PostgreSQL-DB-Cluster zum Veröffentlichen von Protokolldateien in CloudWatch Logs. Der `--cloudwatch-logs-export-configuration`-Wert ist ein JSON-Objekt. Der Schlüssel für dieses Objekt ist `EnableLogTypes` und sein Wert lautet `postgresql` und `instance`.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["postgresql","instance"]}'
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\",
\"instance\"]}'
```

> **Note**
>
> Wenn Sie die Windows-Befehlszeile verwenden, stellen Sie sicher, dass Sie doppelte Anführungszeichen (") im JSON-Code maskieren, indem Sie ihnen einen umgekehrten Schrägstrich (\) voranstellen.

Example

Das folgende Beispiel modifiziert einen vorhandenen Aurora-PostgreSQL-DB-Cluster zum Deaktivieren der Veröffentlichung von Protokolldateien in CloudWatch Logs. Der `--cloudwatch-logs-export-configuration`-Wert ist ein JSON-Objekt. Der Schlüssel für dieses Objekt ist `DisableLogTypes` und sein Wert lautet `postgresql` und `instance`.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
```

```
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":
["postgresql","instance"]}'
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\",
\"instance\"]}"
```

> **Note**
>
> Bei Verwendung der Windows-Befehlszeile müssen doppelte Anführungszeichen (") im JSON-Code mit einem umgekehrten Schrägstrich (\) als Escape-Zeichen versehen werden.

RDS-API

Sie können die Log-Export-Option aktivieren, um die Veröffentlichung von Aurora-PostgreSQL-Protokollen mit der RDS-API zu starten. Hierzu führen Sie den Vorgang ModifyDBCluster mit den folgenden Optionen aus:

- `DBClusterIdentifier` – Die DB-Cluster-ID.
- `CloudwatchLogsExportConfiguration` – Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster aktiviert werden sollen.

Sie können Aurora PostgreSQL-Protokolle auch mit der RDS-API veröffentlichen, indem Sie eine der folgenden RDS-API-Operationen ausführen:

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

Führen Sie die RDS-API-Operation mit den folgenden Parametern aus:

- `DBClusterIdentifier` – Die DB-Cluster-Kennung.

- `Engine`— Die Datenbank-Engine.
- `EnableCloudwatchLogsExports`— Konfigurationseinstellung für die Protokolltypen, die für den Export in CloudWatch Logs für den DB-Cluster aktiviert werden sollen.

Je nach ausgeführtem AWS CLI-Befehl müssen möglicherweise noch weitere Parameter angegeben werden.

Überwachung von Protokollereignissen in Amazon CloudWatch

Wenn Aurora-PostgreSQL-Protokollereignisse veröffentlicht und als Amazon CloudWatch Logs verfügbar sind, können Sie Ereignisse mit Amazon CloudWatch anzeigen und überwachen. Weitere Informationen zur Überwachung finden Sie unter Anzeige von an CloudWatch Logs gesendeten Protokolldaten.

Wenn Sie Log exports (Protokollexporte) aktivieren, wird mit dem Präfix `/aws/rds/cluster/` automatisch eine neue Protokollgruppe mit dem Namen Ihrer Aurora PostgreSQL und dem Protokolltyp, wie im folgenden Muster erstellt.

```
/aws/rds/cluster/your-cluster-name/postgresql
```

Angenommen, ein Aurora-PostgreSQL-DB-Cluster mit dem Namen `docs-lab-apg-small` exportiert sein Protokoll zu Amazon CloudWatch Logs. Der Name der Protokollgruppe in Amazon CloudWatch wird im Folgenden angezeigt.

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

Wenn bereits eine Protokollgruppe mit dem angegebenen Namen existiert, verwendet Aurora diese Protokollgruppe, um Protokolldaten für den Aurora-DB-Cluster zu exportieren. Jede DB-Instance im Aurora PostgreSQL DB-Cluster lädt ihr PostgreSQL-Protokoll als eigenständigen Protokollstream in die Protokollgruppe hoch. Sie können die Protokollgruppe und ihre Protokollströme mithilfe der verschiedenen grafischen und analytischen Tools untersuchen, die in Amazon CloudWatch verfügbar sind.

Sie können beispielsweise Informationen in Protokollereignissen von Ihrem Aurora-PostgreSQL-DB-Cluster suchen und Ereignisse mit der CloudWatch Logs-Konsole, AWS CLI oder der CloudWatch Logs API filtern. Weitere Informationen finden Sie unter Suchen und Filtern von Protokolldaten im Amazon CloudWatch Logs-Benutzerhandbuch.

Standardmäßig werden neue Protokollgruppen mit Läuft niemals ab für ihre Aufbewahrungszeit erstellt. Sie können die CloudWatch Logs-Konsole, die AWS CLI oder die CloudWatch-Logs-API verwenden, um die Protokollaufbewahrungszeit zu ändern. Weitere Informationen finden Sie unter Ändern der Protokoll-Datenaufbewahrung in CloudWatch Logs im Amazon CloudWatch Logs-Benutzerhandbuch.

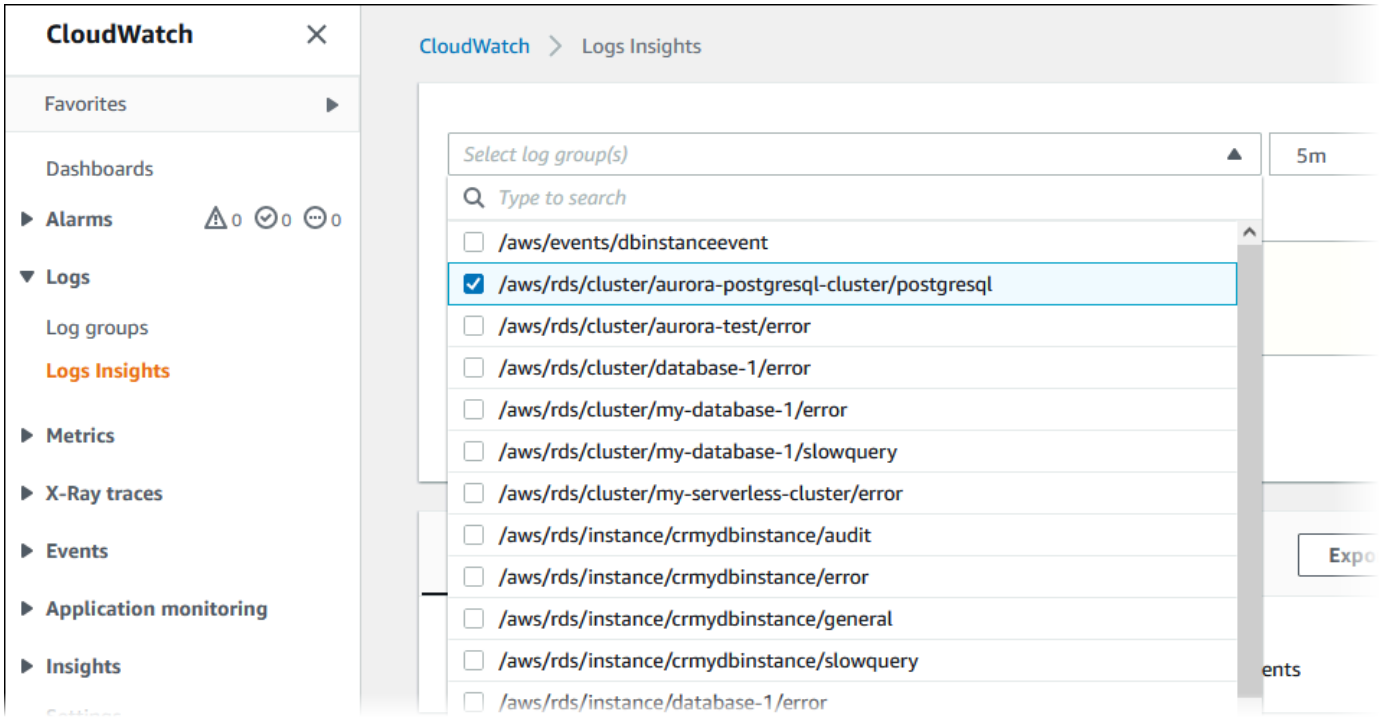
> **Tip**
>
> Sie können die automatisierte Konfiguration, wie z. B. AWS CloudFormation, verwenden, um Protokollgruppen mit vordefinierten Protokollaufbewahrungszeiten, Metrikfiltern und Zugriffsberechtigung zu erstellen.

Analysieren von Aurora-PostgreSQL-Protokollen mit CloudWatch Logs Insights

Wenn die PostgreSQL-Protokolle aus Ihrem Aurora-PostgreSQL-DB-Cluster als CloudWatch Logs veröffentlicht wurden, können Sie CloudWatch Logs Insights verwenden, um interaktiv Ihre Protokolldaten in Amazon CloudWatch Logs zu suchen und zu analysieren. CloudWatch Logs Insights umfasst eine Abfragesprache, Beispielabfragen und andere Tools zur Analyse Ihrer Protokolldaten, damit Sie potenzielle Probleme identifizieren und Korrekturen überprüfen können. Weitere Informationen finden Sie unter Analyse von Protokolldaten mit CloudWatch Logs Insights im Amazon CloudWatch Logs-Benutzerhandbuch.

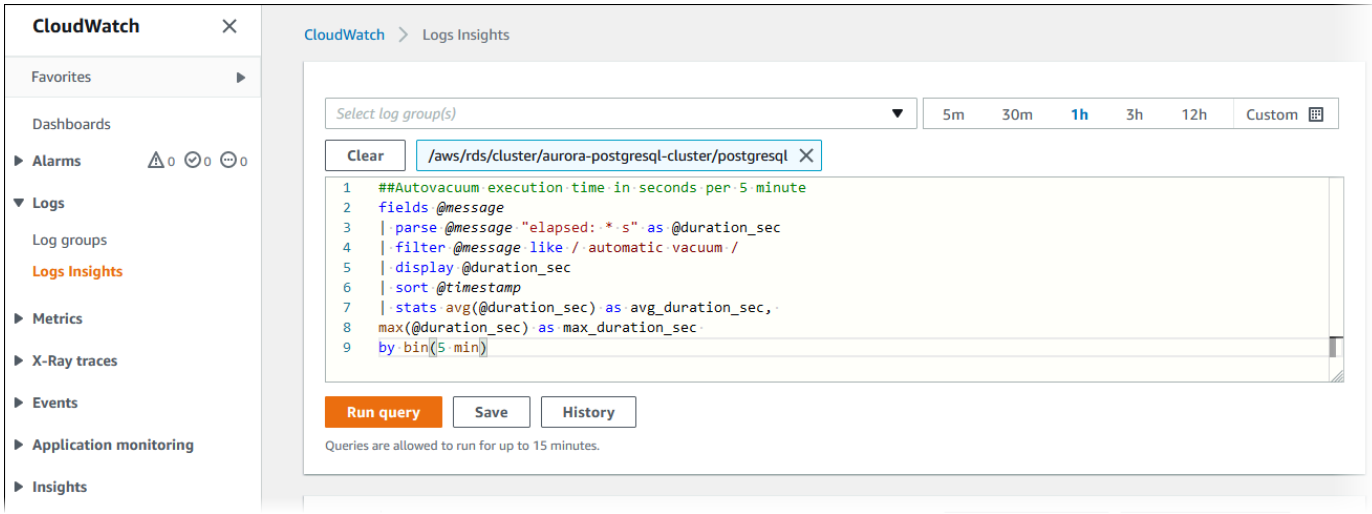
So analysieren Sie PostgreSQL-Protokolle mit CloudWatch Logs Insights

1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Öffnen Sie im Navigationsbereich Logs (Protokolle) und wählen Sie Log insights (Protokoll-Erkenntnisse) aus.
3. Wählen Sie unter Select log group(s) (Protokollgruppe(n) auswählen) die Protokollgruppe für Ihren Aurora-PostgreSQL-DB-Cluster aus.

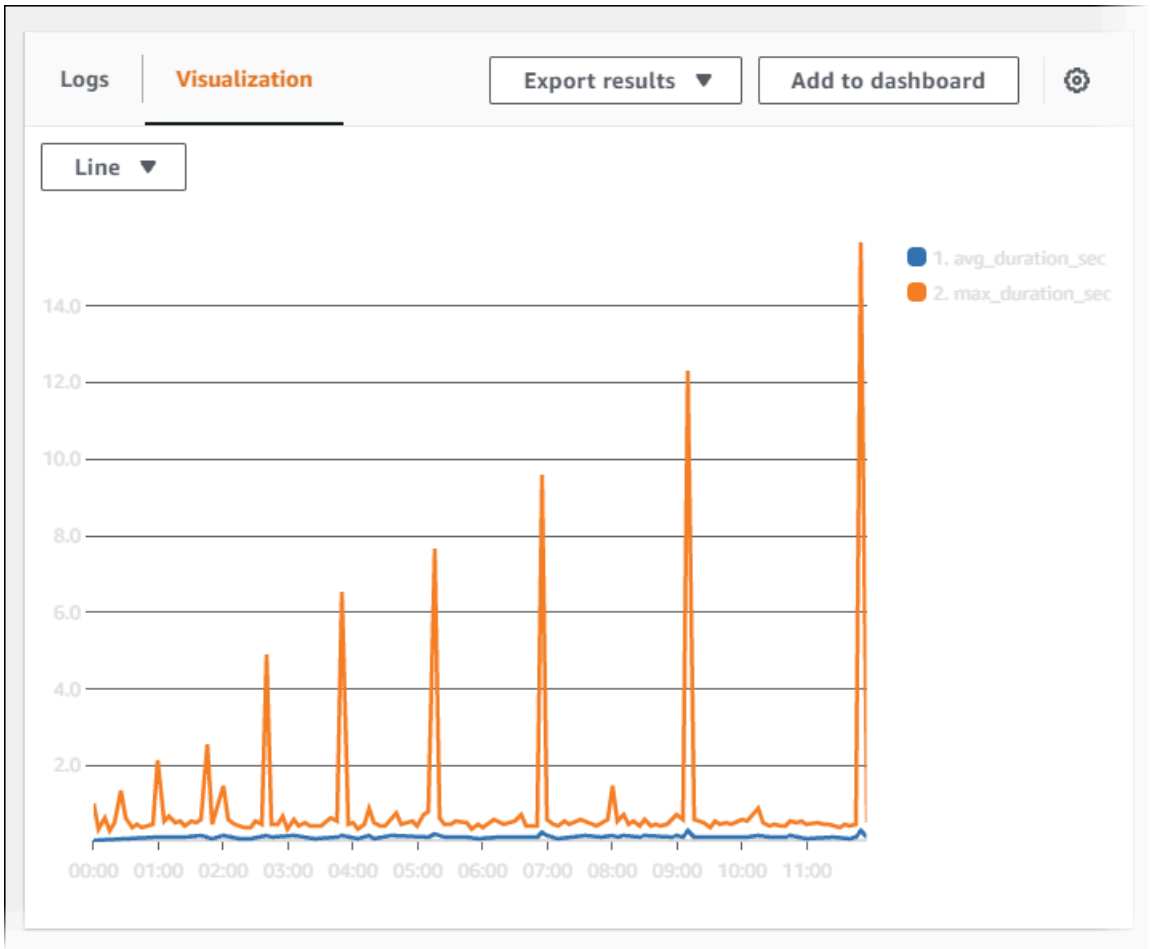


4. Löschen Sie im Abfrage-Editor die aktuell angezeigte Abfrage und geben Sie Folgendes ein. Wählen Sie anschließend Run query (Abfrage ausführen) aus.

```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
| display @duration_sec
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```

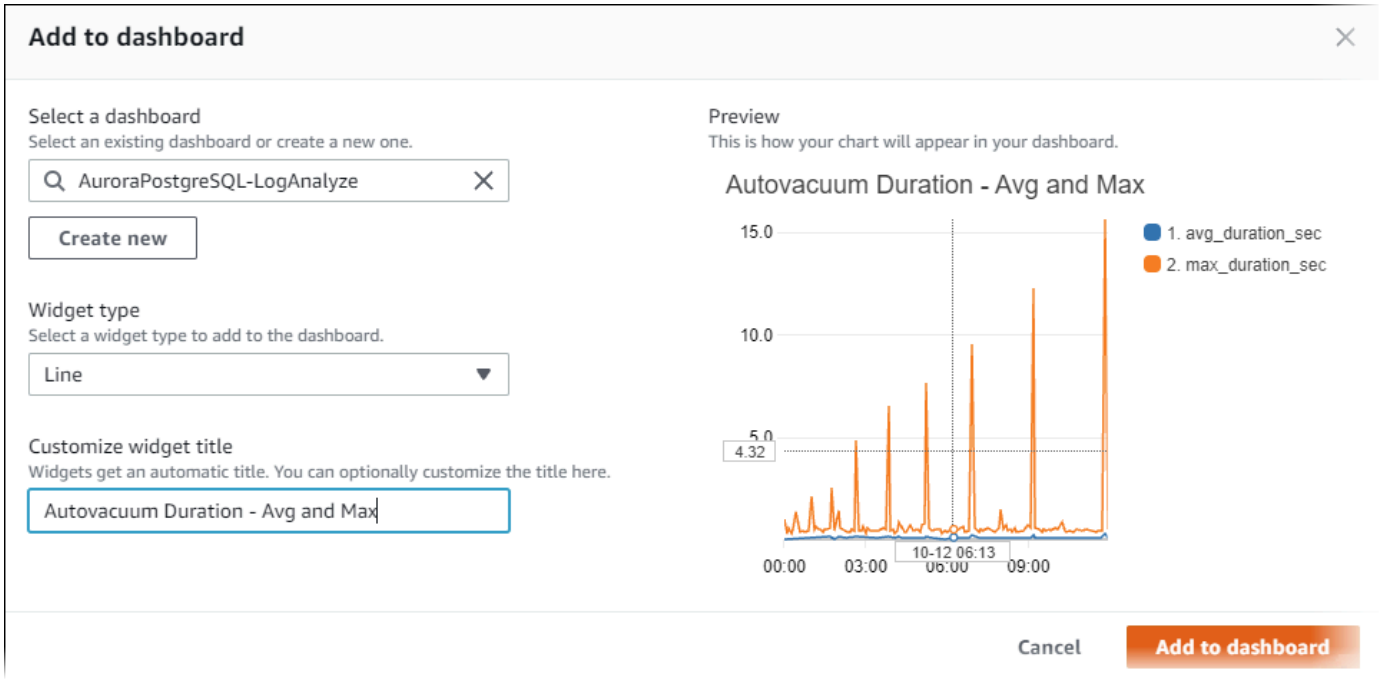


5. Wählen Sie die Registerkarte Visualization (Visualisierung) aus.



6. Wählen Sie Add to dashboard (Zu Dashboard hinzufügen) aus.
7. Wählen Sie unter Dashboard auswählen entweder ein Dashboard aus oder geben Sie einen Namen ein, um ein neues Dashboard zu erstellen.

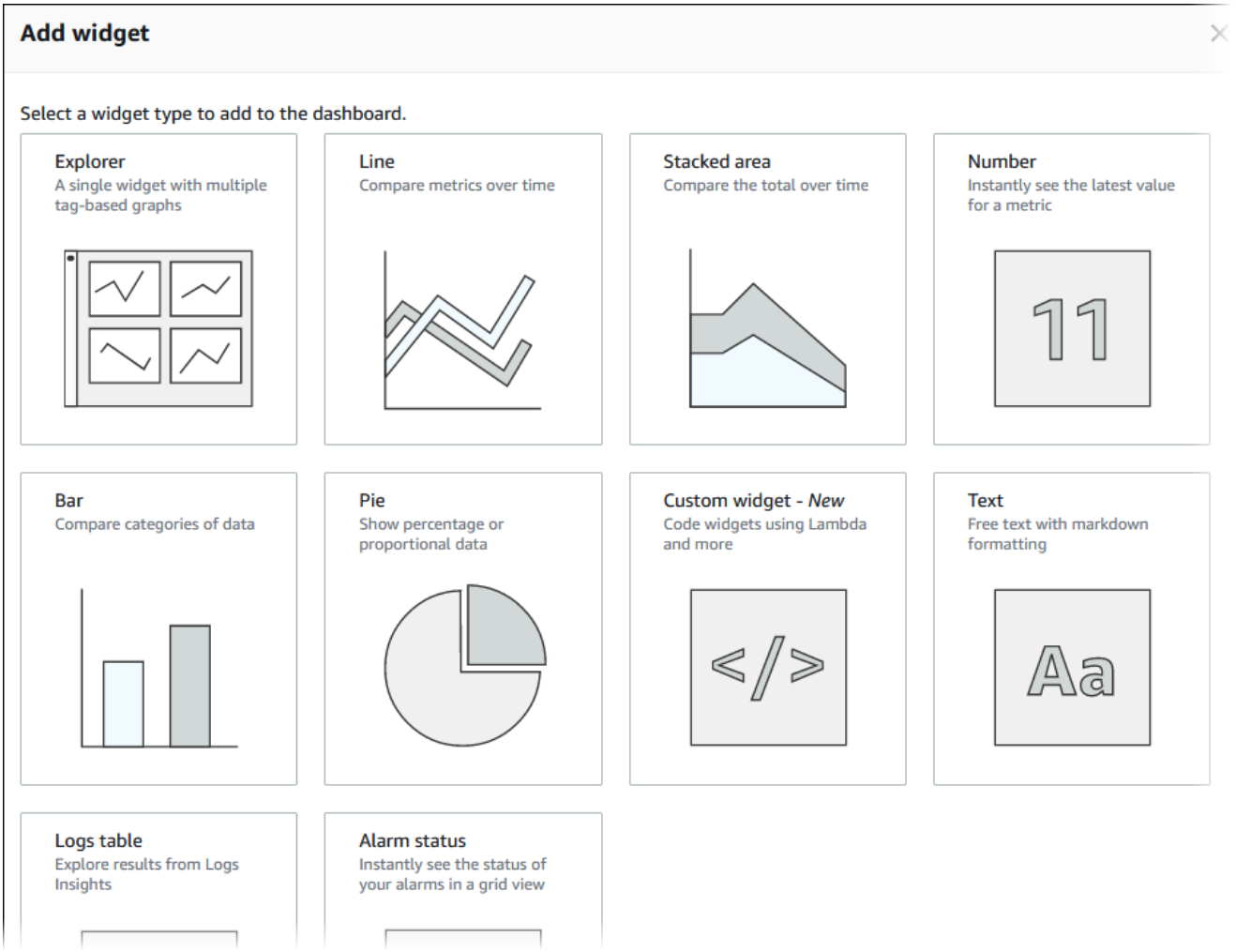
8. Wählen Sie unter Widget-Typ einen Widget-Typ für Ihre Visualisierung aus.



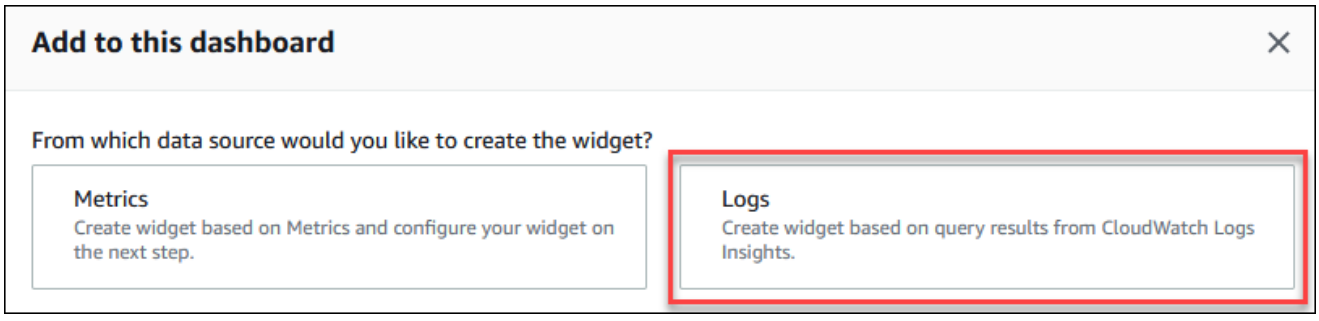
9. (Optional) Fügen Sie basierend auf Ihren Protokollabfrage-Ergebnissen weitere Widgets hinzu.

   a. Wählen Sie Add widget aus.

   b. Wählen Sie einen Widget-Typ aus, z. B. Linie.



c. Wählen Sie im Fenster Zu diesem Dashboard hinzufügen die Option Protokolle aus.

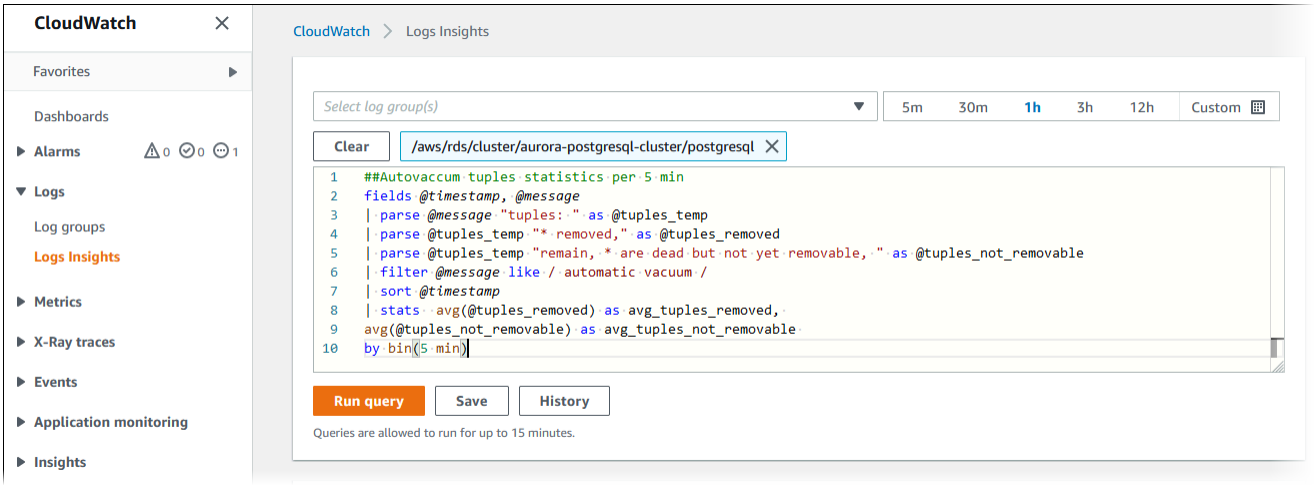


d. Wählen Sie unter Select log group(s) die Protokollgruppe für Ihren DB-Cluster aus.

e. Löschen Sie im Abfrage-Editor die aktuell angezeigte Abfrage und geben Sie Folgendes ein. Wählen Sie anschließend Run query aus.

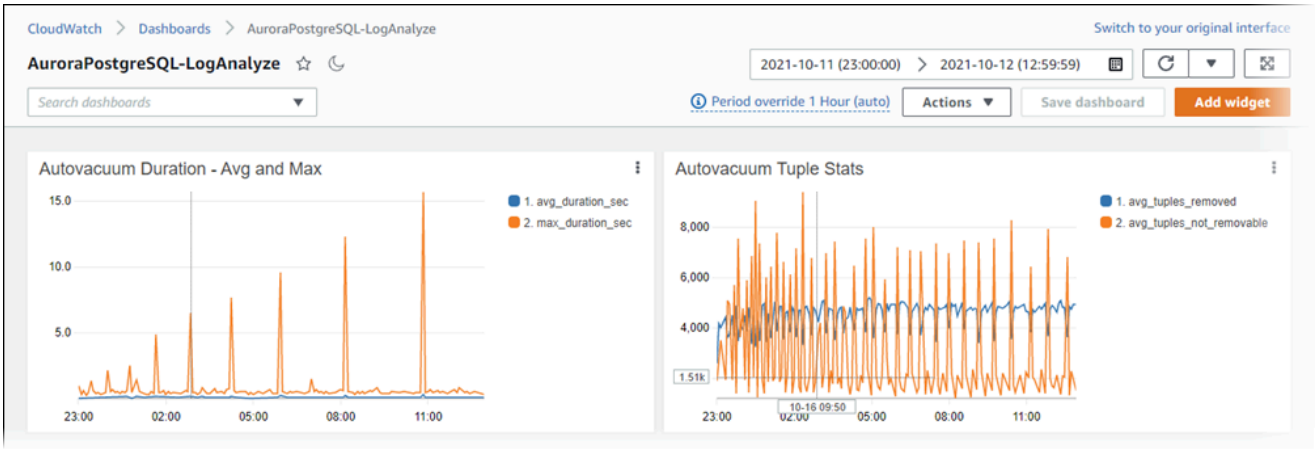
```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
```

```
| parse @tuples_temp "* removed," as @tuples_removed
| parse @tuples_temp "remain, * are dead but not yet removable, " as
 @tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats  avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```



f. Wählen Sie Create widget aus.

Ihr Dashboard sollte folgender Abbildung ähnlich sein.



Überwachen der Ausführungspläne von Abfragen und deren maximale Speichernutzung für Aurora PostgreSQL

Sie können Ausführungspläne von Abfragen in Ihrer DB-Instance von Aurora PostgreSQL überwachen, um die Ausführungspläne zu ermitteln, die zur aktuellen Datenbanklast beitragen, und um mithilfe des Parameters `aurora_compute_plan_id` Leistungsstatistiken der Ausführungspläne im Laufe der Zeit zu verfolgen. Jedes Mal, wenn eine Abfrage ausgeführt wird, erhält der von der Abfrage verwendete Ausführungsplan eine ID, und dieselbe ID wird bei späteren Ausführungen desselben Plans erneut verwendet.

Die `aurora_compute_plan_id` ist in der DB-Parametergruppe von Aurora-PostgreSQL-Versionen 14.10, 15.5 und höheren Versionen standardmäßig `OFF`. Zum Zuweisen einer Plan-ID legen Sie die `aurora_compute_plan_id` in der Parametergruppe auf `ON` fest.

Diese Plan-ID wird in mehreren Dienstprogrammen verwendet, die einem anderen Zweck dienen.

Sie können die maximale Speichernutzung von Abfragen in Ihrer DB-Instance überwachen, um Abfragen zu erkennen, die zu einer hohen Datenbankspeichernutzung beitragen – ab den folgenden Versionen verfügbar:

- 16.3 und alle höheren Versionen
- 15.7 und höhere Versionen
- 14.12 und höhere Versionen

Immer wenn eine Abfrage ausgeführt wird, wird die maximale Speichernutzung der Abfrage nachverfolgt. Abfragen werden in der Regel viele Male ausgeführt. Die Werte für die durchschnittliche, minimale und maximale Speichernutzung aller Durchläufe können für jede Abfrage eingesehen werden.

Themen

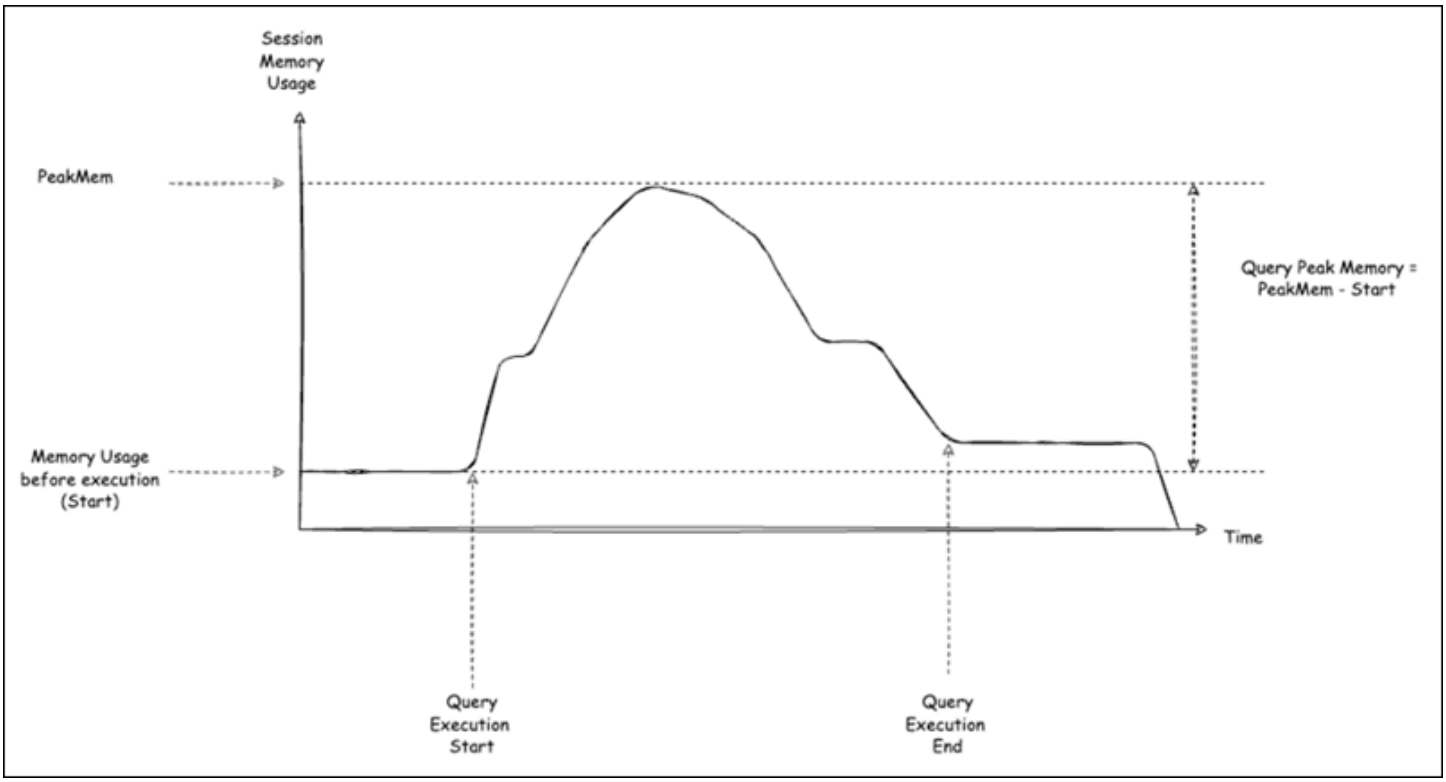
- Zugreifen auf Ausführungspläne von Abfragen und deren maximale Speichernutzung mithilfe von Aurora-Funktionen
- Parameterreferenz für Ausführungspläne von Abfragen in Aurora PostgreSQL

Zugreifen auf Ausführungspläne von Abfragen und deren maximale Speichernutzung mithilfe von Aurora-Funktionen

Mit der `aurora_compute_plan_id` können Sie unter Verwendung der folgenden Funktionen auf die Ausführungspläne zugreifen:

- aurora_stat_activity
- aurora_stat_plans

Die maximale Speichernutzung von Abfragen umfasst nicht den Speicher, der vor Beginn der Abfrageverarbeitung zugewiesen wurde. Die maximale Speichernutzung wird für die Planungs- und Ausführungsphasen jeder Abfrage separat nachverfolgt und gemeldet.



Sie können mit den folgenden Funktionen auf die Statistiken zur maximalen Speichernutzung von Abfragen zugreifen:

- aurora_stat_statements
- aurora_stat_plans

Weitere Informationen zu diesen Funktionen finden Sie unter Aurora-PostgreSQL-Funktionsreferenz.

Parameterreferenz für Ausführungspläne von Abfragen in Aurora PostgreSQL

Sie können die Ausführungspläne von Abfragen mit den folgenden Parametern in einer DB-Parametergruppe überwachen.

Parameter

- aurora_compute_plan_id
- aurora_stat_plans.minutes_until_recapture
- aurora_stat_plans.calls_until_recapture
- aurora_stat_plans.with_costs
- aurora_stat_plans.with_analyze
- aurora_stat_plans.with_timing
- aurora_stat_plans.with_buffers
- aurora_stat_plans.with_wal
- aurora_stat_plans.with_triggers

> **Note**
>
> Die Konfiguration für `aurora_stat_plans.with_*`-Parameter wird nur für neu erfasste Pläne wirksam.

aurora_compute_plan_id

Die `aurora_compute_plan_id` ist ein Konfigurationsparameter, der steuert, ob während der Abfrageausführung eine Plan-ID zugewiesen wird.

Standard	Zulässige Werte	Beschreibung
aus	0 (aus)	Auf `off` setzen, um zu verhindern, dass eine Plan-ID zugewiesen wird
	1 (ein)	Auf on setzen, um eine Plan-ID zuzuweisen

aurora_stat_plans.minutes_until_recapture

Die Anzahl der Minuten, die vergehen sollen, bevor ein Plan erneut erfasst wird. Die Standardeinstellung ist 0, wodurch das erneute Erfassen eines Plans deaktiviert wird. Wenn der `aurora_stat_plans.calls_until_recapture`-Schwellenwert überschritten wird, wird der Plan erneut erfasst.

Standard	Zulässige Werte	Beschreibung
0	0-1073741823	Die Anzahl der Minuten festlegen, die vergehen sollen, bevor ein Plan erneut erfasst wird

aurora_stat_plans.calls_until_recapture

Die Anzahl der Aufrufe eines Plans, bevor er erneut erfasst wird. Die Standardeinstellung ist 0, wodurch das erneute Erfassen eines Plans nach einer Reihe von Aufrufen deaktiviert wird. Wenn der `aurora_stat_plans.minutes_until_recapture`-Schwellenwert überschritten wird, wird der Plan erneut erfasst.

Standard	Zulässige Werte	Beschreibung
0	0-1073741823	Die Anzahl der Aufrufe festlegen, bevor ein Plan erneut erfasst wird

aurora_stat_plans.with_costs

Erfasst einen EXPLAIN-Plan mit geschätzten Kosten. Die zulässigen Werte sind `on` und `off`. Der Standardwert ist `on`.

Standard	Zulässige Werte	Beschreibung
on	0 (aus)	Zeigt keine geschätzten Kosten und Zeilen für jeden Planknoten an
	1 (ein)	Zeigt geschätzte Kosten und Zeilen für jeden Planknoten an

aurora_stat_plans.with_analyze

Steuert den EXPLAIN-Plan mit ANALYZE. Dieser Modus wird nur verwendet, wenn ein Plan zum ersten Mal erfasst wird. Die zulässigen Werte sind on und `off`. Der Standardwert ist `off`.

Standard	Zulässige Werte	Beschreibung
aus	0 (aus)	Schließt keine tatsächlichen Laufzeitstatistiken für den Plan ein
	1 (ein)	Schließt tatsächliche Laufzeitstatistiken für den Plan ein

aurora_stat_plans.with_timing

Der Zeitpunkt des Plans wird in explain erfasst, wenn ANALYZE verwendet wird. Der Standardwert ist on.

Standard	Zulässige Werte	Beschreibung
on	0 (aus)	Schließt die tatsächliche Startzeit und die für jeden Planknoten aufgewendete Zeit nicht ein
	1 (ein)	Schließt die tatsächliche Startzeit und die für jeden Planknoten aufgewendete Zeit ein

aurora_stat_plans.with_buffers

Die Statistiken zur Nutzung des Planpuffers werden in explain erfasst, wenn ANALYZE verwendet wird. Der Standardwert ist `off`.

Standard	Zulässige Werte	Beschreibung
aus	0 (aus)	Schließt keine Informationen zur Puffernutzung ein
	1 (ein)	Schließt Informationen zur Puffernutzung ein

aurora_stat_plans.with_wal

Die Statistiken zur Nutzung des Plan-WAL werden in explain erfasst, wenn ANALYZE verwendet wird. Der Standardwert ist `off`.

Standard	Zulässige Werte	Beschreibung
aus	0 (aus)	Schließt keine Informationen zur Generierung von WAL-Daten sätzen ein
	1 (ein)	Schließt Informationen zur Generierung von WAL-Datensätzen ein

aurora_stat_plans.with_triggers

Die Statistiken zur Ausführung des Plan-Triggers werden in explain erfasst, wenn `ANALYZE` verwendet wird. Der Standardwert ist `off`.

Standard	Zulässige Werte	Beschreibung
aus	0 (aus)	Schließt keine Statistiken zur Ausführung von Triggern ein
	1 (ein)	Schließt Statistiken zur Ausführung von Triggern ein

Verwalten von Abfrageausführungsplänen für Aurora PostgreSQL

Die Abfrageplanverwaltung in Aurora PostgreSQL ist eine optionale Funktion, die Sie mit Ihrem DB-Cluster der mit Amazon Aurora PostgreSQL kompatiblen Edition verwenden können. Diese Funktion ist als `apg_plan_mgmt`-Erweiterung verpackt, die Sie in Ihrem DB-Cluster von Aurora PostgreSQL installieren können. Die Abfrageplanverwaltung ermöglicht Ihnen, die Abfrageausführungspläne zu verwalten, die vom Optimierer für Ihre SQL-Anwendungen generiert wurden. Die AWS-Erweiterung `apg_plan_mgmt` baut auf der nativen Abfrageverarbeitungsfunktionalität der PostgreSQL-Datenbank-Engine auf.

Im Folgenden finden Sie Informationen zu den Funktionen der Abfrageplanverwaltung in Aurora PostgreSQL sowie deren Einrichtung und Verwendung mit Ihrem DB-Cluster von Aurora PostgreSQL. Bevor Sie beginnen, sollten Sie alle Versionshinweise für die spezifische Version der `apg_plan_mgmt`-Erweiterung zu lesen, die für Ihre Aurora-PostgreSQL-Version verfügbar ist. Weitere Informationen finden Sie unter Versionen der apg_plan_mgmt-Erweiterung von Aurora PostgreSQL in den Versionshinweisen für Aurora PostgreSQL.

Themen

- Übersicht über die Abfrageplanverwaltung in Aurora PostgreSQL
- Bewährte Methoden für die Aurora-PostgreSQL-Abfrageplanverwaltung
- Die Abfrageplanverwaltung in Aurora PostgreSQL
- Erfassung von Aurora-PostgreSQL-Ausführungsplänen
- Verwenden von Aurora-PostgreSQL-Plänen
- Untersuchen von Aurora-PostgreSQL-Abfrageplänen in der dba_plans-Ansicht
- Verbessern von Aurora-PostgreSQL-Abfrageplänen
- Löschen von Aurora-PostgreSQL-Abfrageplänen
- Exportieren und Importieren von verwalteten Plänen für Aurora PostgreSQL
- Parameterreferenz für Aurora-PostgreSQL-Abfrageplanverwaltung
- Funktionsreferenz für die Aurora-PostgreSQL-Abfrageplanverwaltung
- Referenz für die apg_plan_mgmt.dba_plans-Ansicht für die Aurora-PostgreSQL-kompatible Edition
- Erweiterte Funktionen der Abfrageplanverwaltung

Übersicht über die Abfrageplanverwaltung in Aurora PostgreSQL

Die Aurora-PostgreSQL-Abfrageplanverwaltung wurde entwickelt, um die Planstabilität unabhängig von Änderungen an der Datenbank zu gewährleisten, die eine Regression des Abfrageplans verursachen könnten. Eine Regression des Abfrageplans tritt auf, wenn der Optimierer nach System- oder Datenbankänderungen einen suboptimalen Plan für eine bestimmte SQL-Anweisung auswählt. Änderungen von Statistiken, Einschränkungen, Umgebungseinstellungen, Abfrageparameter-Bindungen und Upgrades der PostgreSQL-Datenbank-Engine können eine Planregression verursachen.

Mit der Abfrageplanverwaltung in Aurora PostgreSQL steuern Sie, wie und wann sich Abfrageausführungspläne ändern. Zu den Vorteilen der Abfrageplanverwaltung in Aurora PostgreSQL gehören die nachfolgend aufgeführten.

- Verbesserung der Planstabilität, indem der Optimierer zur Auswahl aus einer kleinen Anzahl bekannter, bewährter Pläne gezwungen wird.
- Zentrale Optimierung von Plänen, gefolgt von einer globalen Verteilung der besten Pläne.
- Ermittlung von nicht verwendeten Indizes und Bewertung der Auswirkungen einer Indexerstellung oder -löschung.
- Automatische Erkennung eines vom Optimierer ermittelten Minimalkostenplans.
- Test neuer Optimierer-Funktionen mit geringerem Risiko, da Sie nur die Planänderungen genehmigen können, die zu Performance-Steigerungen führen.

Sie können die von der Abfrageplanverwaltung bereitgestellten Tools proaktiv verwenden, um den besten Plan für bestimmte Abfragen festzulegen. Sie können die Abfrageplanverwaltung auch verwenden, um auf sich ändernde Umstände zu reagieren und Planregressionen zu vermeiden. Weitere Informationen finden Sie unter Bewährte Methoden für die Aurora-PostgreSQL-Abfrageplanverwaltung.

Themen

- Unterstützte SQL-Anweisungen
- Einschränkungen der Abfrageplanverwaltung
- Terminologie der Abfrageplanverwaltung
- Die Abfrageplanverwaltung in Aurora-PostgreSQL-Versionen
- Aktivieren der Abfrageplanverwaltung in Aurora PostgreSQL

- Aktualisieren der Abfrageplanverwaltung in Aurora PostgreSQL
- Deaktivieren der Abfrageplanverwaltung in Aurora PostgreSQL

Unterstützte SQL-Anweisungen

Die Abfrageplanverwaltung unterstützt die folgenden Typen von SQL-Anweisungen.

- SELECT-, INSERT-, UPDATE- oder DELETE-Anweisungen, unabhängig von ihrer Komplexität.
- Vorbereitete Anweisungen. Weitere Informationen finden Sie unter PREPARE in der PostgreSQL-Dokumentation.
- Dynamische Anweisungen, einschließlich solcher, die im Sofortmodus ausgeführt werden. Weitere Informationen finden Sie unter Dynamisches SQL und EXECUTE IMMEDIATE in der PostgreSQL-Dokumentation.
- Eingebettete SQL-Befehle und -Anweisungen. Weitere Informationen finden Sie unter Eingebettete SQL-Befehle in der PostgreSQL-Dokumentation.
- Anweisungen innerhalb benannter Funktionen. Weitere Informationen finden Sie im Abschnitt CREATE FUNCTION der PostgreSQL-Dokumentation.
- Anweisungen, die temporäre Tabellen enthalten.
- Anweisungen innerhalb von Prozeduren und DO-Blöcken.

Sie können die Abfrageplanverwaltung mit `EXPLAIN` im manuellen Modus verwenden, um einen Plan zu erfassen, ohne ihn tatsächlich auszuführen. Weitere Informationen finden Sie unter Analysieren des vom Optimierer ausgewählten Plans. Weitere Informationen zu den Modi der Abfrageplanverwaltung (manuell, automatisch) finden Sie unter Erfassung von Aurora-PostgreSQL-Ausführungsplänen.

Die Abfrageplanverwaltung in Aurora PostgreSQL unterstützt sämtliche PostgreSQL-Sprachfunktionen, darunter partitionierte Tabellen, Vererbung, Sicherheit auf Zeilenebene und rekursive allgemeine Tabellenausdrücke (CTEs). Weitere Informationen zu diesen PostgreSQL-Sprachfunktionen finden Sie unter Table Partitioning (Tabellenpartitionierung), Row Security Policies (Zeilensicherheitsrichtlinien) und WITH Queries (Common Table Expressions) (WITH-Abfragen (allgemeine Tabellenausdrücke)) und anderen Themen in der PostgreSQL-Dokumentation.

Informationen zu den verschiedenen Versionen der Abfrageplanverwaltung von Aurora PostgreSQL finden Sie unter Versionen der apg_plan_mgmt-Erweiterung in den Versionshinweisen für Aurora PostgreSQL.

Einschränkungen der Abfrageplanverwaltung

Die aktuelle Version der Abfrageplanverwaltung von Aurora PostgreSQL unterliegt folgenden Einschränkungen.

- Pläne werden nicht für Anweisungen erfasst, die auf Systembeziehungen verweisen – Anweisungen, die auf Systembeziehungen verweisen, wie z. B. `pg_class`, werden nicht erfasst. Dies ist beabsichtigt, um zu verhindern, dass eine große Anzahl von systemgenerierten Plänen, die intern verwendet werden, erfasst werden. Dies gilt auch für Systemtabellen innerhalb von Ansichten.
- Für Ihren DB-Cluster von Aurora PostgreSQL ist möglicherweise eine größere DB-Instance-Klasse erforderlich – Je nach Workload erfordert die Abfrageplanverwaltung möglicherweise eine DB-Instance-Klasse mit mehr als 2 vCPUs. Die Anzahl von `max_worker_processes` ist durch die Größe der DB-Instance-Klasse begrenzt. Die Anzahl von `max_worker_processes`, die von einer 2-vCPU-DB-Instance-Klasse (z. B. db.t3.medium) bereitgestellt wird, ist möglicherweise für eine bestimmte Workload nicht ausreichend. Wir empfehlen Ihnen, bei Verwendung der Abfrageplanverwaltung eine DB-Instance-Klasse mit mehr als 2 vCPUs für Ihren DB-Cluster von Aurora PostgreSQL auszuwählen.

  Wenn die DB-Instance-Klasse Ihre Workload nicht unterstützen kann, gibt die Abfrageplanverwaltung eine Fehlermeldung wie die folgende aus.

  ```
  WARNING: could not register plan insert background process
  HINT: You may need to increase max_worker_processes.
  ```

  In diesem Fall sollten Sie Ihren DB-Cluster von Aurora PostgreSQL auf eine DB-Instance-Klassengröße mit mehr Speicher hochskalieren. Weitere Informationen finden Sie unter Unterstützte DB-Engines für DB-Instance-Klassen.
- Pläne, die bereits in Sitzungen gespeichert sind, sind nicht betroffen. – Die Abfrageplanverwaltung bietet eine Möglichkeit, Abfragepläne zu beeinflussen, ohne den Anwendungscode ändern zu müssen. Wenn jedoch ein generischer Plan bereits in einer vorhandenen Sitzung gespeichert ist und Sie den Abfrageplan ändern möchten, müssen Sie zunächst `plan_cache_mode` in der DB-Cluster-Parametergruppe auf `force_custom_plan` setzen.
- `queryid` in `apg_plan_mgmt.dba_plans` und `pg_stat_statements` kann in folgenden Fällen abweichen:
  - Objekte werden nach dem Speichern in apg_plan_mgmt.dba_plans gelöscht und neu erstellt.
  - Die Tabelle `apg_plan_mgmt.plans` wird aus einem anderen Cluster importiert.

Informationen zu den verschiedenen Versionen der Abfrageplanverwaltung von Aurora PostgreSQL finden Sie unter Versionen der apg_plan_mgmt-Erweiterung in den Versionshinweisen für Aurora PostgreSQL.

Terminologie der Abfrageplanverwaltung

Die folgenden Begriffe werden in diesem Thema verwendet.

verwaltete Anweisung

Eine SQL-Anweisung, die vom Optimierer im Rahmen der Abfrageplanverwaltung erfasst wurde. Eine verwaltete Anweisung verfügt über einen oder mehrere Abfrageausführungspläne, die in der `apg_plan_mgmt.dba_plans`-Ansicht gespeichert sind.

Plan-Baseline

Die Menge der genehmigten Pläne für eine bestimmte verwaltete Anweisung. Das heißt, alle Pläne für die verwaltete Anweisung, für die in der `status`-Spalte der `dba_plan`-Ansicht „Approved“ angegeben ist.

Planverlauf

Die Menge der erfassten Pläne für eine bestimmte verwaltete Anweisung. Der Planverlauf enthält alle Pläne, die für die Anweisung erfasst wurden, unabhängig vom Status.

Abfrageplanregression

Tritt auf, wenn der Optimierer einen weniger optimalen Plan als vor einer bestimmten Änderung an der Datenbankumgebung auswählt, z. B. eine neue PostgreSQL-Version oder Änderungen an Statistiken.

Die Abfrageplanverwaltung in Aurora-PostgreSQL-Versionen

Die Abfrageplanverwaltung wird von allen derzeit verfügbaren Aurora-PostgreSQL-Versionen unterstützt. Weitere Informationen finden Sie unter Aktualisierungen von Amazon Aurora PostgreSQL im Abschnitt Versionshinweise für Aurora PostgreSQL.

Die Funktion für die Abfrageplanverwaltung wird Ihrem DB-Cluster von Aurora PostgreSQL hinzugefügt, wenn Sie die `apg_plan_mgmt`-Erweiterung installieren. Verschiedene Versionen von Aurora PostgreSQL unterstützen unterschiedliche Versionen der `apg_plan_mgmt`-Erweiterung. Wir empfehlen Ihnen, die Erweiterung für die Abfrageplanverwaltung auf die neueste Version für Ihre Version von Aurora PostgreSQL zu aktualisieren.

> **Note**
>
> Versionshinweise für die Versionen jeder `apg_plan_mgmt`-Erweiterung finden Sie unter Aurora PostgreSQL apg_plan_mgmt extension versions (Versionen der apg_plan_mgmgmt-Erweiterung von Aurora PostgreSQL) in den Release Notes for Aurora PostgreSQL (Versionshinweisen für Aurora PostgreSQL).

Sie können die in Ihrem Cluster ausgeführte Version identifizieren, indem Sie mit `psql` eine Verbindung mit einer Instance herstellen und den Metabefehl \dx verwenden, um Erweiterungen aufzulisten, wie nachfolgend gezeigt.

```
labdb=> \dx
                       List of installed extensions
     Name      | Version |    Schema     |                            Description
---------------+---------+---------------
+-------------------------------------------------------------------
 apg_plan_mgmt | 1.0     | apg_plan_mgmt | Amazon Aurora with PostgreSQL compatibility
 Query Plan Management
 plpgsql       | 1.0     | pg_catalog    | PL/pgSQL procedural language
(2 rows)
```

Die Ausgabe zeigt, dass dieser Cluster die Version 1.0 der Erweiterung verwendet. Für eine bestimmte Aurora-PostgreSQL-Version sind nur bestimmte `apg_plan_mgmt`-Versionen verfügbar. In einigen Fällen müssen Sie möglicherweise den DB-Cluster von Aurora PostgreSQL auf eine neue Nebenversion aktualisieren oder einen Patch anwenden, damit Sie ein Upgrade auf die neueste Version der Abfrageplanverwaltung durchführen können. Die in der Ausgabe angezeigte `apg_plan_mgmt`-Version 1.0 stammt von einem DB-Cluster von Aurora-PostgreSQL-Version 10.17, für den keine neuere `apg_plan_mgmt`-Version verfügbar ist. In diesem Fall sollte der DB-Cluster von Aurora PostgreSQL auf eine neuere Version von PostgreSQL aktualisiert werden.

Weitere Informationen über die Aktualisierung Ihres DB-Clusters von Aurora PostgreSQL auf eine neue PostgreSQL-Version finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL.

Informationen zum Upgrade der `apg_plan_mgmt`-Erweiterung finden Sie unter Aktualisieren der Abfrageplanverwaltung in Aurora PostgreSQL.

Aktivieren der Abfrageplanverwaltung in Aurora PostgreSQL

Das Einrichten der Abfrageplanverwaltung für Ihren DB-Cluster von Aurora PostgreSQL umfasst die Installation einer Erweiterung und das Ändern mehrerer DB-Cluster-Parametereinstellungen. Sie benötigen `rds_superuser`-Berechtigungen, um die `apg_plan_mgmt`-Erweiterung zu installieren und die Funktion für den DB-Cluster von Aurora PostgreSQL zu aktivieren.

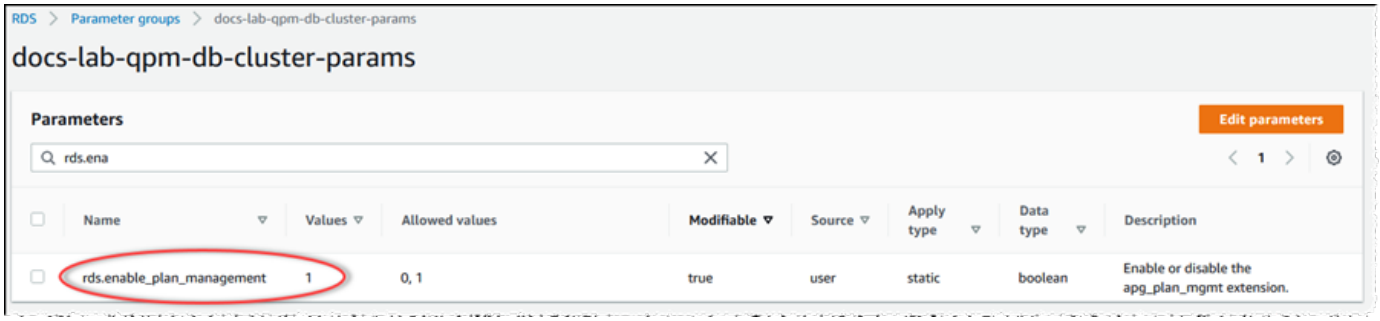
Durch die Installation der Erweiterung wird eine neue Rolle, `apg_plan_mgmt`, erstellt. Diese Rolle ermöglicht Datenbankbenutzern, Abfragepläne anzuzeigen, zu verwalten und zu pflegen. Als Administrator mit `rds_superuser`-Berechtigungen müssen Sie die `apg_plan_mgmt`-Rolle Datenbankbenutzern nach Bedarf zuweisen.

Nur Benutzer mit der `rds_superuser`-Rolle können den folgenden Vorgang ausführen. `rds_superuser` ist erforderlich, um die Erweiterung `apg_plan_mgmt` und die zugehörige Rolle `apg_plan_mgmt` zu erstellen. Benutzer benötigen die Rolle `apg_plan_mgmt` zur Verwaltung der Erweiterung `apg_plan_mgmt`.

So aktivieren Sie die Abfrageplanverwaltung für Ihren DB-Cluster von Aurora PostgreSQL

Mit den folgenden Schritten wird die Abfrageplanverwaltung für alle SQL-Anweisungen aktiviert, die an den DB-Cluster von Aurora PostgreSQL übermittelt werden. Dies wird als automatischer Modus bezeichnet. Weitere Informationen zum Unterschied zwischen den Modi finden Sie unter Erfassung von Aurora-PostgreSQL-Ausführungsplänen.

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe für Ihren DB-Cluster von Aurora PostgreSQL. Sie müssen bestimmte Parameter ändern, um die Abfrageplanverwaltung zu aktivieren und ihr Verhalten festzulegen. Weitere Informationen finden Sie unter Erstellen einer DB-Parametergruppe in Amazon Aurora.
3. Öffnen Sie die benutzerdefinierte DB-Cluster-Parametergruppe und legen Sie den `rds.enable_plan_management`-Parameter auf `1` fest, wie in der folgenden Abbildung gezeigt.



Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

4. Erstellen Sie eine benutzerdefinierte DB-Parametergruppe, mit der Sie Abfrageplanparameter auf Instance-Ebene festlegen können. Weitere Informationen finden Sie unter Erstellen einer DB-Cluster-Parametergruppe in Amazon Aurora.
5. Ändern Sie die Writer-Instance des DB-Clusters von Aurora PostgreSQL, um die benutzerdefinierte DB-Parametergruppe zu verwenden. Weitere Informationen finden Sie unter Ändern einer DB-Instance in einem DB-Cluster.
6. Ändern Sie den DB-Cluster von Aurora PostgreSQL, um die benutzerdefinierte DB-Parametergruppe zu verwenden. Weitere Informationen finden Sie unter Ändern des DB-Clusters über die Konsole, die CLI und die API.
7. Starten Sie Ihre DB-Instance neu, um die Einstellungen der benutzerdefinierten Parametergruppe zu aktivieren.
8. Stellen Sie eine Verbindung mit dem DB-Instance-Endpunkt Ihres DB-Clusters von Aurora PostgreSQL mit `psql` oder `pgAdmin` her. Im folgenden Beispiel wird das `postgres`-Standardkonto für die `rds_superuser`-Rolle verwendet.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=my-db
```

9. Erstellen Sie die `apg_plan_mgmt`-Erweiterung für Ihre DB-Instance, wie nachfolgend gezeigt.

```
labdb=> CREATE EXTENSION apg_plan_mgmt;
CREATE EXTENSION
```

> **Tip**
>
> Installieren Sie die `apg_plan_mgmt`-Erweiterung in der Vorlagendatenbank für Ihre Anwendung. Der Name der Standard-Vorlagendatenbank lautet `template1`. Weitere

Informationen finden Sie unter Template Databases (Vorlagendatenbanken) in der PostgreSQL-Dokumentation.

10. Ändern Sie den Parameter `apg_plan_mgmt.capture_plan_baselines` in `automatic`. Diese Einstellung veranlasst den Optimierer, Pläne für jede SQL-Anweisung zu generieren, die entweder geplant oder mindestens zweimal ausgeführt wird.

> **Note**
>
> Die Abfrageplanverwaltung verfügt auch über einen manuellen Modus, den Sie für bestimmte SQL-Anweisungen verwenden können. Weitere Informationen hierzu finden Sie unter Erfassung von Aurora-PostgreSQL-Ausführungsplänen.

11. Ändern Sie den Wert des `apg_plan_mgmt.use_plan_baselines`-Parameters in „on“. Dieser Parameter veranlasst den Optimierer, einen Plan für die Anweisung aus der Plan-Baseline auszuwählen. Weitere Informationen hierzu finden Sie unter Verwenden von Aurora-PostgreSQL-Plänen.

> **Note**
>
> Sie können den Wert eines dieser dynamischen Parameter für die Sitzung ändern, ohne die Instance neu starten zu müssen.

Wenn die Einrichtung der Abfrageplanverwaltung abgeschlossen ist, müssen Sie die `apg_plan_mgmt`-Rolle allen Datenbankbenutzern zuweisen, die Abfragepläne anzeigen, verwalten oder pflegen müssen.

Aktualisieren der Abfrageplanverwaltung in Aurora PostgreSQL

Wir empfehlen Ihnen, die Erweiterung für die Abfrageplanverwaltung auf die neueste Version für Ihre Version von Aurora PostgreSQL zu aktualisieren.

1. Stellen Sie eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL als Benutzer mit `rds_superuser`-Berechtigungen her. Wenn Sie beim Einrichten Ihrer Instance den Standardnamen beibehalten haben, stellen Sie eine Verbindung als `postgres` her. In diesem Beispiel wird die Verwendung von `psql` gezeigt, Sie können jedoch auch pgAdmin verwenden, wenn Sie möchten.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. Führen Sie die folgende Abfrage aus, um die Erweiterung zu aktualisieren.

```
ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.1';
```

3. Verwenden Sie die Funktion apg_plan_mgmt.validate_plans, um die Hashes aller Pläne zu aktualisieren. Der Optimierer validiert alle genehmigten, nicht genehmigten und abgelehnten Pläne, um sicherzustellen, dass es sich weiterhin um durchführbare Pläne für die neue Version der Erweiterung handelt.

```
SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
```

   Weitere Informationen zu dieser Funktion finden Sie unter Validieren von Plänen.

4. Verwenden Sie die Funktion apg_plan_mgmt.reload, um alle Pläne im gemeinsam genutzten Arbeitsspeicher mit den validierten Plänen aus der dba_plans-Ansicht zu aktualisieren.

```
SELECT apg_plan_mgmt.reload();
```

Weitere Informationen zu allen Funktionen, die für die Abfrageplanverwaltung verfügbar sind, finden Sie unter Funktionsreferenz für die Aurora-PostgreSQL-Abfrageplanverwaltung.

Deaktivieren der Abfrageplanverwaltung in Aurora PostgreSQL

Sie können die Abfrageplanverwaltung jederzeit deaktivieren, indem Sie die `apg_plan_mgmt.use_plan_baselines` und `apg_plan_mgmt.capture_plan_baselines` deaktivieren.

```
labdb=> SET apg_plan_mgmt.use_plan_baselines = off;

labdb=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Bewährte Methoden für die Aurora-PostgreSQL-Abfrageplanverwaltung

Mit der Abfrageplanverwaltung steuern Sie, wie und wann sich Abfrageverwaltungspläne ändern. Als DBA gehören zu Ihren Hauptzielen bei der Verwendung von QPM, Regressionen bei Änderungen

an Ihrer Datenbank zu verhindern und zu kontrollieren, ob der Optimierer einen neuen Plan verwenden darf. Im Folgenden finden Sie einige empfohlene bewährte Methoden zur Verwendung der Abfrageplanverwaltung. Proaktive und reaktive Planverwaltungsansätze unterscheiden sich hinsichtlich der Frage, wie und wann neue Pläne zur Verwendung genehmigt werden.

Inhalt

- Proaktive Planverwaltung zur Vermeidung von Performancerückgängen
  - Sicherstellen der Planstabilität nach einem größeren Versions-Upgrade
- Reaktive Planverwaltung zur Erkennung und Behebung von Performancerückgängen

Proaktive Planverwaltung zur Vermeidung von Performancerückgängen

Um Regressionen bei der Planleistung zu verhindern, entwickeln Sie den Basisplan weiter. Dazu führen Sie ein Verfahren aus, das die Leistung neu entdeckter Pläne mit der Leistung des vorhandenen Basisplans der genehmigten Pläne vergleicht und dann automatisch die schnellsten Pläne als neuen Basisplan genehmigt. Auf diese Weise verbessert sich die Ausgangsbasis der Pläne im Laufe der Zeit, da schnellere Pläne entdeckt werden.

1. Bestimmen Sie in einer Entwicklungsumgebung die SQL-Anweisungen, die die größten Auswirkungen auf die Performance oder den Durchsatz des Systems haben. Erfassen Sie dann die Pläne für diese Anweisungen anhand der unter Manuelles Erfassen von Plänen für bestimmte SQL-Anweisungen und Automatisches Erfassen von Plänen beschriebenen Vorgehensweise.
2. Exportieren Sie die erfassten Pläne aus der Entwicklungsumgebung und importieren Sie sie in die Produktionsumgebung. Weitere Informationen finden Sie unter Exportieren und Importieren von verwalteten Plänen für Aurora PostgreSQL.
3. Führen Sie in der Produktion Ihre Anwendung aus und erzwingen Sie die Verwendung genehmigter verwalteter Pläne. Weitere Informationen finden Sie unter Verwenden von Aurora-PostgreSQL-Plänen. Fügen Sie bei laufender Anwendung auch neue Pläne hinzu, sobald diese vom Optimierer erkannt werden. Weitere Informationen finden Sie unter Automatisches Erfassen von Plänen.
4. Analysieren Sie die nicht genehmigten Pläne und genehmigen Sie jene, die ordnungsgemäß funktionieren. Weitere Informationen finden Sie unter Auswerten der Performance von Plänen.
5. Während Ihre Anwendung ausgeführt wird, verwendet der Optimierer die neuen Pläne bei Bedarf.

Sicherstellen der Planstabilität nach einem größeren Versions-Upgrade

Jede Hauptversion von PostgreSQL enthält Verbesserungen und Änderungen am Abfrageoptimierer, die auf eine Leistungssteigerung ausgelegt sind. Pläne zur Abfrageausführung, die in früheren Versionen vom Optimierer generiert wurden, können jedoch in neueren, aktualisierten Versionen zu Leistungseinbußen führen. Sie können das Abfrageplan-Management verwenden, um diese Leistungsprobleme zu beheben und die Planstabilität nach einem größeren Versions-Upgrade sicherzustellen.

Der Optimierer verwendet immer den kostengünstigsten genehmigten Plan, auch wenn mehr als ein genehmigter Plan für die gleiche Anweisung vorhanden ist. Nach einem Upgrade ermittelt der Optimierer möglicherweise neue Pläne, diese werden jedoch als nicht genehmigte Pläne gespeichert. Diese Pläne werden nur ausgeführt, wenn sie im reaktiven Planverwaltungsstil mit dem Parameter unapproved_plan_execution_threshold genehmigt wurden. Sie können die Planstabilität maximieren, indem Sie den proaktiven Planverwaltungsstil mit dem Parameter evolve_plan_baselines verwenden. Dadurch werden die Leistung der neuen Pläne mit den alten Plänen verglichen und Pläne genehmigt oder abgelehnt, die mindestens 10 % schneller sind als der nächstbeste Plan.

Nach dem Upgrade können Sie die Funktion `evolve_plan_baselines` verwenden, um die Planleistung vor und nach dem Upgrade mit Ihren Abfrageparameter-Bindungen zu vergleichen. Die folgenden Schritte gehen davon aus, dass Sie genehmigte verwaltete Pläne in Ihrer Produktionsumgebung verwendet haben, wie in Verwenden von Aurora-PostgreSQL-Plänen ausgeführt.

1. Führen Sie Ihre Anwendung vor dem Upgrade mit dem laufenden Abfrageplan-Manager aus. Fügen Sie bei laufender Anwendung neue Pläne hinzu, sobald diese vom Optimierer erkannt werden. Weitere Informationen finden Sie unter Automatisches Erfassen von Plänen.
2. Bewerten Sie die Leistung jedes Plans. Weitere Informationen finden Sie unter Auswerten der Performance von Plänen.
3. Analysieren Sie nach dem Upgrade Ihre genehmigten Pläne erneut mit der Funktion `evolve_plan_baselines`. Vergleichen Sie die Leistung vor und nach der Verwendung Ihrer Abfrageparameter-Bindungen. Wenn der neue Plan schnell ist, können Sie ihn zu Ihren genehmigten Plänen hinzufügen. Wenn er für dieselben Parameter-Bindungen schneller als ein anderer Plan ist, können Sie den langsameren Plan als „Rejected“ (Abgelehnt) markieren.

   Weitere Informationen finden Sie unter Genehmigen besserer Pläne. Referenzinformationen zu dieser Funktion finden Sie unter apg_plan_mgmt.evolve_plan_baselines.

Weitere Informationen finden Sie unter Sicherstellen einer konsistenten Leistung nach größeren Versions-Upgrades mit der Abfrageplanverwaltung der PostgreSQL-kompatiblen Edition von Amazon Aurora.

> **Note**
>
> Wenn Sie ein Upgrade der Hauptversion mithilfe der logischen Replikation oder mit AWS DMS durchführen, stellen Sie sicher, dass Sie das Schema `apg_plan_mgmt` replizieren, damit die vorhandenen Pläne auf die aktualisierte Instance kopiert werden. Weitere Informationen zur logischen Replikation finden Sie unter Verwenden der logischen Replikation, um ein Hauptversions-Upgrade für Aurora PostgreSQL durchzuführen.

Reaktive Planverwaltung zur Erkennung und Behebung von Performancerückgängen

Durch Überwachen Ihrer laufende Anwendung können Sie Pläne erkennen, die zu Performancerückgängen führen. Werden Rückgänge erkannt, können Sie die nicht funktionierenden Pläne manuell ablehnen oder reparieren. Führen Sie dazu die folgenden Schritte aus:

1. Erzwingen Sie bei laufender Anwendung die Verwendung verwalteter Pläne und fügen Sie automatisch neu entdeckte Pläne als nicht genehmigte Pläne hinzu. Weitere Informationen erhalten Sie unter Verwenden von Aurora-PostgreSQL-Plänen und Automatisches Erfassen von Plänen.
2. Überwachen Sie die laufende Anwendung auf Performance-Regressionen.
3. Setzen Sie den Status des Plans auf , wenn Sie eine Planregression feststelle `rejected`. Wenn der Optimierer die SQL-Anweisung das nächste Mal ausführt, wird der abgelehnte Plan automatisch ignoriert und stattdessen ein anderer genehmigter Plan verwendet. Weitere Informationen finden Sie unter Ablehnen oder Deaktivieren langsamerer Pläne.

   In einigen Fällen ist es möglicherweise sinnvoll, einen nicht funktionierenden Plan zu reparieren, anstatt ihn abzulehnen, zu deaktivieren oder zu löschen. Verwenden Sie die Erweiterung `pg_hint_plan`, um das Verbessern eines Plans zu testen. Mit `pg_hint_plan` nutzen Sie besondere Kommentare, um die üblicherweise vom Optimierer angewendete Planerstellungsmethode zu überschreiben. Weitere Informationen finden Sie unter Reparieren von Plänen mit pg_hint_plan.

Die Abfrageplanverwaltung in Aurora PostgreSQL

Wenn die Abfrageplanverwaltung für Ihren DB-Cluster von Aurora PostgreSQL aktiviert ist, generiert und speichert der Optimierer Abfrageausführungspläne für jede SQL-Anweisung, die er mehr als einmal verarbeitet. Der Status des zuerst generierten Plans einer verwalteten Anweisung wird vom Optimierer immer auf `Approved` festgelegt und in der `dba_plans`-Ansicht gespeichert.

Der Satz an genehmigten Plänen, der für eine verwaltete Anweisung gespeichert wird, wird als Plan-Baseline bezeichnet. Während der Ausführung Ihrer Anwendung generiert der Optimierer möglicherweise zusätzliche Pläne für verwaltete Anweisungen. Der Status zusätzlich erfasster Pläne wird vom Optimierer auf `Unapproved` festgelegt.

Später können Sie entscheiden, ob die `Unapproved`-Pläne ordnungsgemäß funktionieren und ihren Status bei Bedarf in `Approved`, `Rejected` oder `Preferred` ändern. Dazu verwenden Sie die Funktion `apg_plan_mgmt.evolve_plan_baselines` oder `apg_plan_mgmt.set_plan_status`.

Wenn der Optimierer einen Plan für eine SQL-Anweisung generiert, speichert die Abfrageplanverwaltung den Plan in der `apg_plan_mgmt.plans`-Tabelle. Datenbankbenutzer, denen die `apg_plan_mgmt`-Rolle zugewiesen wurde, können die Plandetails anzeigen, indem sie die `apg_plan_mgmt.dba_plans`-Ansicht abfragen. Die folgende Abfrage listet beispielsweise Details zu Plänen auf, die sich derzeit in der Ansicht für einen DB-Cluster von Aurora PostgreSQL außerhalb der Produktion befinden.

- `sql_hash`: Ein Bezeichner für die SQL-Anweisung, der der Hashwert für den normalisierten Text der SQL-Anweisung ist.
- `plan_hash`: Ein eindeutiger Bezeichner für den Plan, der eine Kombination aus dem `sql_hash` und einem Hash des Plans ist.
- `status`: der Status des Plans Der Optimierer kann einen genehmigten Plan ausführen.
- `enabled`: Gibt an, ob der Plan einsatzbereit ist (true) oder nicht (false).
- `plan_outline`: Eine Darstellung des Plans, mit der der tatsächliche Ausführungsplan neu erstellt wird. Operatoren in der Baumstruktur werden Operatoren in der EXPLAIN-Ausgabe zugeordnet.

Die `apg_plan_mgmt.dba_plans`-Ansicht hat viele weitere Spalten, die alle Details des Plans enthalten, z. B. wann der Plan zuletzt verwendet wurde. Vollständige Details finden Sie unter Referenz für die apg_plan_mgmt.dba_plans-Ansicht für die Aurora-PostgreSQL-kompatible Edition.

Normalisierung und der SQL-Hash

In der `apg_plan_mgmt.dba_plans`-Ansicht können Sie eine verwaltete Anweisung anhand ihres SQL-Hash-Werts erkennen. Der SQL-Hash wird auf Basis einer normalisierten Darstellung der SQL-Anweisung berechnet, bei der einige Unterschiede (z. B. Literalwerte) nicht berücksichtigt sind.

Beim Normalisierungsprozess für jede SQL-Anweisung werden Leerzeichen und Groß- und Kleinschreibung beibehalten, sodass Sie das Wesentliche der SQL-Anweisung weiterhin lesen und verstehen können. Durch die Normalisierung werden die folgenden Elemente entfernt oder ersetzt.

- Führende Blockkommentare
- Das Schlüsselwort EXPLAIN und EXPLAIN-Optionen sowie EXPLAIN ANALYZE
- Leerzeichen am Zeilenende
- Alle Literale

Sehen Sie sich die folgende Anweisung als Beispiel an.

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

Die Abfrageplanverwaltung normalisiert diese Anweisung wie im Folgenden gezeigt:

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

Durch die Normalisierung kann derselbe SQL-Hash für ähnliche SQL-Anweisungen verwendet werden, die sich ggf. ausschließlich in ihren Literal- oder Parameterwerten unterscheiden. Mit anderen Worten, es können mehrere Pläne für denselben SQL-Hash existieren, wobei ein anderer Plan unter anderen Bedingungen optimal ist.

> **Note**
>
> Eine einzelne SQL-Anweisung, die mit verschiedenen Schemas verwendet wird, hat unterschiedliche Pläne, da sie an das spezifische Schema zur Laufzeit gebunden ist. Der Planer verwendet die Statistiken für die Schemabindung, um den optimalen Plan auszuwählen.

Weitere Informationen dazu, wie der Optimierer den Plan auswählt, finden Sie unter Verwenden von Aurora-PostgreSQL-Plänen. In diesem Abschnitt erfahren Sie, wie Sie einen Plan mit `EXPLAIN`

und `EXPLAIN ANALYZE` in der Vorschau anzeigen, bevor er tatsächlich verwendet wird. Details hierzu finden Sie unter Analysieren des vom Optimierer ausgewählten Plans. Ein Abbildung, die den Prozess zur Auswahl eines Plans beschreibt, finden Sie unter Erfahren Sie, wie der Optimierer bestimmt, welche Pläne ausgeführt werden..

Erfassung von Aurora-PostgreSQL-Ausführungsplänen

Die Aurora-PostgreSQL-Abfrageplanverwaltung bietet zwei verschiedene Modi zum Erfassen von Abfrageausführungsplänen: automatisch oder manuell. Sie wählen den Modus aus, indem Sie den Wert `apg_plan_mgmt.capture_plans_baselines` auf `automatic` oder `manual` festlegen. Sie können Ausführungspläne für bestimmte SQL-Anweisungen mit der manuellen Planerfassung erfassen. Alternativ können Sie mit der automatischen Planerfassung alle (oder nur die langsamsten) Pläne erfassen, die bei laufender Anwendung mindestens zwei Mal ausgeführt werden.

Bei der Planerfassung wird der Status des zuerst erfassten Plans einer verwalteten Anweisung vom Optimierer auf gesetz `approved`. Der Status zusätzlich hinzugefügter Pläne, die für eine verwaltete Anweisung erfasst wurden, wird vom Optimierer auf festgeleg `unapproved`. Gelegentlich wird jedoch mehr als ein Plan möglicherweise mit dem Status `approved` gespeichert. Dies kann passieren, wenn für eine Anweisung mehrere Pläne gleichzeitig erstellt werden und der erste Plan für die Anweisung noch nicht übermittelt wurde.

Legen Sie den Parameter `dba_plans` in der Parametergruppe der DB-Instance-Ebene fest, um die maximale Anzahl an Plänen zu bestimmen, die in der `apg_plan_mgmt.max_plans`-Ansicht erfasst und gespeichert werden können. Damit der Parameter `apg_plan_mgmt.max_plans` geändert werden und ein neuer Wert übernommen werden kann, muss die DB-Instance neu gestartet werden. Informieren Sie sich über den Parameter apg_plan_mgmt.max_plans, um weitere Informationen hierzu zu erhalten.

Manuelles Erfassen von Plänen für bestimmte SQL-Anweisungen

Wenn Sie einen bekannten Satz an SQL-Anweisungen verwalten, verschieben Sie die Anweisungen in eine SQL-Skriptdatei und erfassen Sie anschließend Pläne manuell. Im folgenden psql-Beispiel wird gezeigt, wie Abfragepläne für einen Satz an SQL-Anweisungen manuell erfasst werden.

```
psql> SET apg_plan_mgmt.capture_plan_baselines = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Nachdem für jede SQL-Anweisung ein Plan erfasst wurde, fügt der Optimierer der `apg_plan_mgmt.dba_plans`-Ansicht eine neue Zeile hinzu.

Wir empfehlen, dass Sie in der SQL-Skriptdatei entweder EXPLAN- oder EXPLAIN EXECUTE-Anweisungen verwenden. Stellen Sie sicher, dass eine ausreichende Auswahl an Parameterwerten einzukalkulieren, damit alle relevanten Pläne erfasst werden.

Falls Sie einen besseren Plan als den Minimalkostenplan des Optimierers nutzen möchten, können Sie den Optimierer zur Verwendung dieses bevorzugten Plans zwingen, indem Sie einen oder mehrere Optimierungshinweise angeben. Geben Sie hierzu mindestens einen Optimierungshinweis an. Weitere Informationen finden Sie unter Reparieren von Plänen mit pg_hint_plan. Unter `unapproved` erfahren Sie, wie Sie die Performance der `approved`- und Auswerten der Performance von Plänen-Pläne vergleichen und diese genehmigen, ablehnen oder löschen.

Automatisches Erfassen von Plänen

Verwenden Sie die automatische Planerfassung u. a. in den folgenden Fällen:

- Sie wissen nicht, welche spezifischen SQL-Anweisungen Sie verwalten möchten.
- Sie müssen hunderte oder tausende von SQL-Anweisungen verwalten.
- Ihre Anwendung verwendet eine Client-API. Beispielsweise verwendet JDBC unbenannte vorbereitete Anweisungen oder Anweisungen in großen Mengen, die in psql nicht wiedergegeben werden können.

So erfassen Sie Pläne automatisch:

1. Aktivieren Sie die automatische Planerfassung, indem Sie in der Parametergruppe der DB-Instance-Ebene `apg_plan_mgmt.capture_plan_baselines` auf `automatic` setzen. Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.
2. Während der Ausführung der Anwendung erfasst der Optimierer Pläne für alle SQL-Anweisungen, die mindestens zwei Mal ausgeführt werden.

   Wenn die Anwendung mit den standardmäßigen Parametereinstellungen für die Abfrageplanverwaltung ausgeführt wird, erfasst der Optimierer Pläne für alle SQL-Anweisungen, die mindestens zwei Mal ausgeführt werden. Das Erfassen aller Pläne mit den Standardeinstellungen ist mit einem sehr geringen Laufzeitaufwand verbunden und kann in der Produktivumgebung erfolgen.

So deaktivieren Sie die automatische Planerfassung:

- Setzen Sie in der Parametergruppe der DB-Instance-Ebene den Parameter `apg_plan_mgmt.capture_plan_baselines` auf `off`.

Erfahren Sie unter , wie Sie die Performance der nicht genehmigten Pläne messen und diese Pläne genehmigen, ablehnen oder lösche Auswerten der Performance von Plänen.

Verwenden von Aurora-PostgreSQL-Plänen

Setzen Sie den Parameter `apg_plan_mgmt.use_plan_baselines` auf `true`, damit der Optimierer erfasste Pläne für Ihre verwalteten Anweisungen verwendet. Im Folgenden wird ein Beispiel mit einer lokalen Instance gezeigt:

```
SET apg_plan_mgmt.use_plan_baselines = true;
```

Diese Einstellung führt bei laufender Anwendung dazu, dass der Optimierer für jede verwaltete Anweisung jenen kostengünstigsten, bevorzugten oder genehmigten Plan verwendet, der gültig und aktiviert ist.

Analysieren des vom Optimierer ausgewählten Plans

Wenn der Parameter `apg_plan_mgmt.use_plan_baselines` auf `true` gesetzt ist, können Sie mit SQL-Anweisungen vom Typ EXPLAIN ANALYZE erfahren, welchen Plan der Optimierer zum Ausführen der Anweisung auswählen würde. Im Folgenden wird ein Beispiel gezeigt.

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
                                                        QUERY PLAN
-----------------------------------------------------------------------------
 Aggregate  (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1
 loops=1)
   ->  Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
 (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
         Heap Fetches: 10000
 Planning time: 1.408 ms
 Execution time: 7.291 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1984047223, Plan Hash: 512153379
```

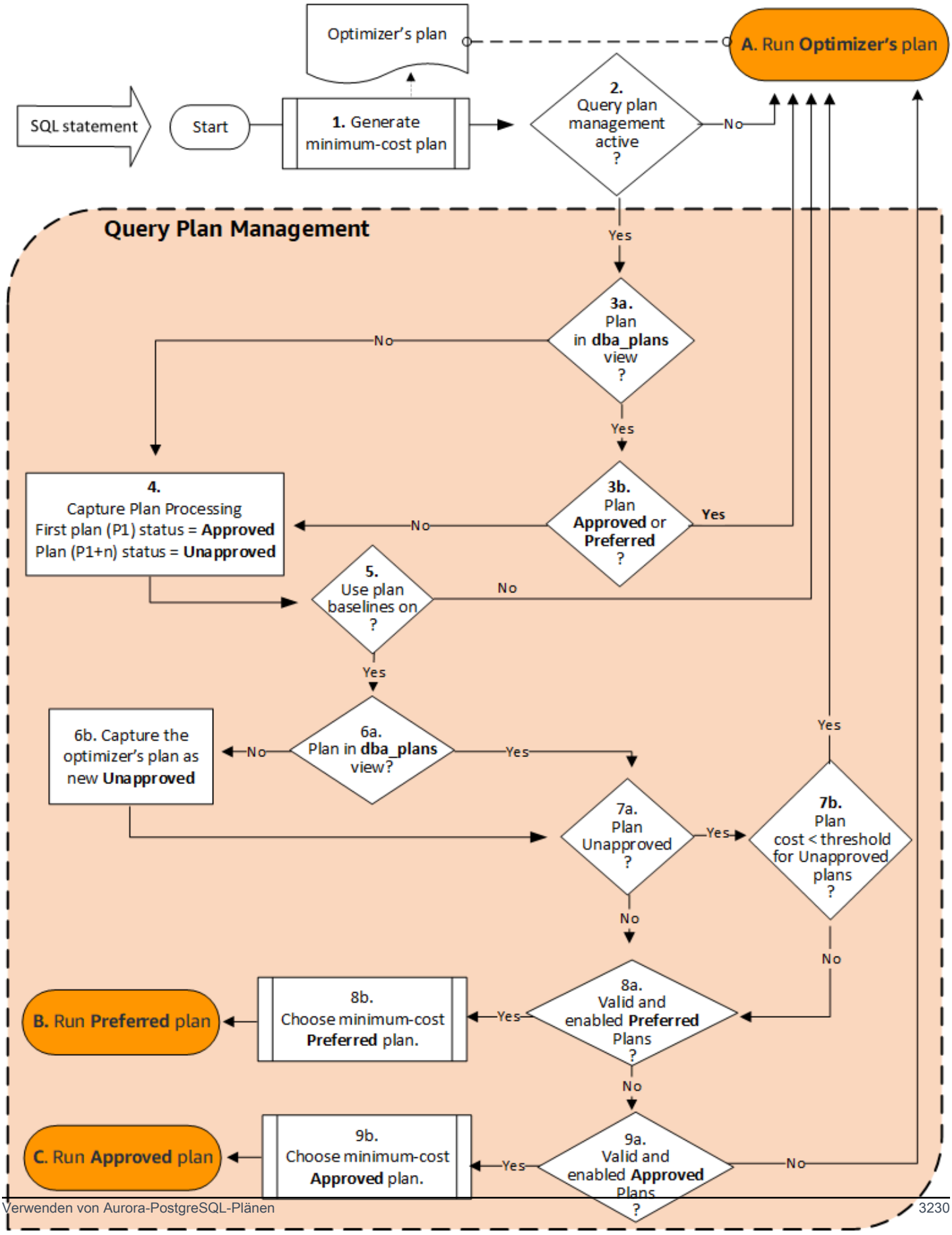
Die Ausgabe zeigt den genehmigten Plan aus der Baseline, der ausgeführt werden würde. Die Ausgabe zeigt jedoch auch, dass ein kostengünstigerer Plan gefunden wurde. In diesem Fall erfassen Sie diesen neuen Minimalkostenplan, indem Sie die unter beschriebene automatische Planerfassung aktiviere Automatisches Erfassen von Plänen.

Neue Pläne werden vom Optimierer immer als `Unapproved` erfasst. Mit der Funktion `apg_plan_mgmt.evolve_plan_baselines` vergleichen Sie Pläne und können deren Status zu „Genehmigt“, „Abgelehnt“ oder „Deaktiviert“ ändern. Weitere Informationen finden Sie unter Auswerten der Performance von Plänen.

Erfahren Sie, wie der Optimierer bestimmt, welche Pläne ausgeführt werden.

Die Kosten für einen Ausführungsplan stellen eine Schätzung dar, die der Optimierer zum Vergleichen verschiedener Pläne durchführt. Bei der Berechnung der Kosten eines Plans berücksichtigt der Optimierer Faktoren wie CPU- und I/O-Vorgänge, die für diesen Plan erforderlich sind. Um mehr über Kostenschätzungen des PostgreSQL-Abfrageplaners zu erfahren, lesen Sie bitte Query Planning (Abfrageplanung) in der PostgreSQL-Dokumentation.

Die folgende Abbildung zeigt, wie ein Plan für eine bestimmte SQL-Anweisung ausgewählt wird, wenn die Abfrageplanverwaltung aktiv und nicht aktiv ist.


Optimizer's plan
A. Run Optimizer's plan
SQL statement
Start
1. Generate minimum-cost plan
2. Query plan management active ?
No
Query Plan Management
Yes
3a. Plan in dba_plans view ?
No
Yes
4. Capture Plan Processing
First plan (P1) status = Approved
Plan (P1+n) status = Unapproved
3b. Plan Approved or Preferred ?
No
Yes
5. Use plan baselines on ?
No
Yes
6a. Plan in dba_plans view?
No
6b. Capture the optimizer's plan as new Unapproved
Yes
7a. Plan Unapproved ?
Yes
7b. Plan cost < threshold for Unapproved plans ?
Yes
No
No
8a. Valid and enabled Preferred Plans ?
Yes
8b. Choose minimum-cost Preferred plan.
B. Run Preferred plan
No
9a. Valid and enabled Approved Plans ?
Yes
9b. Choose minimum-cost Approved plan.
C. Run Approved plan
No

Der Ablauf ist wie folgt:

1. Der Optimierer erstellt einen Minimalkostenplan für die SQL-Anweisung.
2. Wenn die Abfrageplanverwaltung nicht aktiv ist, wird der Plan des Optimierers sofort ausgeführt (A. Optimizer-Plan ausführen). Die Abfrageplanverwaltung ist inaktiv, wenn die `apg_plan_mgmt.capture_plan_baselines` und `apg_plan_mgmt.use_plan_baselines`-Parameter beide ihre Standardeinstellungen haben („off“ bzw. „false“).

   Andernfalls ist die Abfrageplanverwaltung aktiv. In diesem Fall werden die SQL-Anweisung und der Plan des Optimierers dafür weiter bewertet, bevor ein Plan ausgewählt wird.

   > **Tip**
   >
   > Datenbankbenutzer mit der `apg_plan_mgmt`-Rolle können proaktiv Pläne vergleichen, den Status von Plänen ändern und die Verwendung bestimmter Pläne nach Bedarf erzwingen. Weitere Informationen finden Sie unter Verbessern von Aurora-PostgreSQL-Abfrageplänen.

3. Die SQL-Anweisung enthält möglicherweise bereits Pläne, die in der Vergangenheit von der Abfrageplanverwaltung gespeichert wurden. Pläne werden in der `apg_plan_mgmt.dba_plans` aufbewahrt, zusammen mit Informationen über die SQL-Anweisungen, mit denen sie erstellt wurden. Informationen zu einem Plan beinhalten seinen Status. Der Status eines Plans kann wie folgt bestimmen, ob er verwendet wird oder nicht.
   a. Wenn der Plan nicht zu den gespeicherten Plänen für die SQL-Anweisung gehört, bedeutet dies, dass dieser spezielle Plan zum ersten Mal vom Optimierer für die angegebene SQL-Anweisung generiert wurde. Der Plan wird an die Erfassungsplanverarbeitung gesendet (4).
   b. Wenn der Plan zu den gespeicherten Plänen gehört und sein Status Genehmigt oder Bevorzugt ist, wird der Plan ausgeführt (A. Optimizer-Plan ausführen).

      Wenn der Plan zu den gespeicherten Plänen gehört, aber weder Genehmigt noch Bevorzugt ist, wird der Plan an die Erfassungsplanverarbeitung gesendet (4).
4. Wenn ein Plan zum ersten Mal für eine bestimmte SQL-Anweisung erfasst wird, wird der Status des Plans immer auf Genehmigt (P1) festgelegt. Wenn der Optimierer anschließend denselben Plan für dieselbe SQL-Anweisung generiert, wird der Status dieses Plans in Nicht genehmigt (P1+n) geändert.

Nachdem der Plan erfasst und sein Status aktualisiert wurde, wird die Bewertung im nächsten Schritt fortgesetzt (5).

5. Die Baseline eines Plans besteht aus der Historie der SQL-Anweisung und ihrer Pläne in verschiedenen Stadien. Die Abfrageplanverwaltung kann den Basisplan bei der Auswahl eines Plans berücksichtigen, je nachdem, ob die Option Plan-Baselines verwenden aktiviert ist oder nicht, wie folgt.
   - Plan-Baselines ist „Aus“ verwenden, wenn der `apg_plan_mgmt.use_plan_baselines`-Parameter auf den Standardwert (`false`) festgelegt ist. Der Plan wird nicht mit der Baseline verglichen, bevor er ausgeführt wird (A. Optimizer-Plan ausführen).
   - Plan-Baselines ist „Ein“ verwenden, wenn der `apg_plan_mgmt.use_plan_baselines`-Parameter auf den Standardwert (`true`) festgelegt ist. Der Plan wird anhand der Basislinie weiter bewertet (6).
6. Der Plan wird mit anderen Plänen für den Kontoauszug in der Basislinie verglichen.
   a. Wenn der Plan des Optimierers zu den Plänen in der Baseline gehört, wird sein Status überprüft (7a).
   b. Befindet sich der Plan des Optimierers nicht unter den Plänen in der Baseline, wird der Plan als neuer `Unapproved`-Plan zu den Plänen für die Anweisung hinzugefügt.
7. Der Status des Plans wird überprüft, um nur festzustellen, ob er nicht genehmigt ist.
   a. Wenn der Status des Plans Nicht genehmigt lautet, werden die geschätzten Kosten des Plans mit der Kostenschätzung verglichen, die für den Schwellenwert für den nicht genehmigten Ausführungsplan angegeben wurde.
      - Wenn die geschätzten Kosten des Plans unter dem Schwellenwert liegen, verwendet der Optimierer ihn, obwohl es sich um einen nicht genehmigten Plan handelt (A. Optimizer-Plan ausführen). Im Allgemeinen führt der Optimierer keinen nicht genehmigten Plan aus. Wenn der `apg_plan_mgmt.unapproved_plan_execution_threshold`-Parameter jedoch einen Kostenschwellenwert angibt, vergleicht der Optimierer die Kosten des nicht genehmigten Plans mit dem Schwellenwert. Falls die geschätzten Kosten niedriger als der Schwellenwert sind, führt der Optimierer den Plan aus. Weitere Informationen finden Sie unter apg_plan_mgmt.unapproved_plan_execution_threshold.
      - Wenn die geschätzten Kosten des Plans nicht unter dem Schwellenwert liegen, werden die anderen Attribute des Plans überprüft (8a).
   b. Wenn der Status des Plans etwas anderes als Nicht genehmigt ist, werden seine anderen Attribute überprüft (8a).

8. Der Optimierer wird keinen Plan verwenden, der deaktiviert ist. Das heißt, der Plan, dessen `enable`-Attribut auf ‚f' (false) festgelegt ist. Der Optimierer verwendet auch keinen Plan mit dem Status Abgelehnt.

   Der Optimierer kann keine Pläne verwenden, die nicht gültig sind. Pläne können mit der Zeit ungültig werden, wenn die Objekte, von denen sie abhängen, wie Indizes und Tabellenpartitionen, entfernt oder gelöscht werden.

   a. Falls die Anweisung über aktivierte und gültige bevorzugte Pläne verfügt, wählt der Optimierer den kostengünstigsten Plan aus den für diese SQL-Anweisung gespeicherten bevorzugten Plan aus. Der Optimierer führt dann den kostengünstigsten bevorzugten Plan aus.

   b. Wenn die Abrechnung keine aktivierten und gültigen Preferred-Pläne enthält, wird sie im nächsten Schritt bewertet (9).

9. Falls die Anweisung über aktivierte und gültige Genehmigte Pläne verfügt, wählt der Optimierer den kostengünstigsten Plan aus den für diese SQL-Anweisung gespeicherten Genehmigten Plan aus. Der Optimierer führt dann den kostengünstigsten genehmigten Plan aus.

   Wenn die Abrechnung keine gültigen und aktivierten Genehmigten Pläne enthält, verwendet der Optimierer den Mindestkostenplan (A. Optimierer-Plan ausführen).

Untersuchen von Aurora-PostgreSQL-Abfrageplänen in der dba_plans-Ansicht

Datenbankbenutzer und Administratoren, denen die `apg_plan_mgmt`-Rolle zugewiesen wurde, können die in `apg_plan_mgmt.dba_plans` gespeicherten Pläne anzeigen und verwalten. Der Administrator eines DB-Clusters von Aurora PostgreSQL (ein Benutzer mit `rds_superuser`-Berechtigungen) muss diese Rolle explizit den Datenbankbenutzern zuweisen, die mit der Abfrageplanverwaltung arbeiten müssen.

Die `apg_plan_mgmt`-Ansicht enthält den Planverlauf für alle verwalteten SQL-Anweisungen für jede Datenbank auf der Writer-Instance des DB-Clusters von Aurora PostgreSQL. In dieser Ansicht können Sie Pläne, ihren Status, wann sie zuletzt verwendet wurden und alle anderen relevanten Details untersuchen.

Wie in Normalisierung und der SQL-Hash erörtert, wird jeder verwaltete Plan anhand der Kombination aus einem SQL- und einem Plan-Hash-Wert identifiziert. Mit diesen IDs können Sie Tools wie Amazon RDS-Performance-Insights nutzen, um die Performance einzelner Pläne

nachzuverfolgen. Weitere Informationen über Performance-Insights finden Sie unter Verwenden von Amazon RDS-Performance-Insights.

Auflisten von verwalteten Plänen.

Mit der SELECT-Anweisung in der Ansicht `apg_plan_mgmt.dba_plans` listen Sie die verwalteten Pläne auf. Im folgenden Beispiel werden einige Spalten in der `dba_plans`-Ansicht aufgeführt, so z. B. die Spalte `status`, in der Sie sehen können, ob ein Plan genehmigt oder nicht genehmigt wurde.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

 sql_hash   | plan_hash |   status   | enabled | stmt_name
------------+-----------+------------+---------+------------
 1984047223 | 512153379 | Approved   | t       | rangequery
 1984047223 | 512284451 | Unapproved | t       | rangequery
 (2 rows)
```

Zur besseren Lesbarkeit listen die Abfrage und die angezeigte Ausgabe nur einige Spalten aus der `dba_plans`-Ansicht auf. Ausführliche Informationen finden Sie unter Referenz für die apg_plan_mgmt.dba_plans-Ansicht für die Aurora-PostgreSQL-kompatible Edition.

Verbessern von Aurora-PostgreSQL-Abfrageplänen

Verbessern Sie die Verwaltung von Abfrageplänen, indem Sie die Leistung des Plans bewerten und Pläne korrigieren. Weitere Informationen zum Verbessern Ihrer Abfragepläne finden Sie in den folgenden Themen.

Themen

- Auswerten der Performance von Plänen
- Reparieren von Plänen mit pg_hint_plan

Auswerten der Performance von Plänen

Nachdem der Optimierer Pläne als „Nicht genehmigt“ erfasst hat, können Sie mit der Funktion `apg_plan_mgmt.evolve_plan_baselines` Pläne hinsichtlich ihrer tatsächlichen Performance miteinander vergleichen. Je nach Ergebnis dieses Performance-Vergleichs können Sie den Status

eines Plans von „Nicht genehmigt“ zu „Genehmigt“ oder „Abgelehnt“ ändern. Alternativ besteht die Möglichkeit, mit der Funktion `apg_plan_mgmt.evolve_plan_baselines` einen Plan auf bestimmte Zeit zu deaktivieren, falls er Ihre Anforderungen nicht erfüllt.

Genehmigen besserer Pläne

Im folgenden Beispiel wird beschrieben, wie der Status verwalteter Pläne mit der Funktion `apg_plan_mgmt.evolve_plan_baselines` auf „Genehmigt“ geändert werden kann.

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
   plan_hash,
   min_speedup_factor := 1.0,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:     rangequery (1,10000)
NOTICE:     Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:     Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:     Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:     Unapproved -> Approved
evolve_plan_baselines
----------------------
0
(1 row)
```

In der Ausgabe wird ein Performance-Bericht für die`rangequery`-Anweisung mit Parameterbindungen von 1 und 10 000 angezeigt. Der neue nicht genehmigte Plan (`Baseline+1`) ist besser als der beste zuvor genehmigte Plan (`Baseline`). Rufen Sie die `Approved`-Ansicht auf, um zu sehen, ob der neue Plan nun den Status „`apg_plan_mgmt.dba_plans`“ aufweist.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
sql_hash  | plan_hash |  status  | enabled | stmt_name
------------+-----------+----------+---------+------------
1984047223 | 512153379 | Approved | t       | rangequery
1984047223 | 512284451 | Approved | t       | rangequery
(2 rows)
```

Der verwaltete Plan enthält jetzt zwei genehmigte Pläne, die zusammen die Plan-Baseline der Anweisung bilden. Sie können auch die Funktion „`apg_plan_mgmt.set_plan_status`“ aufrufen, um das Statusfeld eines Plans direkt auf `'Approved'`, `'Rejected'`, `'Unapproved'` oder `'Preferred'` festzulegen.

Ablehnen oder Deaktivieren langsamerer Pläne

Leiten Sie `'reject'` oder `'disable'` als Aktionsparameter an die Funktion `apg_plan_mgmt.evolve_plan_baselines` weiter, um Pläne abzulehnen oder zu deaktivieren. In diesem Beispiel wird jeder erfasste Plan mit dem Status „`Unapproved`“ deaktiviert, der um mindestens 10 Prozent langsamer als der beste Plan mit dem Status „`Approved`“ der Anweisung ist.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND   -- plan is Unapproved
origin = 'Automatic';       -- plan was auto-captured
```

Sie können den Status eines Plans auch direkt auf „Abgelehnt“ oder „Deaktiviert“ setzen. Rufen Sie die Funktion `true` auf, um das aktivierte Feld eines Plans direkt auf `false` oder `apg_plan_mgmt.set_plan_enabled` festzulegen. Mit der Funktion „`'Approved'`“ legen Sie das Statusfeld eines Plans direkt auf `'Rejected'`, `'Unapproved'`, `'Preferred'` oder `apg_plan_mgmt.set_plan_status` fest.

Verwenden Sie die `apg_plan_mgmt.validate_plans`-Funktion, um Pläne zu löschen, die nicht gültig sind und voraussichtlich ungültig bleiben werden. Mit dieser Funktion können Sie ungültige Pläne löschen oder deaktivieren. Weitere Informationen finden Sie unter Validieren von Plänen.

Reparieren von Plänen mit pg_hint_plan

Mit dem Abfrageoptimierer kann für jede Anweisung der bestmögliche Plan gefunden werden. In den meisten Fällen findet der Abfrageoptimierer in der Tat einen guten Plan. Manchmal gibt es jedoch einen weitaus besseren als den vom Optimierer erstellten Plan. Um zu gewährleisten, dass der Optimierer einen passenden Plan erstellt, wird zum einen die Verwendung der Erweiterung `pg_hint_plan`, zum anderen die Festlegung der Grand Unified Configuration (GUC)-Variablen in PostgreSQL empfohlen:

- `pg_hint_plan`-Erweiterung – Geben Sie mithilfe der `pg_hint_plan`-Erweiterung von PostgreSQL einen „Hinweis“, um die Arbeitsweise des Planers zu ändern. Informationen zur Installation und Verwendung der `pg_hint_plan`-Erweiterung finden Sie in der pg_hint_plan-Dokumentation.
- GUC-Variablen: Überschreiben Sie einen oder mehrere Kostenmodellparameter oder Parameter des Optimierers, wie z. B. `from_collapse_limit` oder `GEQO_threshold`.

Wenn Sie den Abfrageoptimierer mit einer dieser Methoden zur Verwendung eines Plans zwingen, können Sie auch die Abfrageplanverwaltung nutzen, um den neuen Plan zu erfassen und dessen Anwendung durchzusetzen.

Mit der Erweiterung `pg_hint_plan` können Sie die Join-Reihenfolge, die Join-Methoden oder die Zugriffspfade für eine SQL-Anweisung ändern. Sie nutzen einen SQL-Kommentar mit einer besonderen `pg_hint_plan`-Syntax, um die üblicherweise vom Optimierer angewendete Planerstellungsmethode zu ändern. Angenommen, die entsprechende SQL-Anweisung verfügt über eine Zwei-Wege-Verknüpfung.

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Der Optimierer wählt nun die Join-Reihenfolge (t1, t2) aus. Wir wissen jedoch, dass die Reihenfolge (t2, t1) schneller ist. Der folgende Hinweis zwingt den Optimierer dazu, die schnellere Join-Reihenfolge (t2, t1) zu wählen. Schließen Sie EXPLAIN ein, damit der Optimierer einen Plan für die SQL-Anweisung generiert, ohne jedoch die Anweisung auszuführen. (Ausgabe wird nicht angezeigt.)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Im Folgenden wird beschrieben, wie Sie verwende `pg_hint_plan`.

So ändern und erfassen Sie mit pg_hint_plan den vom Optimierer erstellten Plan:

1. Aktivieren Sie den manuellen Erfassungsmodus.

   ```
   SET apg_plan_mgmt.capture_plan_baselines = manual;
   ```

2. Geben Sie für die entsprechende SQL-Anweisung einen Hinweis an.

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Nach der Ausführung erfasst der Optimierer den Plan in der `apg_plan_mgmt.dba_plans`-Ansicht. Der erfasste Plan berücksichtigt nicht die Syntax des besonderen Kommentars `pg_hint_plan`, weil die Abfrageplanverwaltung die Anweisung durch das Entfernen führender Kommentare normalisiert.

3. Sie können die verwalteten Pläne über die `apg_plan_mgmt.dba_plans`-Ansicht anzeigen.

```
SELECT sql_hash, plan_hash, status, sql_text, plan_outline
FROM apg_plan_mgmt.dba_plans;
```

4. Legen Sie den Status des Plans auf fes `Preferred`. Dadurch wird sichergestellt, dass der Optimierer den Plan ausführt, anstatt ihn aus dem Satz an genehmigten Plänen auszuwählen, wenn der Minimalkostenplan nicht bereits `Approved` oder `Preferred` ist.

```
SELECT apg_plan_mgmt.set_plan_status(sql-hash, plan-hash, 'preferred' );
```

5. Deaktivieren Sie die manuelle Planerfassung und erzwingen Sie die Verwendung verwalteter Pläne.

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

Während der Ausführung der ursprünglichen SQL-Anweisung wählt der Optimierer einen Plan aus, der entweder `Approved` oder `Preferred` ist. Wenn der Minimalkostenplan weder den Status „`Approved`“ noch „`Preferred`“ aufweist, wählt der Optimierer den Plan mit Status „`Preferred`“ aus.

Löschen von Aurora-PostgreSQL-Abfrageplänen

Löschen Sie Ausführungspläne, die Sie nicht verwenden, oder Pläne, die nicht gültig sind. Weitere Informationen zum Löschen von Plänen finden Sie in den folgenden Abschnitten.

Themen

- Löschen von Plänen

- Validieren von Plänen

Löschen von Plänen

Pläne werden automatisch gelöscht, wenn sie seit über einem Monat, genauer gesagt 32 Tagen, nicht mehr verwendet wurden. Dies ist die Standardeinstellung des Parameters `apg_plan_mgmt.plan_retention_period`. Sie können den Aufbewahrungszeitraum des Plans zu einem längeren oder kürzeren Zeitraum ändern, beginnend mit dem Wert 1. Die Bestimmung der Anzahl von Tagen, seit ein Plan zuletzt verwendet wurde, wird berechnet, indem das `last_used`-Datum vom aktuellen Datum abgezogen wird. Das `last_used`-Datum ist das letzte Datum, an dem der Optimierer den Plan als Minimalkostenplan ausgewählt hat oder der Plan ausgeführt wurde. Das Datum wird in der `apg_plan_mgmt.dba_plans`-Ansicht für den Plan gespeichert.

Wir empfehlen Ihnen, Pläne zu löschen, die lange nicht mehr verwendet wurden oder nicht nützlich sind. Jeder Plan verfügt über ein `last_used`-Datum, das vom Optimierer aktualisiert wird, wenn er einen Plan ausführt oder den Plan als Minimalkostenplan für eine Anweisung auswählt. Überprüfen Sie die letzten `last_used`-Daten, um die Pläne zu identifizieren, die Sie ohne Bedenken löschen können.

Die folgende Abfrage gibt eine dreispaltige Tabelle mit der Gesamtanzahl der Pläne, der Anzahl der Pläne, die nicht gelöscht wurden, und der Anzahl der Pläne, die erfolgreich gelöscht wurden, zurück. Sie enthält eine verschachtelte Abfrage, die ein Beispiel dafür ist, wie Sie die `apg_plan_mgmt.delete_plan`-Funktion verwenden, um alle Pläne zu löschen, die in den letzten 31 Tagen nicht als Minimalkostenplan ausgewählt wurden und deren Status nicht `Rejected` lautet.

```
SELECT (SELECT COUNT(*) from apg_plan_mgmt.dba_plans) total_plans,
       COUNT(*) FILTER (WHERE result = -1) failed_to_delete,
       COUNT(*) FILTER (WHERE result = 0) successfully_deleted
       FROM (
            SELECT apg_plan_mgmt.delete_plan(sql_hash, plan_hash) as result
            FROM apg_plan_mgmt.dba_plans
            WHERE last_used < (current_date - interval '31 days')
            AND status <> 'Rejected'
            ) as dba_plans ;
```

```
 total_plans | failed_to_delete | successfully_deleted
-------------+------------------+----------------------
```

```
          3 |                0 |                    2
```

Weitere Informationen finden Sie unter apg_plan_mgmt.delete_plan.

Verwenden Sie die `apg_plan_mgmt.validate_plans`-Funktion, um Pläne zu löschen, die nicht gültig sind und voraussichtlich ungültig bleiben werden. Mit dieser Funktion können Sie ungültige Pläne löschen oder deaktivieren. Weitere Informationen finden Sie unter Validieren von Plänen.

> **Important**
>
> Wenn Sie Ihre irrelevanten Pläne nicht löschen, verfügen Sie möglicherweise über keinen gemeinsam genutzten Speicher mehr, der für die Abfrageplanverwaltung reserviert wird. Mit dem Parameter `apg_plan_mgmt.max_plans` legen Sie fest, wie viel Speicher für verwaltete Pläne zur Verfügung steht. Legen Sie diesen Parameter in der benutzerdefinierten Parametergruppe fest und starten Sie Ihre DB-Instance neu, damit die Änderungen wirksam werden. Informieren Sie sich über den Parameter apg_plan_mgmt.max_plans, um weitere Informationen hierzu zu erhalten.

Validieren von Plänen

Verwenden Sie die Funktion `apg_plan_mgmt.validate_plans`, um ungültige Pläne zu löschen oder zu deaktivieren.

Pläne können ungültig oder veraltet werden, wenn Objekte entfernt werden, von denen sie abhängig sind (z. B. ein Index oder eine Tabelle). Wenn das entfernte Objekt jedoch erneut erstellt wird, kann ein Plan auch nur für eine bestimmte Zeit ungültig sein. Falls ein ungültiger Plan zu einem späteren Zeitpunkt gültig werden kann, ist es möglicherweise sinnvoller, diesen nicht zu löschen, sondern zu deaktivieren oder einfach nichts zu unternehmen.

Über die Funktion `apg_plan_mgmt.validate_plans` suchen und löschen Sie alle ungültigen und während der vergangenen Woche nicht verwendeten Pläne. Gehen Sie hierzu folgendermaßen vor:

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');
```

Verwenden Sie die Funktion `apg_plan_mgmt.set_plan_enabled` zum direkten Aktivieren oder Deaktivieren eines Plans.

Exportieren und Importieren von verwalteten Plänen für Aurora PostgreSQL

Sie können Ihre verwalteten Pläne exportieren und in eine andere DB-Instance importieren.

So exportieren Sie verwaltete Pläne:

Autorisierte Benutzer können ein beliebiges Subset der `apg_plan_mgmt.plans`-Tabelle in eine andere Tabelle kopieren und dann mit dem Befehl `pg_dump` speichern. Im Folgenden wird ein Beispiel gezeigt.

```
CREATE TABLE plans_copy AS SELECT *
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

So importieren Sie verwaltete Pläne:

1. Kopieren Sie die .tar-Datei der exportierten verwalteten Pläne in das System, in dem die Pläne wiederhergestellt werden sollen.
2. Verwenden Sie den Befehl `pg_restore`, um die .tar-Datei in eine neue Tabelle zu kopieren.

   ```
   % pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
   ```
3. Führen Sie die Tabellen `plans_copy` und `apg_plan_mgmt.plans` entsprechend dem folgenden Beispiel zusammen.

   > **Note**
   >
   > In einigen Fällen verwenden Sie möglicherweise für die Sicherung eine bestimmte Version der Erweiterung `apg_plan_mgmt` und für die Wiederherstellung eine andere Version der Erweiterung. In diesem Fällen fallen die Spalten in der Plantabelle möglicherweise unterschiedlich aus. Geben Sie den Spalten in diesem Fall einen bestimmten Namen, anstatt SELECT* zu verwenden.

   ```
   INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy
   ```

```
ON CONFLICT ON CONSTRAINT plans_pkey
DO UPDATE SET
status = EXCLUDED.status,
enabled = EXCLUDED.enabled,
-- Save the most recent last_used date
--
last_used = CASE WHEN EXCLUDED.last_used > plans.last_used
THEN EXCLUDED.last_used ELSE plans.last_used END,
-- Save statistics gathered by evolve_plan_baselines, if it ran:
--
estimated_startup_cost = EXCLUDED.estimated_startup_cost,
estimated_total_cost = EXCLUDED.estimated_total_cost,
planning_time_ms = EXCLUDED.planning_time_ms,
execution_time_ms = EXCLUDED.execution_time_ms,
total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. Laden Sie die verwalteten Pläne erneut in den gemeinsam genutzten Speicher und entfernen Sie die Tabelle für die temporären Pläne.

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

Parameterreferenz für Aurora-PostgreSQL-Abfrageplanverwaltung

Sie können Ihre Einstellungen für die `apg_plan_mgmt`-Erweiterung unter Verwendung der in diesem Abschnitt aufgeführten Parameter vornehmen. Diese sind im benutzerdefinierten DB-Cluster-Parameter und in der DB-Parametergruppe verfügbar, die Ihrem Aurora PostgreSQL DB-Cluster zugeordnet ist. Diese Parameter steuern das Verhalten der Abfrageplanverwaltungsfunktion und deren Auswirkungen auf den Optimierer. Weitere Informationen zum Einrichten einer Abfrageverwaltung finden Sie unter Aktivieren der Abfrageplanverwaltung in Aurora PostgreSQL. Das Ändern der folgenden Parameter hat keine Auswirkung, wenn die `apg_plan_mgmt`-Erweiterung nicht wie in diesem Abschnitt beschrieben eingerichtet ist. Weitere Informationen zum Ändern von Parametern finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora und DB-Parametergruppen für DB-Instances von Amazon Aurora.

Parameter

- apg_plan_mgmt.capture_plan_baselines
- apg_plan_mgmt.plan_capture_threshold

- apg_plan_mgmt.explain_hashes
- apg_plan_mgmt.log_plan_enforcement_result
- apg_plan_mgmt.max_databases
- apg_plan_mgmt.max_plans
- apg_plan_mgmt.plan_hash_version
- apg_plan_mgmt.plan_retention_period
- apg_plan_mgmt.unapproved_plan_execution_threshold
- apg_plan_mgmt.use_plan_baselines
- auto_explain.hashes

apg_plan_mgmt.capture_plan_baselines

Erfasst Abfrageausführungspläne, die vom Optimierer für jede SQL-Anweisung generiert wurden, und speichert sie im der `dba_plans`-Ansicht. Standardmäßig beträgt die maximale Anzahl von Plänen, die gespeichert werden können, 10 000, wie im `apg_plan_mgmt.max_plans`-Parameter angegeben. Referenzinformationen finden Sie unter apg_plan_mgmt.max_plans.

Sie können diesen Parameter in der benutzerdefinierten DB-Cluster-Parametergruppe oder in der benutzerdefinierten DB-Parametergruppe festlegen. Um den Wert dieses Parameters zu ändern, ist kein Neustart erforderlich.

Standard	Zulässige Werte	Beschreibung
aus	Automatisch	Wenden Sie diese Einstellung entweder auf Sitzungsebene oder in einer Parametergruppe an, um Pläne zu erfassen, die zweimal oder öfter verwendet werden.
	Manuell	Wenden Sie diese Einstellung entweder auf Sitzungsebene oder in einer Parametergruppe an, um Pläne zu erfassen, die einmal oder öfter verwendet werden.
	aus	Deaktiviert die Planerfassung.

Weitere Informationen finden Sie unter Erfassung von Aurora-PostgreSQL-Ausführungsplänen.

apg_plan_mgmt.plan_capture_threshold

Gibt einen Schwellenwert an, sodass der Plan nicht in der Ansicht `apg_plan_mgmt.dba_plans` erfasst wird, wenn die Gesamtkosten des Abfrageausführungsplans unter dem Schwellenwert liegen.

Um den Wert dieses Parameters zu ändern, ist kein Neustart erforderlich.

Standard	Zulässige Werte	Beschreibung
0	0 – 1.79769e+308	Legt den Schwellenwert der Gesamtausführungskosten des Abfrageplans `apg_plan_mgmt` für die Erfassung von Plänen fest.

Weitere Informationen finden Sie unter Untersuchen von Aurora-PostgreSQL-Abfrageplänen in der dba_plans-Ansicht.

apg_plan_mgmt.explain_hashes

Gibt an, ob der `EXPLAIN [ANALYZE]` am Ende seiner Ausgabe `sql_hash` und `plan_hash` anzeigt. Um den Wert dieses Parameters zu ändern, ist kein Neustart erforderlich.

Standard	Zulässige Werte	Beschreibung
0	0 (aus)	EXPLAIN zeigt sql_hash und plan_hash ohne die Option true für die Hashes nicht an.
	1 (ein)	EXPLAIN zeigt sql_hash und plan_hash ohne die Option true für die Hashes an.

apg_plan_mgmt.log_plan_enforcement_result

Gibt an, ob die Ergebnisse aufgezeichnet werden müssen, um zu überprüfen, ob die von QPM verwalteten Pläne ordnungsgemäß verwendet werden. Wenn ein gespeicherter generischer Plan verwendet wird, werden keine Datensätze in die Protokolldateien geschrieben. Um den Wert dieses Parameters zu ändern, ist kein Neustart erforderlich.

Standard	Zulässige Werte	Beschreibung
Keine	Keine	Zeigt in den Protokolldateien kein Ergebnis der Plandurch setzung an.
	on_error	Zeigt nur das Ergebnis der Plandurchsetzung in Protokolldateien an, wenn QPM keine verwalteten Pläne verwendet.
	all	Zeigt alle Ergebnisse der Plandurchsetzung in Protokolldateien an, einschließlich Erfolgen und Fehlern.

apg_plan_mgmt.max_databases

Gibt die maximale Anzahl an Datenbanken in der Writer-Instance Ihres Aurora-PostgreSQL-Datenbank-Clusters an, die die Abfrageplanverwaltung nutzen können. Standardmäßig können bis zu 10 Datenbanken die Abfrageplanverwaltung verwenden. Wenn die Instance über mehr als 10 Datenbanken verfügt, können Sie den Wert dieser Einstellung ändern. Um herauszufinden, wie viele Datenbanken sich in einer bestimmten Instance befinden, stellen Sie eine Verbindung mit der Instance über `psql` her. Verwenden Sie dann den psql-Meta-Befehl, `\l`, um die Datenbanken aufzulisten.

Wenn Sie den Wert dieses Parameters ändern, müssen Sie die Instance neu starten, damit die Einstellung wirksam wird.

Standard	Zulässige Werte	Beschreibung
10	10-2147483647	Maximale Anzahl von Datenbanken, die die Abfrageplanverwalt ung auf der Instance verwenden können.

Sie können diesen Parameter in der benutzerdefinierten DB-Cluster-Parametergruppe oder in der benutzerdefinierten DB-Parametergruppe festlegen.

apg_plan_mgmt.max_plans

Legt die maximale Anzahl von SQL-Anweisungen fest, die die Abfrageplanverwaltung in der Ansicht `apg_plan_mgmt.dba_plans` anzeigen kann. Wir empfehlen, diesen Parameter für alle Aurora-PostgreSQL-Versionen auf `10000` oder höher zu setzen.

Sie können diesen Parameter in der benutzerdefinierten DB-Cluster-Parametergruppe oder in der benutzerdefinierten DB-Parametergruppe festlegen. Wenn Sie den Wert dieses Parameters ändern, müssen Sie die Instance neu starten, damit die Einstellung wirksam wird.

Standard	Zulässige Werte	Beschreibung
10000	10-2147483647	Maximale Anzahl von Plänen, die in der `apg_plan_mgmt.dba_plans` -Ansicht gespeichert werden können. Die Standardeinstellung für Aurora-PostgreSQL-Version 10 und älter ist 1000.

Weitere Informationen finden Sie unter Untersuchen von Aurora-PostgreSQL-Abfrageplänen in der dba_plans-Ansicht.

apg_plan_mgmt.plan_hash_version

Gibt die Anwendungsfälle an, für die die plan_hash-Berechnung konzipiert ist. Eine höhere Version von `apg_plan_mgmt.plan_hash_version` deckt den gesamten Funktionsumfang der niedrigeren Version ab. Version 3 deckt beispielsweise die von Version 2 unterstützten Anwendungsfälle ab.

Auf die Änderung des Werts dieses Parameters muss ein Aufruf von `apg_plan_mgmt.validate_plans('update_plan_hash')` folgen. Dadurch werden die plan_hash-Werte in jeder Datenbank, in der apg_plan_mgmt installiert ist, und Einträge in der Plantabelle aktualisiert. Weitere Informationen finden Sie unter Validieren von Plänen

Standard	Zulässige Werte	Beschreibung
1	1	Standardberechnung von plan_hash.
	2	plan_hash-Berechnung für die Unterstützung mehrerer Schemas geändert.
	3	plan_hash-Berechnung für die Unterstützung mehrerer Schemas und Unterstützung partitionierter Tabellen geändert.
	4	plan_hash-Berechnung für parallele Operatoren und zur Unterstützung von Materialisierungsknoten geändert.

apg_plan_mgmt.plan_retention_period

Gibt an, wie viele Tage lang Pläne in der `apg_plan_mgmt.dba_plans`-Ansicht aufbewahrt werden, danach werden sie automatisch gelöscht. Standardmäßig wird ein Plan gelöscht, wenn 32 Tage seit der letzten Verwendung des Plans vergangen sind (Die `last_used`-Spalte in der `apg_plan_mgmt.dba_plans`-Ansicht). Sie können diese Einstellung auf eine beliebige Zahl ändern (1 und höher).

Wenn Sie den Wert dieses Parameters ändern, müssen Sie die Instance neu starten, damit die Einstellung wirksam wird.

Standard	Zulässige Werte	Beschreibung
32	1-2147483647	Maximale Anzahl von Tagen seit der letzten Nutzung eines Plans, bevor er gelöscht wird.

Weitere Informationen finden Sie unter Untersuchen von Aurora-PostgreSQL-Abfrageplänen in der dba_plans-Ansicht.

apg_plan_mgmt.unapproved_plan_execution_threshold

Gibt einen Kostenschwellenwert an, unterhalb dessen ein nicht genehmigter Plan vom Optimierer verwendet werden kann. Der Schwellenwert lautet standardmäßig 0, nicht genehmigte Pläne werden vom Optimierer also nicht ausgeführt. Wenn dieser Parameter auf einen belanglos niedrigen Kostenschwellenwert wie 100 gesetzt wird, wird der Overhead für die Plandurchsetzung bei trivialen Plänen vermieden. Sie können diesen Parameter auch unter Verwendung des reaktiven Planverwaltungstils auf einen extrem hohen Wert wie 10000000 festlegen. Auf diese Weise kann der Optimierer alle ausgewählten Pläne ohne Overhead für die Plandurchsetzung verwenden. Wenn jedoch ein fehlerhafter Plan gefunden wird, können Sie diesen manuell als „abgelehnt“ markieren, sodass er beim nächsten Mal nicht verwendet wird.

Der Wert dieses Parameters stellt eine Kostenschätzung für die Ausführung eines bestimmten Plans dar. Wenn ein nicht genehmigter Plan unter diesen geschätzten Kosten liegt, verwendet der Optimierer ihn für die SQL-Anweisung. Sie können erfasste Pläne und ihren Status (Genehmigt, Nicht genehmigt) in der `dba_plans`-Ansicht anzeigen. Weitere Informationen hierzu finden Sie unter Untersuchen von Aurora-PostgreSQL-Abfrageplänen in der dba_plans-Ansicht.

Um den Wert dieses Parameters zu ändern, ist kein Neustart erforderlich.

Standard	Zulässige Werte	Beschreibung
0	0-2147483647	Geschätzte Plankosten, unter denen ein nicht genehmigter Plan verwendet wird.

Weitere Informationen finden Sie unter Verwenden von Aurora-PostgreSQL-Plänen.

apg_plan_mgmt.use_plan_baselines

Gibt an, dass der Optimierer einen der genehmigten Pläne verwenden soll, der in der `apg_plan_mgmt.dba_plans`-Ansicht erfasst und gespeichert ist. Standardmäßig ist dieser Parameter deaktiviert (false), was dazu führt, dass der Optimierer den von ihm generierten Mindestkostenplan ohne weitere Bewertung verwendet. Wenn Sie diesen Parameter aktivieren (ihn auf true setzen), muss der Optimierer einen Abfrageausführungsplan für die Anweisung aus seiner Plan-Baseline auswählen. Weitere Informationen finden Sie unter Verwenden von Aurora-PostgreSQL-Plänen. Ein Bild, das diesen Prozess detailliert beschreibt, finden Sie unter Erfahren Sie, wie der Optimierer bestimmt, welche Pläne ausgeführt werden..

Sie können diesen Parameter in der benutzerdefinierten DB-Cluster-Parametergruppe oder in der benutzerdefinierten DB-Parametergruppe festlegen. Um den Wert dieses Parameters zu ändern, ist kein Neustart erforderlich.

Standard	Zulässige Werte	Beschreibung
false	true	Verwenden Sie einen genehmigten, bevorzugten oder nicht genehmigten Plan aus dem `apg_plan_mgmt.dba_plans` . Wenn keiner von ihnen alle Bewertungskriterien für den Optimierer erfüllt, kann er seinen eigenen generierten Mindestko stenplan verwenden. Weitere Informationen finden Sie unter Erfahren Sie, wie der Optimierer bestimmt, welche Pläne ausgeführt werden..
	false	Verwenden Sie den vom Optimierer generierten Minimalko stenplan.

Sie können die Reaktionszeiten verschiedener erfasster Pläne auswerten und den Planstatus nach Bedarf ändern. Weitere Informationen finden Sie unter Verbessern von Aurora-PostgreSQL-Abfrageplänen.

auto_explain.hashes

Gibt an, ob die auto_explain-Ausgabe sql_hash und plan_hash anzeigt. Um den Wert dieses Parameters zu ändern, ist kein Neustart erforderlich.

Standard	Zulässige Werte	Beschreibung
0 (aus)	0 (aus)	Das Ergebnis von `auto_explain` zeigt `sql_hash` und `plan_hash` nicht an.
	1 (ein)	Das Ergebnis von `auto_explain` zeigt `sql_hash` und `plan_hash` an.

Funktionsreferenz für die Aurora-PostgreSQL-Abfrageplanverwaltung

Die Erweiterung `apg_plan_mgmt` stellt die folgenden Funktionen bereit:

Funktionen

- apg_plan_mgmt.copy_outline
- apg_plan_mgmt.delete_plan
- apg_plan_mgmt.evolve_plan_baselines
- apg_plan_mgmt.get_explain_plan
- apg_plan_mgmt.plan_last_used
- apg_plan_mgmt.reload
- apg_plan_mgmt.set_plan_enabled
- apg_plan_mgmt.set_plan_status
- apg_plan_mgmt.update_plans_last_used
- apg_plan_mgmt.validate_plans

apg_plan_mgmt.copy_outline

Kopiert einen bestimmten SQL-Plan-Hash und eine Plangliederung in einen Ziel-SQL-Plan-Hash und eine Gliederung, wodurch der Plan-Hash und die Gliederung des Ziels überschrieben werden. Diese Funktion ist in `apg_plan_mgmt` 2.3 und höheren Versionen verfügbar.

Syntax

```
apg_plan_mgmt.copy_outline(
    source_sql_hash,
    source_plan_hash,
    target_sql_hash,
    target_plan_hash,
    force_update_target_plan_hash
)
```

Rückgabewert

Gibt 0 zurück, wenn das Kopieren erfolgreich ist. Löst Ausnahmen bei ungültigen Eingaben aus.

Parameter

Parameter	Beschreibung
`source_sql_hash`	Die `sql_hash`-ID, die mit `plan_hash` verknüpft ist, um sie in die Zielabfrage zu kopieren.
`source_plan_hash`	Die `plan_hash` -ID, die in die Zielabfrage kopiert werden soll.
`target_sql_hash`	Die `sql_hash`-ID der Abfrage, die mit dem Hash und der Gliederung des Quellplans aktualisiert werden soll.
`target_plan_hash`	Die `plan_hash` -ID der Abfrage, die mit dem Hash und der Gliederung des Quellplans aktualisiert werden soll.
`force_update_target_plan_hash`	(Optional) Die `target_plan_hash` -ID der Abfrage wird aktualisiert, auch wenn der

Parameter	Beschreibung
	Quellplan für den `target_sql_hash` nicht reproduzierbar ist. Bei der Einstellung „true“ kann die Funktion verwendet werden, um Pläne über Schemas hinweg zu kopieren, bei denen die Namen und Spalten der Beziehungen konsistent sind.

Nutzungshinweise

Mit dieser Funktion können Sie einen Plan-Hash und eine Plangliederung kopieren, die Hinweise auf andere, ähnliche Anweisungen verwendet, und Ihnen so die Verwendung von Inline-Hinweisanweisungen bei jedem Vorkommen in den Zielanweisungen erspart. Wenn die aktualisierte Zielabfrage zu einem ungültigen Plan führt, löst diese Funktion einen Fehler aus und macht den Aktualisierungsversuch rückgängig.

apg_plan_mgmt.delete_plan

Löschen Sie einen verwalteten Plan.

Syntax

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

Rückgabewert

Gibt bei erfolgreichen Löschvorgängen „0“, bei gescheiterten Löschvorgängen „-1“ zurück.

Parameter

Parameter	Beschreibung
`sql_hash`	Die `sql_hash`-ID der verwalteten SQL-Anweisung des Plans.

Parameter	Beschreibung
`plan_hash`	Die `plan_hash` -ID des verwalteten Plans.

apg_plan_mgmt.evolve_plan_baselines

Dieser Parameter überprüft, ob ein bereits genehmigter Plan oder ein vom Optimierer erkannter Minimalkostenplan schneller ist.

Syntax

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

Rückgabewert

Die Anzahl der Pläne, die nicht schneller als der beste genehmigte Plan waren.

Parameter

Parameter	Beschreibung
`sql_hash`	Die `sql_hash`-ID der verwalteten SQL-Anweisung des Plans.
`plan_hash`	Die `plan_hash` -ID des verwalteten Plans. Verwenden Sie NULL, um alle Pläne mit demselben `sql_hash`-ID-Wert anzugeben.
`min_speed up_factor`	Der Mindestbeschleunigungsfaktor kann angeben, um wie viel schneller ein Plan im Vergleich zum besten der bereits genehmigten Pläne sein muss, um genehmigt zu werden. Alternativ kann dieser Faktor auch angeben, um wie viel langsamer ein Plan sein muss, um abgelehnt oder deaktiviert zu werden. Dieser Wert ist ein positiver Gleitkommawert.

Parameter	Beschreibung
`action`	Bezeichnet die Aktion, die von der Funktion ausgeführt wird. Im Folgenden sind Beispiele für gültige Werte aufgeführt. Die Groß-/Kle inschreibung muss nicht beachtet werden. • `'disable'` – Deaktiviert alle übereinstimmenden Pläne, die die Kriterien des Mindestbeschleunigungsfaktors nicht erfüllen. • `'approve'` – Aktiviert alle übereinstimmenden Pläne, die die Kriterien des Mindestbeschleunigungsfaktors erfüllen, und setzt ihren Status auf `approved`. • `'reject'` – Setzt den Status aller übereinstimmenden Pläne, die die Kriterien des Mindestbeschleunigungsfaktors nicht erfüllen, auf `rejected`. • NULL – Die Funktion gibt die Anzahl der Pläne zurück, die zu keiner Performance-Verbesserung führen, weil sie die Kriterien des Mindestbe schleunigungsfaktors nicht erfüllen.

Nutzungshinweise

Je nachdem, ob die kombinierte Planungs- und Ausführungszeit eines Plans kürzer ist als beim schnellsten genehmigten Plan, können Sie anhand eines von Ihnen festlegbaren Faktors den Status bestimmter Pläne auf „Genehmigt“, „Abgelehnt“ oder „Deaktiviert“ setzen. Der Aktionsparameter kann auf `'approve'` oder `'reject'` gesetzt werden, um einen Plan automatisch zu genehmigen oder abzulehnen, wenn er die Performance-Kriterien erfüllt. Alternativ kann auch die Einstellung '' (leere Zeichenfolge) gewählt werden, um den Performance-Vergleich durchzuführen und einen Bericht zu erstellen, wobei in diesem Fall keine Aktion ausgeführt wird.

Sie können vermeiden, dass die Funktion `apg_plan_mgmt.evolve_plan_baselines` bei einem Plan unnötigerweise erneut ausgeführt wird. Dazu schränken Sie die Auswahl der Pläne auf die kürzlich erstellten, nicht genehmigten Pläne ein. Das Ausführen der Funktion `apg_plan_mgmt.evolve_plan_baselines` kann auch vermieden werden, wenn ein genehmigter Plan über einen aktuellen `last_verified`-Zeitstempel verfügt.

Führen Sie einen Performance-Vergleich durch, um die kombinierte Planungs- und Ausführungszeit aller Baseline-Pläne miteinander zu vergleichen. In einigen Fällen gibt es für eine Anweisung nur einen Plan, der dann auch genehmigt wird. Vergleichen Sie in einem solchen Fall die kombinierte

Planungs- und Ausführungszeit des Plans mit der kombinierten Planungs- und Ausführungszeit, die sich ergibt, wenn kein Plan verwendet wird.

Der inkrementelle Vorteil (oder Nachteil) aller Pläne wird in der `apg_plan_mgmt.dba_plans`-Ansicht in der Spalte `total_time_benefit_ms` aufgezeichnet. Ist dieser Wert positiv, führt die Aufnahme dieses Plans in die Baseline zu einem messbaren Performance-Vorteil.

Zusätzlich zur Planungs- und Ausführungszeit aller ausgewählten Pläne wird in der Spalte `last_verified` der `apg_plan_mgmt.dba_plans`-Ansicht auch der aktuelle `current_timestamp` angezeigt. Durch den Zeitstempel `last_verified` kann das erneute Ausführen dieser Funktion bei einem Plan vermieden werden, dessen Performance erst kürzlich verifiziert wurde.

apg_plan_mgmt.get_explain_plan

Erzeugt den Text einer `EXPLAIN`-Anweisung für die angegebene SQL-Anweisung.

Syntax

```
apg_plan_mgmt.get_explain_plan(
    sql_hash,
    plan_hash,
    [explainOptionList]
)
```

Rückgabewert

Gibt Laufzeitstatistiken für die angegebenen SQL-Anweisungen zurück. Verwenden ohne `explainOptionList`, um einen einfachen `EXPLAIN`-Plan zurückzugeben.

Parameter

Parameter	Beschreibung
`sql_hash`	Die `sql_hash`-ID der verwalteten SQL-Anweisung des Plans.
`plan_hash`	Die `plan_hash` -ID des verwalteten Plans.
`explainOptionList`	Eine durch Kommas getrennte Liste von Explain-Optionen. Gültige Werte sind

Parameter	Beschreibung
	`'analyze'` , `'verbose'` , `'buffers'` , `'hashes'` und `'format json'`. Wenn `explainOptionList` NULL ist oder eine leere Zeichenfolge („), generiert diese Funktion eine `EXPLAIN`-Anweisung, ohne irgendwelche Statistiken.

Nutzungshinweise

Für `explainOptionList` können Sie eine der gleichen Optionen verwenden, die Sie mit einer `EXPLAIN`-Anweisung verwenden würden. Der Aurora-PostgreSQL-Optimizer verkettet die Liste der Optionen, die Sie für die `EXPLAIN`-Anweisung bereitstellen.

apg_plan_mgmt.plan_last_used

Gibt das `last_used`-Datum des angegebenen Plans aus dem gemeinsam genutzten Speicher zurück.

> **Note**
>
> Der Wert im gemeinsam genutzten Speicher ist auf der primären DB-Instance im DB-Cluster immer aktuell. Der Wert wird nur periodisch in die `last_used`-Spalte der `apg_plan_mgmt.dba_plans`-Ansicht gespült.

Syntax

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

Rückgabewert

Gibt das Datum des Typs `last_used` zurück.

Parameter

Parameter	Beschreibung
`sql_hash`	Die `sql_hash`-ID der verwalteten SQL-Anwei sung des Plans.
`plan_hash`	Die `plan_hash` -ID des verwalteten Plans.

apg_plan_mgmt.reload

Laden Sie Pläne in der `apg_plan_mgmt.dba_plans`-Ansicht erneut in den gemeinsam genutzten Speicher.

Syntax

```
apg_plan_mgmt.reload()
```

Rückgabewert

Keine.

Parameter

Keine.

Nutzungshinweise

Rufen Sie `reload` in den folgenden Fällen auf:

- Verwenden Sie diesen Parameter, um den gemeinsam genutzten Speicher eines schreibgeschützten Replica sofort zu aktualisieren. So müssen Sie nicht warten, bis neue Pläne an das Replica übermittelt werden.
- Verwenden Sie diesen Parameter auch nach dem Import verwalteter Pläne.

apg_plan_mgmt.set_plan_enabled

Aktivieren oder deaktivieren Sie einen verwalteten Plan.

Syntax

```
apg_plan_mgmt.set_plan_enabled(
    sql_hash,
    plan_hash,
    [true | false]
)
```

Rückgabewert

Gibt bei erfolgreicher Einstellung „0“ und „-1“ zurück, wenn die Einstellung nicht übernommen wurde.

Parameter

Parameter	Beschreibung
`sql_hash`	Die `sql_hash`-ID der verwalteten SQL-Anweisung des Plans.
`plan_hash`	Die `plan_hash` -ID des verwalteten Plans.
`enabled`	Boolescher Wert („true“ oder „false“) • Durch einen Wert des Typs `true` wird der Plan aktiviert. • Durch einen Wert des Typs `false` wird der Plan deaktiviert.

apg_plan_mgmt.set_plan_status

Legen Sie den Status eines verwalteten Plans auf `Approved`, `Unapproved`, `Rejected` oder `Preferred` fest.

Syntax

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
    status
)
```

Rückgabewert

Gibt bei erfolgreicher Einstellung „0“ und „-1“ zurück, wenn die Einstellung nicht übernommen wurde.

Parameter

Parameter	Beschreibung
`sql_hash`	Die `sql_hash`-ID der verwalteten SQL-Anweisung des Plans.
`plan_hash`	Die `plan_hash` -ID des verwalteten Plans.
`status`	Eine Zeichenfolge mit einem der folgenden Werte: • `'Approved'` • `'Unapproved'` • `'Rejected'` • `'Preferred'` Der von Ihnen verwendete Fall spielt keine Rolle, aber der Statuswert wird in der `apg_plan_mgmt.dba_plans` -Ansicht auf anfängliche Großbuchstaben gesetzt. Weitere Informationen zu diesen Werten finden Sie unter `status`. Referenz für die apg_plan_mgmt.dba_plans-Ansicht für die Aurora-PostgreSQL-kompatible Edition

apg_plan_mgmt.update_plans_last_used

Aktualisiert die Plantabelle sofort mit dem `last_used`-Datum, das im freigegebenen Speicher gespeichert ist.

Syntax

```
apg_plan_mgmt.update_plans_last_used()
```

Rückgabewert

Keine.

Parameter

Keine.

Nutzungshinweise

Rufen Sie `update_plans_last_used` auf, um sicherzustellen, dass Abfragen für die Spalte `dba_plans.last_used` die neusten Informationen verwenden. Wenn das `last_used`-Datum nicht sofort aktualisiert wird, aktualisiert ein Hintergrundprozess die Plantabelle einmal stündlich mit dem `last_used`-Datum (standardmäßig).

Wenn beispielsweise eine Anweisung mit einem bestimmten `sql_hash` langsam ausgeführt wird, können Sie feststellen, welche Pläne für diese Anweisung seit Beginn der Leistungsregression ausgeführt wurden. Leeren Sie dazu zuerst die Daten im freigegebenen Speicher auf die Festplatte, damit die `last_used`-Daten aktuell sind, und fragen Sie dann alle Pläne des `sql_hash` der Anweisung mit der Leistungsregression ab. Stellen Sie in der Abfrage sicher, dass das Datum `last_used` größer oder gleich dem Datum ist, an dem die Leistungsregression begann. Die Abfrage identifiziert den Plan oder die Gruppe von Plänen, die für die Leistungsregression verantwortlich sein könnten. Sie können `apg_plan_mgmt.get_explain_plan` verwenden, wenn `explainOptionList` auf `verbose, hashes` gesetzt ist. Sie können `apg_plan_mgmt.evolve_plan_baselines` auch verwenden, um den Plan und alle alternativen Pläne zu analysieren, die möglicherweise eine bessere Leistung erbringen.

Die `update_plans_last_used`-Funktion wirkt sich nur auf die primäre DB-Instance des DB-Clusters aus.

apg_plan_mgmt.validate_plans

Überprüfen Sie, ob der Optimierer immer noch eine Neuerstellung von Plänen durchführen kann. Der Optimierer überprüft die Pläne `Approved`, `Unapproved` und `Preferred` dahingehend, ob der Plan aktiviert oder deaktiviert ist. `Rejected`-Pläne werden nicht validiert. Alternativ können Sie auch die Funktion `apg_plan_mgmt.validate_plans` verwenden, um ungültige Pläne zu löschen oder zu deaktivieren.

Syntax

```
apg_plan_mgmt.validate_plans(
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
```

```
    action)
```

Rückgabewert

Die Anzahl ungültiger Pläne

Parameter

Parameter	Beschreibung
`sql_hash`	Die `sql_hash`-ID der verwalteten SQL-Anweisung des Plans.
`plan_hash`	Die `plan_hash` -ID des verwalteten Plans. Verwenden Sie NULL, um alle Pläne für denselben `sql_hash`-ID-Wert anzugeben.
`action`	Die Aktion, die die Funktion für ungültige Pläne ausführen wird. Gültige Zeichenfolgenwerte sind unter anderem die folgenden: Die Groß-/Kleinschreibung muss nicht beachtet werden. • `'disable'` – Alle ungültigen Pläne werden deaktiviert. • `'delete'` – Alle ungültigen Pläne werden gelöscht. • `'update_plan_hash'` – Aktualisiert die `plan_hash` -ID für Pläne, die nicht genau reproduziert werden können. Damit können Sie auch einen Plan korrigieren, indem Sie die SQL-Anweisung neu schreiben. Anschließend können Sie den korrigierten Plan als `Approved`-Plan für die ursprüngliche SQL-Anweisung registrieren. • NULL – Diese Funktion gibt die Anzahl der ungültigen Pläne zurück. Keine weitere Aktion wird ausgeführt. • '' – Durch eine leere Zeichenfolge wird eine Nachricht mit der Anzahl gültiger und ungültiger Pläne erstellt. Alle anderen Werte werden wie die leere Zeichenfolge behandelt.

Nutzungshinweise

Mit dem Formular `validate_plans(action)` validieren Sie alle verwalteten Pläne für alle verwalteten Anweisungen in der gesamten `apg_plan_mgmt.dba_plans`-Ansicht.

Nutzen Sie das Formular `validate_plans(sql_hash, plan_hash, action)`, um einen mit `plan_hash` angegebenen verwalteten Plan für eine mit `sql_hash` spezifizierte verwaltete Anweisung zu validieren.

Mit dem Formular `validate_plans(sql_hash, NULL, action)` können Sie alle verwalteten Pläne für die verwaltete Anweisung validieren, die über den Parameter `sql_hash` spezifiziert wird.

Referenz für die apg_plan_mgmt.dba_plans-Ansicht für die Aurora-PostgreSQL-kompatible Edition

Die `apg_plan_mgmt.dba_plans`-Ansicht enthält die folgenden Spalten mit Planinformationen:

dba_plans-Spalte	Beschreibung
`cardinality_error`	Hierbei handelt es sich um ein Maß zur Angabe des Fehlers zwischen der geschätzten und der tatsächlichen Kardinalität. Bei der Kardinalität handelt es sich um die Anzahl der Tabellenzeilen, die der Plan verarbeiten soll. Bei einem schwerwiegenden Kardinalitätsfehler besteht eine erhöhte Wahrscheinlichkeit dafür, dass ein Plan nicht optimal funktioniert. Diese Spalte ist durch die Funktion apg_plan_mgmt.evolve_plan_baselines ausgefüllt.
`compatibility_level`	Dieser Parameter zeigt an, wann ein Abfrageplan zuletzt validiert wurde. In den Aurora-PostgreSQL-Versionen 12.19, 13.15, 14.12, 15.7, 16.3 und höher wird die Aurora-Versionsnummer angezeigt. Bei früheren Versionen wird eine funktionsspezifische Versionsnummer angezeigt. **Note** Belassen Sie diesen Parameterwert bei der Standardeinstellung. Aurora PostgreSQL legt diesen Wert automatisch fest und aktualisiert ihn.
`created_by`	Hier sehen Sie den authentifizierten Benutzer (`session_user`), der den Plan erstellt hat.

dba_plans-Spalte	Beschreibung
`enabled`	Gibt an, ob der Plan aktiviert oder deaktiviert ist. Alle Pläne sind standardmäßig aktiviert. Sie können Pläne deaktivieren, damit sie nicht vom Optimierer verwendet werden können. Verwenden Sie die Funktion apg_plan_mgmt.set_plan_enabled, um diesen Wert zu ändern.
`environment_variables`	Gibt Aufschluss über die Parameter und Werte der PostgreSQL Grand Unified Configuration (GUC), die vom Optimierer zum Zeitpunkt der Planerfassung überschrieben wurden.
`estimated_startup_cost`	Diese Spalte bezieht sich auf die geschätzten Einrichtungskosten für den Optimierer, bevor dieser Tabellenzeilen übermittelt.
`estimated_total_cost`	Diese Spalte informiert über die geschätzten Optimiererkosten für das Übermitteln der letzten Tabellenzeile.
`execution_time_benefit_ms`	Die Ausführungszeitersparnis (in Millisekunden), die beim Aktivieren des Plans erzielt wird Diese Spalte ist durch die Funktion apg_plan_mgmt.evolve_plan_baselines ausgefüllt.
`execution_time_ms`	Diese Spalte informiert über die geschätzte Laufzeit des Plans (in Millisekunden). Diese Spalte ist durch die Funktion apg_plan_mgmt.evolve_plan_baselines ausgefüllt.
`has_side_effects`	Dieser Wert gibt an, dass die SQL-Anweisung eine Data Manipulation Language (DML)- oder eine SELECT-Anweisung ist, die eine Funktion des Typs TEMPORÄR enthält.

dba_plans-Spalte	Beschreibung
`last_used`	Dieser Wert wird auf das aktuelle Datum aktualisiert, wenn der Plan ausgeführt wird oder als Minimalkostenplan des Abfrageoptimierers fungiert. Dieser Wert wird im gemeinsam genutzten Speicher gespeichert und regelmäßig an den Datenträger übertragen. Um den neuesten Wert zu erhalten, rufen Sie durch das Ausführen der Funktion `apg_plan_mgmt.plan_last_used(sql_hash, plan_hash)` das Datum aus dem gemeinsam genutzten Speicher ab, anstatt den Wert `last_used` zu nutzen. Weitere Informationen erhalten Sie, indem Sie sich über den apg_plan_mgmt.plan_retention_period-Parameter informieren.
`last_validated`	Informiert über den letzten Zeitpunkt, an dem verifiziert wurde, dass der Plan entweder über die Funktion apg_plan_mgmt.validate_plans oder die Funktion apg_plan_mgmt.evolve_plan_baselines neu erstellt werden könnte.
`last_verified`	Informiert über den letzten Zeitpunkt, an dem ein Plan über die Funktion apg_plan_mgmt.evolve_plan_baselines als bester Plan für die angegebenen Parameter ermittelt wurde.
`origin`	Gibt an, wie der Plan mithilfe des Parameters apg_plan_mgmt.capture_plan_baselines erfasst wurde. Gültige Werte sind unter anderem: `M`: Der Plan wurde mit manueller Planerfassung erfasst. `A`: Der Plan wurde mit automatischer Planerfassung erfasst.
`param_list`	Die Parameterwerte, die an die Anweisung gesendet wurden, falls es sich um eine vorbereitete Anweisung handelt.
`plan_created`	Datum und Uhrzeit der Planerstellung
`plan_hash`	Die ID des Plans. Durch die Kombination von `plan_hash` und `sql_hash` wird ein bestimmter Plan eindeutig identifiziert.

dba_plans-Spalte	Beschreibung
`plan_outline`	Eine datenbankunabhängige Darstellung des Plans, mit der der tatsächliche Ausführungsplan neu erstellt wird. Die Operatoren in der Strukturansicht entsprechen den Operatoren in der EXPLAIN-Ausgabe.
`planning_time_ms`	Die tatsächliche Zeit (in Millisekunden), die zum Ausführen des Planers benötigt wird. Diese Spalte ist durch die Funktion apg_plan_mgmt.evolve_plan_baselines ausgefüllt.
`queryId`	Ein von der Erweiterung `pg_stat_statements` berechneter Anweisungs-Hash. Hierbei handelt es sich nicht um eine stabile oder datenbankunabhängige ID, weil dieser Hash von Objekt-IDs (OIDs) abhängig ist. Der Wert wird `0` sein, wenn `compute_query_id` bei der Erfassung des Abfrageplans auf `off` gesetzt ist.
`sql_hash`	Ein Hash-Wert des SQL-Anweisungstexts, normalisiert und ohne Literale
`sql_text`	Der vollständige Text der SQL-Anweisung.

dba_plans-Spalte	Beschreibung
`status`	Informiert über den Status eines Plans. Der Status bestimmt, wie ein Plan vom Optimierer verwendet wird. Im Folgenden sind Beispiele für gültige Werte aufgeführt. • `Approved` – ein nutzbarer Plan, der vom Optimierer ausgeführt werden kann. Der Optimierer wählt aus dem Satz genehmigter Pläne (Baseline) einer verwalteten Anweisung den kostengünstigsten Plan zur Ausführung aus. Verwenden Sie die Funktion apg_plan_mgmt.evolve_plan_baselines, um einen Plan auf „Genehmigt“ zurückzusetzen. • `Unapproved` – ein erfasster Plan, der nicht zur Verwendung verifiziert wurde. Weitere Informationen finden Sie unter Auswerten der Performance von Plänen. • `Rejected` – ein Plan, der nicht vom Optimierer verwendet wird. Weitere Informationen finden Sie unter Ablehnen oder Deaktivieren langsamerer Pläne. • `Preferred` – ein Plan, der von Ihnen für eine verwaltete Anweisung als bevorzugter Plan bestimmt wurde. Wenn der Minimalkostenplan des Optimierers kein genehmigter oder bevorzugter Plan ist, können Sie den Overhead der Plandurchführung reduzieren. Dazu können Sie einer Teilmenge der genehmigten Pläne den Status „“ zuweise `Preferred` . Wenn der Minimalkostenplan des Optimierers kein `Approved`-Plan ist, erhält ein `Preferred` -Plan den Vorzug vor einem `Approved`-Plan. Verwenden Sie die Funktion „`Preferred` “, um einen Plan auf „apg_plan_mgmt.set_plan_status“ zurückzusetzen.
`stmt_name`	Der Name der SQL-Anweisung innerhalb einer PREPARE-Anweisung. Dieser Wert ist eine leere Zeichenfolge für eine namenlose vorbereitete Anweisung. Dieser Wert beträgt bei nicht vorbereiteten Anweisungen NULL.

dba_plans-Spalte	Beschreibung
`total_time_benefit_ms`	Die gesamte Zeitersparnis (in Millisekunden), die beim Aktivieren dieses Plans erzielt wird. Dieser Wert berücksichtigt sowohl die Planungs- als auch die Ausführungszeit. Ist dieser Wert negativ, sind mit dem Aktivieren dieses Plans Nachteile verbunden. Diese Spalte ist durch die Funktion apg_plan_mgmt.evolve_plan_baselines ausgefüllt.

Erweiterte Funktionen der Abfrageplanverwaltung

Im Folgenden finden Sie Informationen zu den erweiterten Funktionen der Abfrageplanverwaltung (Query Plan Management, QPM) in Aurora PostgreSQL:

Themen

- Erfassung von Aurora-PostgreSQL-Ausführungsplänen
- Unterstützung der Tabellenpartition

Erfassung von Aurora-PostgreSQL-Ausführungsplänen

QPM (Query Plan Management) ermöglicht es Ihnen, die von Aurora Replicas generierten Abfragepläne zu erfassen und sie in der primären DB-Instance des Aurora-DB-Clusters zu speichern. Sie können die Abfragepläne von allen Aurora Replicas sammeln und eine Reihe optimaler Pläne in einer zentralen persistenten Tabelle auf der primären Instance verwalten. Sie können diese Pläne dann bei Bedarf auf andere Replicas anwenden. Dies hilft Ihnen, die Stabilität der Ausführungspläne aufrechtzuerhalten und die Leistung der Abfragen in den DB-Clustern und Engine-Versionen zu verbessern.

Themen

- Voraussetzungen
- Verwaltung der Planerfassung für Aurora Replicas
- Fehlerbehebung

Voraussetzungen

Aktivieren Sie **`capture_plan_baselines parameter`** in Aurora Replica – Setzen Sie den `capture_plan_baselines`-Parameter auf automatisch oder manuell, um Pläne in Aurora Replicas zu erfassen. Weitere Informationen finden Sie unter apg_plan_mgmt.capture_plan_baselines.

Installieren Sie die postgres_fdw-Erweiterung – Sie müssen die `postgres_fdw`-Foreign-Data-Wrapper-Erweiterung installieren, um Pläne in Aurora Replicas zu erfassen. Führen Sie den folgenden Befehl in jeder Datenbank aus, um die Erweiterung zu installieren.

```
postgres=> CREATE EXTENSION IF NOT EXISTS postgres_fdw;
```

Verwaltung der Planerfassung für Aurora Replicas

Aktivierung der Planerfassung für Aurora Replicas

Sie müssen über `rds_superuser`-Berechtigungen verfügen, um die Planerfassung in Aurora Replicas zu erstellen oder zu entfernen. Weitere Informationen zu Benutzerrollen und Berechtigungen finden Sie unter PostgreSQL-Rollen und -Berechtigungen verstehen.

Um Pläne zu erfassen, rufen Sie die Funktion apg_plan_mgmt.create_replica_plan_capture in der Writer-DB-Instance auf, wie im Folgenden dargestellt:

```
postgres=> CALL apg_plan_mgmt.create_replica_plan_capture('endpoint', 'password');
```

- endpoint – Der Schreiber-Endpunkt von Aurora Global Database oder cluster_endpoint bietet Failover-Unterstützung für die Planerfassung in Aurora-Replikaten.

  Weitere Informationen zum Schreiber-Endpunkt von Aurora Global Database finden Sie unter Anzeigen der Endpunkte einer globalen Amazon-Aurora-Datenbank.

  Weitere Informationen zu Cluster-Endpunkten finden Sie unter Cluster-Endpunkte für Amazon Aurora.
- password – Wir empfehlen, bei der Erstellung des Passworts die folgenden Richtlinien zu beachten, um die Sicherheit zu erhöhen:
  - Es muss mindestens 8 Zeichen enthalten.
  - Es muss mindestens einen Großbuchstaben, einen Kleinbuchstaben und eine Ziffer enthalten.
  - Es muss mindestens ein Sonderzeichen (?, !, #, <, >, * usw.) enthalten.

> **Note**
>
> Wenn Sie den Endpunkt, das Passwort oder die Portnummer ändern, müssen Sie `apg_plan_mgmt.create_replica_plan_capture()` erneut mit dem Endpunkt und dem Passwort ausführen, um die Planerfassung erneut zu initialisieren. Andernfalls schlägt die Erfassung von Plänen von Aurora Replicas fehl.

Deaktivieren der Planerfassung für Aurora Replicas

Sie können den `capture_plan_baselines`-Parameter in Aurora Replica deaktivieren, indem Sie seinen Wert in der Gruppe Parameter auf `off` setzen.

Entfernen der Planerfassung für Aurora Replicas

Sie können die Planerfassung in Aurora Replicas vollständig entfernen, stellen vorher jedoch Folgendes sicher: Rufen Sie `apg_plan_mgmt.remove_replica_plan_capture` wie folgt auf, um Plan Capture zu entfernen:

```
postgres=> CALL apg_plan_mgmt.remove_replica_plan_capture();
```

Sie müssen apg_plan_mgmt.create_replica_plan_capture () erneut aufrufen, um die Planerfassung in Aurora-Replikaten mit dem Endpunkt und dem Passwort zu aktivieren.

Fehlerbehebung

Im Folgenden finden Sie Tipps zur Fehlerbehebung und Workarounds für den Fall, dass der Plan nicht wie erwartet in Aurora Replicas erfasst wird.

- Parametereinstellungen – Prüfen Sie, ob der `capture_plan_baselines`-Parameter auf den richtigen Wert gesetzt ist, um die Planerfassung zu aktivieren.
- Die **`postgres_fdw`**-Erweiterung ist installiert – Verwenden Sie die folgende Abfrage, um zu überprüfen, ob `postgres_fdw` installiert ist.

  ```
  postgres=> SELECT * FROM pg_extension WHERE extname = 'postgres_fdw'
  ```

- create_replica_plan_capture() wird aufgerufen – Verwenden Sie den folgenden Befehl, um zu überprüfen, ob die Benutzerzuordnung vorhanden ist. Rufen Sie andernfalls `create_replica_plan_capture()` auf, um das Feature zu initialisieren.

```
postgres=> SELECT * FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- Endpunkt und Portnummer – Überprüfen Sie, ob der Endpunkt und die Portnummer korrekt sind. Wenn diese Werte inkorrekt sind, wird keine Fehlermeldung angezeigt.

  Verwenden Sie den folgenden Befehl, um zu überprüfen, ob der Endpunkt in create () verwendet wurde und in welcher Datenbank er sich befindet:

```
postgres=> SELECT srvoptions FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- reload() – Sie müssen apg_plan_mgmt.reload() aufrufen, nachdem Sie apg_plan_mgmt.delete_plan() in Aurora Replicas aufgerufen haben, damit die Löschfunktion wirksam wird. Dadurch wird sichergestellt, dass die Änderung erfolgreich implementiert wurde.
- Passwort – Sie müssen das Passwort in create_replica_plan_capture() gemäß den genannten Richtlinien eingeben. Andernfalls wird eine Fehlermeldung angezeigt. Weitere Informationen finden Sie unter Verwaltung der Planerfassung für Aurora Replicas. Verwenden Sie ein anderes Passwort, das den Anforderungen entspricht.
- Regionsübergreifende Verbindung – Die Planerfassung in Aurora Replicas wird auch in der globalen Aurora-Datenbank unterstützt, wo sich Writer-Instance und Aurora Replicas in verschiedenen Regionen befinden können. Stellen Sie sicher, dass Sie den Schreiber-Endpunkt von Aurora Global Database verwenden, um die Konnektivität nach Failover- oder Umstellungsereignissen aufrechtzuerhalten. Weitere Informationen zu den Endpunkten von Aurora Global Database finden Sie unter Anzeigen der Endpunkte einer Amazon Aurora Global Database. Die Writer-Instance und das regionsübergreifende Replikat müssen in der Lage sein, mithilfe von VPC-Peering zu kommunizieren. Weitere Informationen finden Sie unter VPC-Peering. Wenn ein regionsübergreifendes Failover auftritt, müssen Sie den Endpunkt auf einen neuen primären DB-Cluster-Endpunkt umkonfigurieren.

> **Note**
>
> Wenn Sie einen Cluster-Endpunkt anstelle eines Schreiber-Endpunkts von Aurora Global Database verwenden, müssen Sie den Cluster-Endpunkt aktualisieren, nachdem Sie entweder eine globale Failover- oder Umstellungsoperation durchgeführt haben.

Unterstützung der Tabellenpartition

Aurora PostgreSQL Query Plan Management (QPM) unterstützt die deklarative Tabellenpartitionierung in den folgenden Versionen:

- 15.3 und höhere 15-Versionen
- 14.8 und höhere 14-Versionen
- 13.11 und höhere 13-Versionen

Weitere Informationen finden Sie unter Table Partitioning.

Themen

- Einrichten der Tabellenpartition
- Erfassen von Plänen für die Tabellenpartition
- Erzwingen eines Tabellenpartitionsplans
- Namenskonvention

Einrichten der Tabellenpartition

Gehen Sie wie folgt vor, um die Tabellenpartition in Aurora PostgreSQL QPM einzurichten:

1. Setzen Sie `apg_plan_mgmt.plan_hash_version` in der DB-Cluster-Parametergruppe auf 3 oder mehr.
2. Navigieren Sie zu einer Datenbank, die Query Plan Management verwendet und Einträge in der Ansicht `apg_plan_mgmt.dba_plans` aufweist.
3. Rufen Sie `apg_plan_mgmt.validate_plans('update_plan_hash')` auf, um den `plan_hash`-Wert in der Plantabelle zu aktualisieren.
4. Wiederholen Sie die Schritte 2 bis 3 für alle Datenbanken, für die Query Plan Management aktiviert ist und die Einträge in `apg_plan_mgmt.dba_plans` aufweisen.

Weitere Informationen zu diesen Parametern finden Sie unter Parameterreferenz für Aurora-PostgreSQL-Abfrageplanverwaltung.

Erfassen von Plänen für die Tabellenpartition

In QPM werden verschiedene Pläne durch ihren `plan_hash`-Wert unterschieden. Um die Änderungen am `plan_hash` zu verstehen, müssen Sie zunächst ähnliche Pläne verstehen.

Die Kombination von Zugriffsmethoden, Indexnamen ohne Ziffern und Partitionsnamen ohne Ziffern, die auf der Ebene des Append-Knotens gespeichert sind, muss konstant sein, damit die Pläne als identisch gelten. Die spezifischen Partitionen, auf die in den Plänen zugegriffen wird, sind nicht wichtig. Im folgenden Beispiel wird eine Tabelle `tbl_a` mit 4 Partitionen erstellt.

```
postgres=>create table tbl_a(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table tbl_a1 partition of tbl_a for values from (0) to (1000);
CREATE TABLE
postgres=>create table tbl_a2 partition of tbl_a for values from (1001) to (2000);
CREATE TABLE
postgres=>create table tbl_a3 partition of tbl_a for values from (2001) to (3000);
CREATE TABLE
postgres=>create table tbl_a4 partition of tbl_a for values from (3001) to (4000);
CREATE TABLE
postgres=>create index t_i on tbl_a using btree (i);
CREATE INDEX
postgres=>create index t_j on tbl_a using btree (j);
CREATE INDEX
postgres=>create index t_k on tbl_a using btree (k);
CREATE INDEX
```

Die folgenden Pläne gelten als identisch, da unabhängig von der Anzahl der Partitionen, nach denen die Abfrage sucht, eine einzige Scanmethode zum Scannen von `tbl_a` verwendet wird.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 999 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Seq Scan on tbl_a1 tbl_a
    Filter: ((i >= 990) AND (i <= 999) AND (j < 9910) AND (k > 50))
SQL Hash: 1553185667, Plan Hash: -694232056
(3 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;
```

```
                        QUERY PLAN
------------------------------------------------------------------
Append
    ->  Seq Scan on tbl_a1 tbl_a_1
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    ->  Seq Scan on tbl_a2 tbl_a_2
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    SQL Hash: 1553185667, Plan Hash: -694232056
    (6 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(8 rows)
```

Die folgenden 3 Pläne gelten ebenfalls als identisch, da auf der übergeordneten Ebene die Zugriffsmethoden, Indexnamen ohne Ziffern und Partitionsnamen ohne Ziffern `SeqScan tbl_a` und `IndexScan (i_idx) tbl_a` lauten.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                               QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_i_idx on tbl_a2 tbl_a_2
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
```

```
(7 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(11 rows)
```

Unabhängig von der unterschiedlichen Reihenfolge und Anzahl der Vorkommen in untergeordneten Partitionen sind die Zugriffsmethoden, Indexnamen ohne Ziffern und Partitionsnamen ohne Ziffern auf der übergeordneten Ebene für jeden der oben aufgeführten Pläne konstant.

Die Pläne würden jedoch nicht als identisch gelten, wenn eine der folgenden Bedingungen erfüllt ist:

- Es werden zusätzliche Zugriffsmethoden im Plan verwendet.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 1134525070
(11 rows)
```

- Eine der im Plan enthaltenen Zugriffsmethoden wird nicht mehr verwendet.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(6 rows)
```

- Der einer Indexmethode zugeordnete Index wird geändert.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                               QUERY PLAN
-------------------------------------------------------------------------
 Append
```

```
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_j_idx on tbl_a2 tbl_a_2
         Index Cond: (j < 9910)
         Filter: ((i >= 990) AND (i <= 1100) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993343726
(7 rows)
```

Erzwingen eines Tabellenpartitionsplans

Genehmigte Pläne für partitionierte Tabellen werden durch Positionskorrespondenz erzwungen. Die Pläne sind nicht spezifisch für die Partitionen und können für andere Partitionen als die in der ursprünglichen Abfrage referenzierten Pläne erzwungen werden. Pläne können auch für Abfragen erzwungen werden, die auf eine andere Anzahl von Partitionen als die ursprünglich genehmigte Gliederung zugreifen.

Beispiel: Die genehmigte Gliederung gilt für den folgenden Plan:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

Dann kann dieser Plan auch für SQL-Abfragen erzwungen werden, die 2, 4 oder mehr Partitionen referenzieren. Die möglichen Pläne, die sich aus diesen Szenarien für den Zugriff auf 2 und 4 Partitionen ergeben könnten, sind folgende:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;
```

```
                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(8 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a4 tbl_a_4
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(12 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
```

```
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(14 rows)
```

Betrachten Sie einen anderen genehmigten Plan mit unterschiedlichen Zugriffsmethoden für jede Partition:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
--------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 2032136998
(12 rows)
```

In diesem Fall würde jeder Plan, der aus zwei Partitionen liest, nicht erzwungen werden. Solange nicht alle Kombinationen (Zugriffsmethode, Indexname) aus dem genehmigten Plan verwendet werden können, kann der Plan nicht erzwungen werden. Die folgenden Pläne haben beispielsweise unterschiedliche Plan-Hashes und der genehmigte Plan kann in diesen Fällen nicht erzwungen werden:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;
```

```
                              QUERY PLAN
------------------------------------------------------------------------
 Append
   ->  Bitmap Heap Scan on tbl_a1 tbl_a_1
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a1_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
   ->  Bitmap Heap Scan on tbl_a2 tbl_a_2
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a2_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
  Note: This is not an Approved plan.  No usable Approved plan was found.
  SQL Hash: 1553185667, Plan Hash: -568647260
(13 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1900) AND (j < 9910) AND (k > 50))
 Note: This is not an Approved plan.  No usable Approved plan was found.
 SQL Hash: 1553185667, Plan Hash: -496793743
(8 rows)
```

Namenskonvention

Damit QPM einen Plan mit deklarativ partitionierten Tabellen durchsetzen kann, müssen Sie bestimmte Benennungsregeln für übergeordnete Tabellen, Tabellenpartitionen und Indizes befolgen:

- Namen von übergeordneten Tabellen – Diese Namen müssen sich durch alphanumerische oder Sonderzeichen und nicht nur durch Ziffern unterscheiden. Beispielsweise sind tA, tB und tC zulässige Namen für separate übergeordnete Tabellen, t1, t2 und t3 hingegen nicht.

- Namen einzelner Partitionstabellen – Partitionen derselben übergeordneten Tabelle dürfen sich nur durch Ziffern voneinander unterscheiden. Zulässige Partitionsnamen von tA könnten beispielsweise tA1, tA2 oder t1A, t2A oder auch mehrere Ziffern sein.

  Alle anderen Unterschiede (Buchstaben, Sonderzeichen) garantieren das Erzwingen des Plans nicht.

- Indexnamen – Stellen Sie in der Hierarchie der Partitionstabellen sicher, dass alle Indizes eindeutige Namen haben. Das bedeutet, dass die nicht numerischen Teile der Namen unterschiedlich sein müssen. Wenn Sie beispielsweise eine partitionierte Tabelle namens `tA` mit einem Index namens `tA_col1_idx1` haben, können Sie keinen anderen Index namens `tA_col1_idx2` haben. Sie können jedoch einen Index namens `tA_a_col1_idx2` haben, da der nicht numerische Teil des Namens eindeutig ist. Diese Regel gilt für Indizes, die sowohl für die übergeordnete Tabelle als auch für einzelne Partitionstabellen erstellt wurden.

Wenn die oben genannten Namenskonventionen nicht eingehalten werden, kann dies dazu führen, dass die genehmigten Pläne nicht erzwungen werden. Das folgende Beispiel veranschaulicht ein fehlgeschlagenes Erzwingen:

```
postgres=>create table t1(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table t1a partition of t1 for values from (0) to (1000);
CREATE TABLE
postgres=>create table t1b partition of t1 for values from (1001) to (2000);
CREATE TABLE
postgres=>SET apg_plan_mgmt.capture_plan_baselines TO 'manual';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 0;

                           QUERY PLAN
-----------------------------------------------------------------------------
 Aggregate
   ->  Append
         ->  Seq Scan on t1a t1_1
               Filter: (i > 0)
         ->  Seq Scan on t1b t1_2
               Filter: (i > 0)
 SQL Hash: -1720232281, Plan Hash: -1010664377
(7 rows)
```

```
postgres=>SET apg_plan_mgmt.use_plan_baselines TO 'on';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 1000;

                            QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   ->  Seq Scan on t1b t1
         Filter: (i > 1000)
 Note: This is not an Approved plan. No usable Approved plan was found.
 SQL Hash: -1720232281, Plan Hash: 335531806
(5 rows)
```

Auch wenn die beiden Pläne identisch erscheinen mögen, unterscheiden sich ihre `Plan Hash`-Werte aufgrund der Namen der untergeordneten Tabellen. Die Tabellennamen unterscheiden sich nach Alphazeichen und nicht nur nach Ziffern, was dazu führt, dass die Durchsetzung fehlschlägt.

Arbeiten mit Erweiterungen und Fremddaten-Wrappern

Sie können verschiedene PostgreSQL-Erweiterungen installieren und verwenden, um Ihrem DB-Cluster der Aurora-PostgreSQL-kompatiblen Edition einige Funktionen hinzuzufügen. Wenn Ihr Anwendungsfall beispielsweise eine intensive Dateneingabe in sehr großen Tabellen verlangt, können Sie die `pg_partman`-Erweiterung installieren, um Ihre Daten zu partitionieren und damit den Workload zu verteilen.

> **Note**
>
> Ab Aurora PostgreSQL 14.5 unterstützt Aurora PostgreSQL Trusted Language Extensions für PostgreSQL. Diese Funktion ist als Erweiterung `pg_tle` implementiert, die Sie Ihrem Aurora PostgreSQL hinzufügen können. Mithilfe dieser Erweiterung können Entwickler ihre eigenen PostgreSQL-Erweiterungen in einer sicheren Umgebung erstellen, was die Setup- und Konfigurationsanforderungen sowie einen Großteil der Vorabtests für neue Erweiterungen vereinfacht. Weitere Informationen finden Sie unter Arbeiten mit Trusted Language Extensions für PostgreSQL.

In einigen Fällen bietet es sich an, anstatt eine Erweiterung zu installieren, ein bestimmtes Modul zur Liste der `shared_preload_libraries` in der benutzerdefinierten DB-Cluster-Parametergruppe Ihres DB-Clusters von Aurora PostgreSQL hinzuzufügen. In der Regel lädt die standardmäßige DB-Cluster-Parametergruppe nur die `pg_stat_statements`. Es stehen jedoch weitere Module zur Verfügung, die der Liste hinzugefügt werden können. Sie können beispielsweise Planungsfunktionen hinzufügen, indem Sie das `pg_cron`-Modul hinzufügen, wie unter Planen der Wartung mit der PostgreSQL-Erweiterung pg_cron beschrieben. Als weiteres Beispiel können Sie Abfrageausführungspläne protokollieren, indem Sie das `auto_explain`-Modul laden. Weitere Informationen finden Sie unter Protokollieren von Abfrageausführungsplänen im AWS-Wissenscenter.

Eine Erweiterung, die Zugriff auf externe Daten ermöglicht, wird als Fremddaten-Wrapper (FDW) bezeichnet. Zum Beispiel ermöglicht die Erweiterung `oracle_fdw` Ihrem Aurora-PostgreSQL-DB-Cluster die Zusammenarbeit mit Oracle-Datenbanken.

Sie können auch genau angeben, welche Erweiterungen auf Ihrer Aurora PostgreSQL DB-Instance installiert werden können, indem Sie sie im Parameter `rds.allowed_extensions` aufführen. Weitere Informationen finden Sie unter Einschränkung der Installation von PostgreSQL-Erweiterungen.

Im Folgenden finden Sie Informationen zum Einrichten und Verwenden einiger Erweiterungen, Module und FDWs, die für Aurora PostgreSQL verfügbar sind. Der Einfachheit halber werden diese alle als „Erweiterungen“ bezeichnet. Eine Auflistung der Erweiterungen, die Sie mit den aktuell verfügbaren Aurora-PostgreSQL-Versionen verwenden können, finden Sie unter Versionen der Erweiterungen für Amazon Aurora PostgreSQL in Versionshinweise für Aurora PostgreSQL.

- Verwalten großer Objekte mit dem lo-Modul
- Verwalten von Geodaten mit der PostGIS-Erweiterung
- Verwalten von PostgreSQL-Partitionen mit der Erweiterung pg_partman
- Planen der Wartung mit der PostgreSQL-Erweiterung pg_cron
- Verwenden von pgAudit zur Protokollierung der Datenbankaktivität
- Verwenden von pglogical, um Daten zwischen Instances zu synchronisieren
- Arbeiten mit Oracle-Datenbanken unter Verwendung der Erweiterung oracle_fdw
- Arbeiten mit SQL-Server-Datenbanken unter Verwendung der Erweiterung tds_fdw

Verwenden der Unterstützung für delegierte Erweiterungen von Amazon Aurora für PostgreSQL

Mithilfe der Unterstützung für delegierte Erweiterungen von Amazon Aurora für PostgreSQL können Sie die Erweiterungsverwaltung an einen Benutzer delegieren, der kein `rds_superuser` sein muss. Mit dieser Unterstützung für delegierte Erweiterungen wird eine neue Rolle namens `rds_extension` erstellt und Sie müssen diese einem Benutzer zuweisen, um andere Erweiterungen zu verwalten. Mit dieser Rolle können Erweiterungen erstellt, aktualisiert und gelöscht werden.

Sie können die Erweiterungen angeben, die auf Ihrer DB-Instance von Aurora PostgreSQL installiert werden können, indem Sie sie im Parameter `rds.allowed_extensions` aufführen. Weitere Informationen finden Sie unter Verwenden von PostgreSQL-Erweiterungen mit Amazon RDS für PostgreSQL.

Mit dem Parameter `rds.allowed_delegated_extensions` können Sie die Liste der Erweiterungen einschränken, die vom Benutzer mit der `rds_extension`-Rolle verwaltet werden kann.

Die Unterstützung für delegierte Erweiterungen ist in den folgenden Versionen verfügbar:

- Alle höheren Versionen

- 15.5 und höhere 15-Versionen
- 14.10 und höhere 14 Versionen
- 13.13 und höhere 13 Versionen
- 12.17 und höhere 12 Versionen

Themen

- Aktivieren der Unterstützung für delegierte Erweiterungen für einen Benutzer
- Konfiguration, die in der Unterstützung für delegierte Erweiterungen von Aurora für PostgreSQL verwendet wird
- Deaktivieren der Unterstützung für die delegierte Erweiterung
- Vorteile der Verwendung der Unterstützung für delegierte Erweiterungen von Amazon Aurora
- Einschränkung der Unterstützung für delegierte Erweiterungen von Amazon Aurora für PostgreSQL
- Für bestimmte Erweiterungen erforderliche Berechtigungen
- Sicherheitsüberlegungen
- Drop extension cascade ist deaktiviert
- Beispielerweiterungen, die mithilfe der Unterstützung für delegierte Erweiterungen hinzugefügt werden können

Aktivieren der Unterstützung für delegierte Erweiterungen für einen Benutzer

Sie müssen die folgenden Schritte ausführen, um die Unterstützung für delegierte Erweiterungen für einen Benutzer zu aktivieren:

1. Einem Benutzer die **`rds_extension`**-Rolle erteilen – Stellen Sie als `rds_superuser` eine Verbindung zur Datenbank her und führen Sie den folgenden Befehl aus:

   ```
   Postgres => grant rds_extension to user_name;
   ```

2. Die Liste der Erweiterungen festlegen, die von delegierten Benutzern verwaltet werden können – Mit `rds.allowed_delegated_extensions` können Sie unter Verwendung von `rds.allowed_extensions` im DB-Cluster-Parameter eine Teilmenge der verfügbaren Erweiterungen angeben. Sie können dies auf einer der folgenden Ebenen durchführen:

   - Im Cluster oder in der Instanzparametergruppe, über die API AWS-Managementkonsole oder. Weitere Informationen finden Sie unter Parametergruppen für Amazon Aurora.

- Verwenden Sie den folgenden Befehl auf Datenbankebene:

```
alter database database_name set rds.allowed_delegated_extensions =
 'extension_name_1,
                   extension_name_2,...extension_name_n';
```

- Verwenden Sie den folgenden Befehl auf Benutzerebene:

```
alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
                   extension_name_2,...extension_name_n';
```

> **Note**
>
> Sie müssen die Datenbank nicht neu starten, nachdem Sie den dynamischen Parameter `rds.allowed_delegated_extensions` geändert haben.

3. Dem delegierten Benutzer Zugriff auf Objekte gewähren, die während dem Prozess der Erweiterungserstellung erstellt wurden – Bestimmte Erweiterungen erstellen Objekte, für die zusätzliche Berechtigungen erteilt werden müssen, bevor der Benutzer mit der `rds_extension`-Rolle auf sie zugreifen kann. Der `rds_superuser` muss dem delegierten Benutzer Zugriff auf diese Objekte gewähren. Eine der Optionen besteht darin, einen Ereignisauslöser zu verwenden, um dem delegierten Benutzer automatisch die Berechtigung zu erteilen.

   Beispiel für einen Event-Trigger

   Wenn Sie einem delegierten Benutzer mit `rds_extension` die Verwendung von Erweiterungen ermöglichen möchten, für die Berechtigungen für seine bei der Erstellung der Erweiterung erstellten Objekte erforderlich sind, können Sie das folgende Beispiel für einen Ereignisauslöser anpassen und nur die Erweiterungen hinzufügen, für die die delegierten Benutzer Zugriff auf die gesamte Funktionalität haben sollen. Dieser Ereignisauslöser kann auf template1 (der Standardvorlage) erstellt werden, sodass alle mit template1 erstellten Datenbanken über diesen Ereignisauslöser verfügen. Wenn ein delegierter Benutzer die Erweiterung installiert, gewährt dieser Trigger automatisch die Eigentümerschaft an den von der Erweiterung erstellten Objekten.

```
CREATE OR REPLACE FUNCTION create_ext()
```

```
  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;

BEGIN

  IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
    RAISE NOTICE 'SECURITY INVOKER';
    RAISE NOTICE 'user: %', current_user;
    FOR r IN SELECT * FROM pg_catalog.pg_event_trigger_ddl_commands()
    LOOP
        CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
 'extension';

        schemaname = (
            SELECT n.nspname
            FROM pg_catalog.pg_extension AS e
            INNER JOIN pg_catalog.pg_namespace AS n
            ON e.extnamespace = n.oid
            WHERE e.oid = r.objid
        );

        databaseowner = (
            SELECT pg_catalog.pg_get_userbyid(d.datdba)
            FROM pg_catalog.pg_database d
            WHERE d.datname = current_database()
        );
        RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %,
 schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user,
 schemaname, databaseowner;
        IF r.object_identity = 'address_standardizer_data_us' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_gaz TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_lex TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_rules TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'dict_int' THEN
```

```
            EXECUTE pg_catalog.format('ALTER TEXT SEARCH DICTIONARY %I.intdict
 OWNER TO %I;', schemaname, databaseowner);
        ELSIF r.object_identity = 'pg_partman' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE
 ON TABLE %I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname,
 databaseowner);
        ELSIF r.object_identity = 'postgis_topology' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL
 TABLES IN SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT USAGE, SELECT ON ALL SEQUENCES IN
 SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA
 topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT USAGE ON SCHEMA topology TO %I WITH
 GRANT OPTION;', databaseowner);
        END IF;
    END LOOP;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE
 create_ext();
```

Konfiguration, die in der Unterstützung für delegierte Erweiterungen von Aurora für PostgreSQL verwendet wird

Konfig tionsna	Description	Standardwert	Hinweise	Wer Berechtig ungen ändern oder erteilen kann
rds.a ed_de ted_e sions	Dieser Parameter schränkt die Erweiterungen ein, die eine rds_exten	leere Zeichenfolge	• Standardm äßig ist dieser Parameter eine leere	rds_superuser

Konfig tionsna	Description	Standardwert	Hinweise	Wer Berechtig ungen ändern oder erteilen kann
	sion-Rolle in einer Datenbank verwalten kann. Es muss sich um eine Teilmenge von rds.allowed_extens ions handeln.		Zeichenfolge, was bedeutet, dass keine Erweiteru ngen an Benutzer mit `rds_exten sion` delegiert wurden. • Jede unterstüt zte Erweiterung kann hinzugefügt werden, wenn der Benutzer dazu berechtigt ist. Setzen Sie dazu den Parameter `rds.allow ed_delega ted_exten sions` auf eine Zeichenfolge von kommagetr ennten Erweiteru ngsnamen. Indem Sie diesem Parameter eine Liste von Erweiterungen hinzufügen, identifizieren Sie explizit die Erweiteru ngen, die der Benutzer mit	

Konfigurationsna	Description	Standardwert	Hinweise	Wer Berechtigungen ändern oder erteilen kann
			der `rds_extension` -Rolle installieren kann. • Wenn dieser Wert auf * gesetzt ist, bedeutet dies, dass alle unter `rds_allowed_extensions` aufgeführten Erweiterungen an Benutzer mit der `rds_extension` -Rolle delegiert werden. Weitere Informationen zum Einrichten dieses Parameters finden Sie unter Aktivieren der Unterstützung für delegierte Erweiterungen für einen Benutzer.	

Konfigurationsna	Description	Standardwert	Hinweise	Wer Berechtigungen ändern oder erteilen kann
`rds.allowed_extensions`	Mit diesem Parameter kann der Kunde die Erweiterungen einschränken, die in der DB-Instance von Aurora PostgreSQL installiert werden können. Weitere Informationen finden Sie unter Einschränken der Installation von PostgreSQL-Erweiterungen.	"*"	Standardmäßig ist dieser Parameter auf „*“ gesetzt, was bedeutet, dass alle von RDS für PostgreSQL und Aurora PostgreSQL unterstützten Erweiterungen von Benutzern mit den erforderlichen Berechtigungen erstellt werden dürfen. Leer bedeutet, dass keine Erweiterungen in der DB-Instance von Aurora PostgreSQL installiert werden können.	Administrator

Konfig tionsna	Description	Standardwert	Hinweise	Wer Berechtig ungen ändern oder erteilen kann
`rds-deleg ated_ nsior ow_dr ascac`	Dieser Parameter steuert, ob Benutzer mit `rds_exten sion` die Erweiterung mithilfe der Option cascade löschen können.	aus	`rds-deleg ated_exte nsion_all ow_drop_c ascade` ist standardmäßig auf `off` festgeleg t. Dies bedeutet, dass Benutzer mit `rds_extension` eine Erweiterung mithilfe der Option cascade nicht löschen dürfen. Zur Erteilung dieser Fähigkeit sollte der Parameter `rds.deleg ated_exte nsion_all ow_drop_c ascade` auf on gesetzt werden.	rds_superuser

Deaktivieren der Unterstützung für die delegierte Erweiterung

Teilweises Deaktivieren

Die delegierten Benutzer können keine neuen Erweiterungen erstellen, aber trotzdem bestehende Erweiterungen aktualisieren.

- Setzen Sie `rds.allowed_delegated_extensions` in der DB-Cluster-Parametergruppe auf den Standardwert zurück.
- Verwenden Sie den folgenden Befehl auf Datenbankebene:

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- Verwenden Sie den folgenden Befehl auf Benutzerebene:

```
alter user user_name reset rds.allowed_delegated_extensions;
```

Vollständiges Deaktivieren

Wenn Sie einem Benutzer die `rds_extension`-Rolle entziehen, erhält der Benutzer wieder die Standardberechtigungen. Der Benutzer kann keine Erweiterungen mehr erstellen, aktualisieren oder löschen.

```
postgres => revoke rds_extension from user_name;
```

Vorteile der Verwendung der Unterstützung für delegierte Erweiterungen von Amazon Aurora

Durch die Verwendung der Unterstützung für delegierte Erweiterungen von Amazon Aurora für PostgreSQL delegieren Sie die Erweiterungsverwaltung sicher an Benutzer, die nicht über die `rds_superuser`-Rolle verfügen. Diese Funktion bietet die folgenden Vorteile:

- Sie können die Erweiterungsverwaltung ganz einfach an Benutzer Ihrer Wahl delegieren.
- Dafür ist keine `rds_superuser`-Rolle erforderlich.
- Bietet die Möglichkeit, verschiedene Gruppen von Erweiterungen für unterschiedliche Datenbanken im selben DB-Cluster zu unterstützen

Einschränkung der Unterstützung für delegierte Erweiterungen von Amazon Aurora für PostgreSQL

- Objekte, die während der Erweiterungserstellung erstellt wurden, benötigen möglicherweise zusätzliche Berechtigungen, damit die Erweiterung ordnungsgemäß funktioniert.

Für bestimmte Erweiterungen erforderliche Berechtigungen

Um die folgenden Erweiterungen zu erstellen, zu verwenden oder zu aktualisieren, sollte der delegierte Benutzer über die erforderlichen Berechtigungen für die folgenden Funktionen, Tabellen und Schemas verfügen.

Erwe ngen für die eine Eiger rscha oder Bere unge erfor ich sind	Funktion	Tabellen	Schema	Wörterbuch für die Textsuche	Comment
addre tanda er_da s		us_gaz, us_lex, us_lex, l.us_rules			
amch	bt_index_check, bt_index_ parent_check				
dict_i				intdict	
pg_p n		custom_ti me_partitions, part_config, part_config_sub			
pg_st tatem					

Erwe ngen für die eine Eiger rscha oder Bere unge erfor ich sind	Funktion	Tabellen	Schema	Wörterbuch für die Textsuche	Comment
Post(	st_tileenvelope	spatial_ref_sys			
postg aster					
postg opolo		topology, layer	topology		Der delegierte Benutzer muss der Datenbank besitzer sein.
log_f	create_fo reign_tab le_for_log_file				
rds_t	role_pass word_encr yption_type				
postg iger_ oder		geocode_s ettings_default, geocode_s ettings	tiger		

Erwe ngen für die eine Eiger rscha oder Bere unge erfor ich sind	Funktion	Tabellen	Schema	Wörterbuch für die Textsuche	Comment
pg_fr acem	pg_freespace				
pg_vi lity	pg_visibility				

Sicherheitsüberlegungen

Denken Sie daran, dass ein Benutzer mit der `rds_extension`-Rolle Erweiterungen für alle Datenbanken verwalten kann, für die er die Connect-Berechtigung hat. Wenn beabsichtigt ist, dass ein delegierter Benutzer die Erweiterung für eine einzelne Datenbank verwaltet, empfiehlt es sich, alle öffentlichen Berechtigungen für jede Datenbank zu entziehen und dann dem delegierten Benutzer explizit die Connect-Berechtigung für diese spezifische Datenbank zu gewähren.

Es gibt mehrere Erweiterungen, mit denen ein Benutzer auf Informationen aus mehreren Datenbanken zugreifen kann. Stellen Sie sicher, dass die Benutzer, denen Sie `rds_extension` gewähren, über datenbankübergreifende Funktionen verfügen, bevor Sie diese Erweiterungen zu `rds.allowed_delegated_extensions` hinzufügen. `postgres_fdw` und `dblink` stellen beispielsweise Funktionen für datenbankübergreifende Abfragen auf derselben Instance oder auf Remote-Instances bereit. `log_fdw` liest die Protokolldateien der Postgres-Engine, die für alle Datenbanken in der Instance gelten und möglicherweise langsame Abfragen oder Fehlermeldungen von mehreren Datenbanken enthalten. `pg_cron` ermöglicht die Ausführung von geplanten

Hintergrundaufträgen auf der DB-Instance und kann Aufträge so konfigurieren, dass sie in einer anderen Datenbank ausgeführt werden.

Drop extension cascade ist deaktiviert

Die Möglichkeit, die Erweiterung mit der Option cascade durch einen Benutzer mit der `rds_extension`-Rolle zu löschen, wird durch den Parameter `rds.delegated_extension_allow_drop_cascade` gesteuert. `rds-delegated_extension_allow_drop_cascade` ist standardmäßig auf `off` festgelegt. Das bedeutet, dass Benutzer mit der Rolle `rds_extension` keine Erweiterung mithilfe der Option cascade löschen dürfen, wie in der untenstehenden Abfrage gezeigt.

```
DROP EXTENSION CASCADE;
```

Da dadurch automatisch Objekte gelöscht werden, die von der Erweiterung abhängen – und in der Folge auch alle Objekte, die wiederum von diesen Objekten abhängig sind. Der Versuch, die Option cascade zu verwenden, führt zu einem Fehler.

Zur Erteilung dieser Fähigkeit sollte der Parameter `rds.delegated_extension_allow_drop_cascade` auf on gesetzt werden.

Das Ändern des dynamischen Parameters `rds.delegated_extension_allow_drop_cascade` erfordert keinen Neustart der Datenbank. Sie können dies auf einer der folgenden Ebenen tun:

- Im Cluster oder in der Instanzparametergruppe, über die API AWS-Managementkonsole oder.
- Verwenden Sie den folgenden Befehl auf Datenbankebene:

  ```
  alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

- Verwenden Sie den folgenden Befehl auf Benutzerebene:

  ```
  alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

Beispielerweiterungen, die mithilfe der Unterstützung für delegierte Erweiterungen hinzugefügt werden können

- `rds_tools`

  ```
  extension_test_db=> create extension rds_tools;
  ```

```
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- amcheck

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- pg_freespacemap

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- pg_visibility

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- postgres_fdw

```
extension_test_db=> create extension postgres_fdw;
CREATE EXTENSION
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
```

```
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

Verwalten großer Objekte mit dem lo-Modul

Das lo-Modul (Erweiterung) richtet sich an Datenbankbenutzer und Entwickler, die mit PostgreSQL-Datenbanken über JDBC- oder ODBC-Treiber arbeiten. Sowohl JDBC als auch ODBC erwarten, dass die Datenbank das Löschen großer Objekte bewältigt, wenn sich Verweise auf sie ändern. PostgreSQL funktioniert jedoch nicht so. PostgreSQL geht nicht davon aus, dass ein Objekt gelöscht werden sollte, wenn sich sein Verweis ändert. Demzufolge verbleiben Objekte auf der Festplatte und sind nicht referenziert. Die Erweiterung lo enthält eine Funktion, mit der bei Referenzänderungen das Löschen von Objekten bei Bedarf ausgelöst wird.

> **Tip**
>
> Wenn Sie feststellen möchten, ob Ihre Datenbank von der lo-Erweiterung profitieren kann, suchen Sie mit dem `vacuumlo`-Dienstprogramm nach verwaisten großen Objekten. Um die Anzahl von verwaisten großen Objekten zu erhalten, ohne Maßnahmen zu ergreifen, führen Sie das Dienstprogramm mit der `-n` Option (no-op) aus. Weitere Informationen erhalten Sie unter vacuumlo utility.

Das lo-Modul ist für Aurora PostgreSQL 13.7, 12.11, 11.16, 10.21 und höhere Nebenversionen verfügbar.

Zum Installieren des Moduls (Erweiterung) benötigen Sie `rds_superuser`-Berechtigungen. Durch die Installation der lo-Erweiterung wird Ihrer Datenbank Folgendes hinzugefügt:

- `lo` – Dies ist ein Datentyp für große Objekte (lo), den Sie für binäre große Objekte (BLOBs) und andere große Objekte verwenden können. Der `lo`-Datentyp ist eine Domäne des `oid`-Datentyps. Mit anderen Worten, es ist eine Objekt-ID mit optionalen Beschränkungen. Weitere Informationen finden Sie unter Objekt-IDs in der PostgreSQL-Dokumentation. Vereinfacht gesagt, können Sie anhand des `lo`-Datentyps Ihre Datenbankspalten, die große Objektreferenzen enthalten, von anderen Objekt-IDs (OIDs) unterscheiden.
- `lo_manage` – Dies ist eine Funktion, die Sie in Triggern für Tabellenspalten verwenden können, die große Objektreferenzen enthalten. Immer wenn Sie einen Wert löschen oder ändern, der auf ein großes Objekt verweist, hebt der Trigger die Verknüpfung des Objekts (`lo_unlink`) mit

seiner Referenz auf. Verwenden Sie den Trigger für eine Spalte nur, wenn die Spalte der einzige Datenbankverweis auf das große Objekt ist.

Weitere Informationen zum Modul für große Objekte erhalten Sie unter lo in der PostgreSQL-Dokumentation.

Installieren der lo-Erweiterung

Stellen Sie vor der Installation der lo-Erweiterung sicher, dass Sie über `rds_superuser`-Berechtigungen verfügen.

So installieren Sie die lo-Erweiterung

1. Verwenden Sie `psql`, um eine Verbindung mit der primären DB-Instance Ihres Aurora-PostgreSQL-DB-Clusters herzustellen.

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   Geben Sie bei der Aufforderung Ihr Passwort ein. Der `psql`-Client verbindet und zeigt die standardmäßige administrative Verbindungsdatenbank `postgres=>` als Eingabeaufforderung an.

2. Installieren Sie die Erweiterung wie folgt.

   ```
   postgres=> CREATE EXTENSION lo;
   CREATE EXTENSION
   ```

Sie können jetzt den `lo`-Datentyp zum Definieren von Spalten in Ihren Tabellen verwenden. Beispielsweise können Sie eine Tabelle (`images`) erstellen, die Raster-Bilddaten enthält. Sie können den `lo`-Datentyp für eine Spalte `raster` verwenden, wie im folgenden Beispiel, mit dem eine Tabelle erstellt wird, gezeigt.

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

Verwenden der Triggerfunktion lo_manage zum Löschen von Objekten

Sie können die `lo_manage`-Funktion in einem Trigger in einer `lo` oder anderen großen Objektspalten verwenden, um verwaiste Objekte zu bereinigen (und zu verhindern), wenn `lo` aktualisiert oder gelöscht wird.

So richten Sie Trigger für Spalten ein, die auf große Objekte verweisen

- Führen Sie eine der folgenden Aktionen aus:
  - Erstellen Sie einen BEFORE UPDATE ODER DELETE-Trigger für jede Spalte, damit diese eindeutige Verweise auf große Objekte enthält, wobei der Spaltenname als Argument verwendet wird.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

  - Wenden Sie einen Trigger nur an, wenn die Spalte aktualisiert wird.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

Die `lo_manage`-Trigger-Funktion funktioniert nur im Kontext des Einfügens oder Löschens von Spaltendaten, je nachdem, wie Sie den Trigger definieren. Er hat keine Auswirkung, wenn Sie eine `DROP`- oder `TRUNCATE`-Operation in einer Datenbank ausführen. Das bedeutet, dass Sie vor der Verwendung der DROP-Operation Objektspalten aus allen Tabellen löschen sollten, um das Erstellen von verwaisten Objekten zu verhindern.

Angenommen, Sie möchten die Datenbank entfernen, die die Tabelle `images` enthält. Sie löschen die Spalte wie folgt.

```
postgres=> DELETE FROM images COLUMN raster
```

Vorausgesetzt, die `lo_manage`-Funktion ist in dieser Spalte definiert, um Löschvorgänge zu behandeln, können Sie die Tabelle jetzt sicher entfernen.

Entfernen verwaister großer Objekte mit **vacuumlo**

Das vacuumlo-Dienstprogramm identifiziert verwaiste große Objekte und kann diese aus Datenbanken entfernen. Dieses Dienstprogramm ist seit PostgreSQL 9.1.24 verfügbar. Wenn Ihre Datenbankbenutzer routinemäßig mit großen Objekten arbeiten, empfehlen wir Ihnen, vacuumlogelegentlich auszuführen, um verwaiste große Objekte zu bereinigen.

Bevor Sie die lo-Erweiterung installieren, können Sie mit vacuumlobeurteilen, ob Ihr Aurora-PostgreSQL-DB-Cluster davon profitieren kann. Führen Sie dazu vacuumlo mit der -n-Option (no-op) aus, um zu zeigen, was entfernt würde, wie im Folgenden gezeigt:

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -
p 5433 -U postgres docs-lab-spatial-db
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

Wie die Ausgabe zeigt, stellen verwaiste große Objekte für diese bestimmte Datenbank kein Problem dar.

Weitere Informationen über dieses Dienstprogramm finden Sie unter vacuumlo in der PostgreSQL-Dokumentation.

Grundlegendes zur Funktionsweise von **vacuumlo**

Der vacuumlo-Befehl entfernt verwaiste große Objekte (LOs) aus Ihrer PostgreSQL-Datenbank, ohne Ihre Benutzertabellen zu beeinträchtigen oder Konflikte mit ihnen zu verursachen.

Der Befehl lautet wie folgt:

1. vacuumlo beginnt mit der Erstellung einer temporären Tabelle, die alle Objekt-IDs (OIDs) der großen Objekte in Ihrer Datenbank enthält.
2. vacuumlo durchsucht dann jede Spalte in der Datenbank, die die Datentypen oid oder lo verwendet. Wenn vacuumlo in diesen Spalten eine passende OID gefunden wird, wird die OID aus der temporären Tabelle entfernt. vacuumlo überprüft nur Spalten, die die ausdrücklich mit oid oder lo benannt sind, keine Domains, die auf diesen Typen basieren.
3. Die verbleibenden Einträge in der temporären Tabelle stellen verwaiste LOs dar, die vacuumlo dann sicher entfernt.

Verbessern der Leistung von `vacuumlo`

Sie können die Leistung von `vacuumlo` potenziell verbessern, indem Sie die Batchgröße mithilfe der Option `-l` erhöhen. Dadurch kann `vacuumlo` mehr LOs gleichzeitig verarbeiten.

Wenn Ihr System über ausreichend Arbeitsspeicher verfügt und Sie die temporäre Tabelle vollständig im Arbeitsspeicher unterbringen können, kann eine Erhöhung der Einstellung `temp_buffers` auf Datenbankebene die Leistung verbessern. Dadurch kann sich die Tabelle vollständig im Arbeitsspeicher befinden, was die Gesamtleistung verbessern kann.

Die folgende Abfrage schätzt die Größe der temporären Tabelle:

```
SELECT
    pg_size_pretty(SUM(pg_column_size(oid))) estimated_lo_temp_table_size
FROM
    pg_largeobject_metadata;
```

Überlegungen zu großen Objekten

Im Folgenden finden Sie einige wichtige Überlegungen, die Sie beachten sollten, wenn Sie mit großen Objekten arbeiten:

- `Vacuumlo` ist die einzige Lösung, da es derzeit keine andere Methode gibt, verwaiste LOs zu entfernen.
- Tools wie pglogical, native logische Replikation und AWS DMS, die Replikationstechnologien verwenden, unterstützen die Replikation großer Objekte nicht.
- Vermeiden Sie beim Entwerfen Ihres Datenbankschemas nach Möglichkeit die Verwendung großer Objekte und ziehen Sie stattdessen alternative Datentypen wie `bytea` in Betracht.
- Führen Sie `vacuumlo` regelmäßig, mindestens wöchentlich, aus, um Probleme mit verwaisten LOs zu vermeiden.
- Verwenden Sie einen Trigger mit der Funktion `lo_manage` für Tabellen, in denen große Objekte gespeichert sind, um zu verhindern, dass verwaiste LOs erstellt werden.

Verwalten von Geodaten mit der PostGIS-Erweiterung

PostGIS ist eine Erweiterung von PostgreSQL zur Speicherung und Verwaltung von Geodaten. Weitere Informationen zu PostGIS finden Sie unter PostGIS.net.

Ab Version 10.5 unterstützt PostgreSQL die libprotobuf-1.3.0-Bibliothek, die von PostGIS für die Arbeit mit Vektorkacheldaten der Kartenbox verwendet wird.

Für das Einrichten der PostGIS-Erweiterung sind `rds_superuser`-Berechtigungen erforderlich. Wir empfehlen Ihnen, einen Benutzer (Rolle) zum Verwalten der PostGIS-Erweiterung und Ihrer räumlichen Daten zu erstellen. Die PostGIS-Erweiterung und ihre zugehörigen Komponenten fügen PostgreSQL Tausende von Funktionen hinzu. Erstellen Sie die PostGIS-Erweiterung ggf. in einem eigenen Schema, wenn dies für Ihren Anwendungsfall sinnvoll ist. Das folgende Beispiel zeigt, wie die Erweiterung in einer eigenen Datenbank installiert wird. Dies ist jedoch nicht erforderlich.

Themen

- Schritt 1: Erstellen Sie einen Benutzer (Rolle) zum Verwalten der PostGIS-Erweiterung.
- Schritt 2: Laden der PostGIS-Erweiterungen.
- Schritt 3: Übertragen der Eigentümerschaft der Erweiterungsschemas
- Schritt 4: Übertragen der Eigentümerschaft der PostGIS-Tabellen
- Schritt 5: Testen der Erweiterungen
- Schritt 6: Upgrade der PostGIS-Erweiterung
- PostGIS-Erweiterungsversionen
- Upgrade von PostGIS 2 auf PostGIS 3

Schritt 1: Erstellen Sie einen Benutzer (Rolle) zum Verwalten der PostGIS-Erweiterung.

Zuerst stellen Sie eine Verbindung mit Ihrer DB-Instance von RDS für PostgreSQL als Benutzer her, der über `rds_superuser`-Berechtigungen verfügt. Wenn Sie beim Einrichten Ihrer Instance den Standardnamen beibehalten haben, stellen Sie eine Verbindung als `postgres` her.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

Erstellen Sie eine separate Rolle (Benutzer), um die PostGIS-Erweiterung zu verwalten.

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

Gewähren Sie dieser Rolle `rds_superuser`-Berechtigungen, damit die Rolle die Erweiterung installieren kann.

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

Erstellen Sie eine Datenbank, die Sie für Ihre PostGIS-Artefakte verwenden können. Dieser Schritt ist optional. Oder Sie können in Ihrer Benutzerdatenbank ein Schema für die PostGIS-Erweiterungen erstellen, aber das ist auch nicht erforderlich.

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

Gewähren Sie `gis_admin` alle Berechtigungen für die `lab_gis`-Datenbank.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

Beenden Sie die Sitzung und stellen Sie wieder eine Verbindung mit Ihrer RDS-for-PostgreSQL-DB-Instance als `gis_admin` her.

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

Fahren Sie mit der Einrichtung der Erweiterung fort, wie in den nächsten Schritten beschrieben.

Schritt 2: Laden der PostGIS-Erweiterungen.

Die PostGIS-Erweiterung enthält mehrere verwandte Erweiterungen, die zusammenarbeiten, um Geodatenfunktionen bereitzustellen. Je nach Anwendungsfall benötigen Sie möglicherweise nicht alle Erweiterungen, die Sie in diesem Schritt erstellt haben.

Verwenden Sie `CREATE EXTENSION`-Anweisungen, um die PostGIS-Erweiterungen zu laden.

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
```

```
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

Sie können die Ergebnisse überprüfen, indem Sie die in dem folgenden Beispiel gezeigte SQL-Abfrage ausführen, in der die Erweiterungen und ihre Besitzer aufgeführt sind.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
 topology     | rdsadmin
(4 rows)
```

Schritt 3: Übertragen der Eigentümerschaft der Erweiterungsschemas

Verwenden Sie die ALTER SCHEMA-Anweisungen, um die Eigentümerschaft der Schemata auf die Rolle `gis_admin` zu übertragen.

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

Sie können die Eigentümeränderung bestätigen, indem Sie die folgende SQL-Abfrage ausführen. Oder Sie können den Metabefehl `\dn` über die psql-Befehlszeile verwenden.

```
SELECT n.nspname AS "Name",
```

```
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

       List of schemas
     Name     |     Owner
--------------+---------------
 public       | postgres
 tiger        | gis_admin
 tiger_data   | gis_admin
 topology     | gis_admin
(4 rows)
```

Schritt 4: Übertragen der Eigentümerschaft der PostGIS-Tabellen

> **Note**
>
> Ändern Sie nicht die Eigentümerschaft der PostGIS-Funktionen. Ein ordnungsgemäßer Betrieb sowie zukünftige Upgrades von PostGIS setzen voraus, dass sich an der Eigentümerschaft dieser Funktionen nichts ändert. Weitere Informationen zu PostGIS-Berechtigungen finden Sie unter PostgreSQL-Sicherheit.

Verwenden Sie die folgende Funktion, um die Eigentümerschaft der PostGIS-Tabellen auf die Rolle `gis_admin` zu übertragen. Führen Sie die folgende Anweisung aus der psql-Aufforderung aus, um die Funktion zu erstellen.

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

Führen Sie als Nächstes die folgende Abfrage aus, um die `exec`-Funktion auszuführen, die wiederum die Anweisungen ausführt und die Berechtigungen ändert.

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
```

```
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

Schritt 5: Testen der Erweiterungen

Um zu vermeiden, dass der Schemaname angegeben werden muss, fügen Sie das `tiger`-Schema Ihrem Suchpfad unter Verwendung des folgenden Befehls hinzu.

```
SET search_path=public,tiger;
SET
```

Testen Sie das `tiger`-Schema unter Verwendung der folgenden SELECT-Anweisung.

```
SELECT address, streetname, streettypeabbrev, zip
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

Weitere Informationen zu dieser Erweiterung finden Sie unter Tiger Geocoder in der PostGIS-Dokumentation.

Testen Sie den Zugriff auf das `topology`-Schema, indem Sie folgende `SELECT`-Anweisung verwenden. Das ruft die `createtopology`-Funktion zum Registrieren eines neuen Topologieobjekts (my_new_topo) mit der angegebenen Raumbezugskennung (26986) und der Standardtoleranz (0,5) auf. Weitere Informationen finden Sie unter CreateTopology (Topologie erstellen) in der PostgreSQL-Dokumentation.

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
----------------
              1
(1 row)
```

Schritt 6: Upgrade der PostGIS-Erweiterung

Jede neue Version von PostgreSQL unterstützt eine oder mehrere Versionen der PostGIS-Erweiterung, die mit dieser Version kompatibel sind. Bei einem Upgrade der PostgreSQL-Engine auf eine neue Version wird die PostGIS-Erweiterung nicht automatisch aktualisiert. Vor dem Upgrade der

PostgreSQL-Engine aktualisieren Sie PostGIS in der Regel auf die neueste verfügbare Version für die aktuelle PostgreSQL-Version. Details hierzu finden Sie unter PostGIS-Erweiterungsversionen.

Nach dem Upgrade der PostgreSQL-Engine aktualisieren Sie die PostGIS-Erweiterung erneut auf die Version, die für die neu aktualisierte PostgreSQL-Engine-Version unterstützt wird. Weitere Informationen zum Upgrade der PostgreSQL-Engine finden Sie unter Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion. .

Sie können jederzeit prüfen, ob eine neue Version der PostGIS-Erweiterung auf dem Aurora-PostgreSQL-DB-Cluster verfügbar ist. Führen Sie dazu den folgenden Befehl aus. Diese Funktion ist mit PostGIS 2.5.0 und höheren Versionen verfügbar.

```
SELECT postGIS_extensions_upgrade();
```

Wenn Ihre Anwendung die neueste PostGIS-Version nicht unterstützt, können Sie weiterhin eine ältere Version von PostGIS installieren, die in Ihrer Hauptversion wie folgt verfügbar ist.

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

Wenn Sie von einer älteren Version auf eine bestimmte PostGIS-Version aktualisieren möchten, können Sie auch den folgenden Befehl verwenden.

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

Abhängig von der Version, von der Sie ein Upgrade durchführen, müssen Sie diese Funktion möglicherweise noch einmal verwenden. Das Ergebnis des ersten Durchlaufs der Funktion bestimmt, ob eine zusätzliche Upgrade-Funktion benötigt wird. Dies ist beispielsweise bei einem Upgrade von PostGIS 2 auf PostGIS 3 der Fall. Weitere Informationen finden Sie unter Upgrade von PostGIS 2 auf PostGIS 3.

Wenn Sie diese Erweiterung aktualisiert haben, um ein Upgrade der Hauptversion der PostgreSQL-Engine vorzubereiten, können Sie mit anderen vorbereitenden Aufgaben fortfahren. Weitere Informationen finden Sie unter Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion.

PostGIS-Erweiterungsversionen

Wir empfehlen Ihnen, die Versionen aller Erweiterungen wie PostGIS zu installieren, wie sie unter Extension versions for Aurora PostgreSQL-Compatible Edition (Versionen aller Erweiterungen für

Editionen, die mit Aurora PostgreSQL kompatibel sind) in Versionshinweise für Aurora PostgreSQL aufgelistet sind. . Sie können mithilfe des folgenden Befehls auflisten, welche Versionen in Ihrer Version verfügbar sind.

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

Versionsinformationen finden Sie in den folgenden Abschnitten in den Versionshinweisen zu Aurora PostgreSQL:

- Erweiterungsversionen für Aurora PostgreSQL 14
- Extension versions for Aurora PostgreSQL-Compatible Edition 13 (Versionen aller Erweiterungen für Edition 13, die mit Aurora PostgreSQL kompatibel ist)
- Extension versions for Aurora PostgreSQL-Compatible Edition 12 (Versionen aller Erweiterungen für Edition 12, die mit Aurora PostgreSQL kompatibel ist)
- Extension versions for Aurora PostgreSQL-Compatible Edition 11 (Versionen aller Erweiterungen für Edition 11, die mit Aurora PostgreSQL kompatibel ist)
- Extension versions for Aurora PostgreSQL-Compatible Edition10 (Versionen aller Erweiterungen für Edition 10, die mit Aurora PostgreSQL kompatibel ist)
- Extension versions for Aurora PostgreSQL-Compatible Edition 9.6 (Versionen aller Erweiterungen für Edition 9.6, die mit Aurora PostgreSQL kompatibel ist)

Upgrade von PostGIS 2 auf PostGIS 3

Ab Version 3.0 ist die PostGIS-Raster-Funktionalität jetzt eine separate Erweiterung, `postgis_raster`. Diese Erweiterung hat einen eigenen Installations- und Upgrade-Pfad. Dadurch werden Dutzende von Funktionen, Datentypen und anderen Artefakten, die für die Rasterbildverarbeitung erforderlich sind, aus der `postgis`-Kernerweiterung entfernt. Das heißt, wenn Ihr Anwendungsfall keine Raster-Verarbeitung erfordert, müssen Sie die `postgis_raster`-Erweiterung nicht installieren.

Im folgenden Upgrade-Beispiel extrahiert der erste Upgrade-Befehl die Raster-Funktionalität in die `postgis_raster`-Erweiterung. Ein zweiter Upgrade-Befehl ist dann erforderlich, um ein Upgrade von `postgis_raster` auf die neue Version durchzuführen.

So führen Sie ein Upgrade von PostGIS 2 auf PostGIS 3 durch

1. Identifizieren Sie die Standardversion von PostGIS, die für die PostgreSQL-Version auf Ihrem Aurora-PostgreSQL-DB-Cluster verfügbar ist. Führen Sie dazu die folgende Abfrage durch.

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name   | default_version | installed_version |                          comment
---------+-----------------+-------------------
+-------------------------------------------------------------
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
(1 row)
```

2. Identifizieren Sie die Versionen von PostGIS, die in jeder Datenbank auf der Writer-Instance Ihres Aurora-PostgreSQL-DB-Clusters installiert sind. Anders gesagt: Fragen Sie jede Benutzerdatenbank wie folgt ab.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
  Name   | Version | Schema |                            Description
---------+---------+--------
+-------------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

   Diese Nichtübereinstimmung zwischen der Standardversion (PostGIS 3.1.4) und der installierten Version (PostGIS 2.3.7) bedeutet, dass Sie die PostGIS-Erweiterung aktualisieren müssen.

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. Führen Sie die folgende Abfrage aus, um sicherzustellen, dass sich die Raster-Funktionalität jetzt in einem eigenen Paket befindet.

```
SELECT
    probin,
    count(*)
FROM
    pg_proc
WHERE
    probin LIKE '%postgis%'
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
(2 rows)
```

Die Ausgabe zeigt, dass es immer noch einen Unterschied zwischen den Versionen gibt. Die PostGIS-Funktionen sind Version 3 (postgis-3), während die Raster-Funktionen (rtpostgis) Version 2 (rtpostgis-2.3) sind. Um das Upgrade abzuschließen, führen Sie den Upgrade-Befehl erneut wie folgt aus.

```
postgres=> SELECT postgis_extensions_upgrade();
```

Die Warnmeldungen können Sie ohne Bedenken ignorieren. Führen Sie die folgende Abfrage erneut aus, um zu überprüfen, ob das Upgrade abgeschlossen ist. Das Upgrade ist abgeschlossen, wenn PostGIS und alle zugehörigen Erweiterungen nicht als aktualisierungsbedürftig gekennzeichnet sind.

```
SELECT postgis_full_version();
```

4. Verwenden Sie die folgende Abfrage, um den abgeschlossenen Upgrade-Vorgang und die separat gepackten Erweiterungen anzuzeigen und zu überprüfen, ob ihre Versionen übereinstimmen.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
      Name      | Version | Schema |                            Description
----------------+---------+--------
+------------------------------------------------------------------------
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

   Die Ausgabe zeigt, dass die PostGIS-2-Erweiterung auf PostGIS 3 aktualisiert wurde und beide `postgis` und die jetzt getrennte `postgis_raster`-Erweiterungen Version 3.1.5 sind.

Wenn Sie nach Abschluss dieses Upgrades die Raster-Funktionalität nicht verwenden möchten, können Sie die Erweiterung wie folgt löschen.

```
DROP EXTENSION postgis_raster;
```

Verwalten von PostgreSQL-Partitionen mit der Erweiterung pg_partman

Die PostgreSQL-Tabellenpartitionierung bietet ein Framework für den leistungsstarken Umgang mit Dateneingaben und Berichten. Verwenden Sie die Partitionierung für Datenbanken, die eine sehr schnelle Eingabe großer Datenmengen erfordern. Die Partitionierung ermöglicht auch

schnellere Abfragen großer Tabellen. Die Partitionierung hilft bei der Verwaltung von Daten, ohne die Datenbank-Instance zu beeinträchtigen, da sie weniger I/O-Ressourcen benötigt.

Durch die Partitionierung können Sie Daten zur Verarbeitung in benutzerdefinierte Blöcke aufteilen. Sie können beispielsweise Zeitreihendaten für Bereiche wie stündlich, täglich, wöchentlich, monatlich, vierteljährlich, jährlich, benutzerdefiniert oder eine beliebige Kombination davon partitionieren. Wenn Sie für ein Beispiel für Zeitreihendaten die Tabelle nach Stunden partitionieren, enthält jede Partition die Daten einer Stunde. Wenn Sie die Zeitreihentabelle nach Tag partitionieren, enthalten die Partitionen die Daten eines Tages und so weiter. Der Partitionsschlüssel steuert die Größe einer Partition.

Wenn Sie den SQL-Befehl `INSERT` oder `UPDATE` für eine partitionierte Tabelle verwenden, leitet die Datenbank-Engine die Daten an die entsprechende Partition weiter. PostgreSQL-Tabellenpartitionen, die die Daten speichern, sind untergeordnete Tabellen der Haupttabelle.

Während der Lesevorgänge der Datenbankabfrage untersucht der PostgreSQL-Optimierer die `WHERE`-Klausel der Abfrage und leitet den Datenbank-Scan nach Möglichkeit nur an die relevanten Partitionen weiter.

Beginnend mit Version 10 verwendet PostgreSQL deklarative Partitionierung, um die Tabellenpartitionierung zu implementieren. Dies wird auch als native PostgreSQL-Partitionierung bezeichnet. Vor PostgreSQL Version 10 wurden Auslöser verwendet, um Partitionen zu implementieren.

Die PostgreSQL-Tabellenpartitionierung bietet die folgenden Funktionen:

- Erstellung neuer Partitionen zu jeder Zeit.
- Variable Partitionsbereiche.
- Entfernbare und wiederverwendbare Partitionen mit DDL-Anweisungen (Data Definition Language).

  Zum Beispiel sind entfernbare Partitionen nützlich, um Verlaufsdaten aus der Hauptpartition zu entfernen, aber Verlaufsdaten für die Analyse zu behalten.
- Neue Partitionen erben die Eigenschaften der übergeordneten Datenbanktabelle, einschließlich folgender Eigenschaften:
  - Indizes
  - Primärschlüssel, die die Partitionsschlüsselspalte enthalten müssen
  - Fremdschlüssel
  - Einschränkungen prüfen

  - Referenzen
- Erstellen von Indizes für die vollständige Tabelle oder jede spezifische Partition.

Sie können das Schema für eine einzelne Partition nicht ändern. Sie können jedoch die übergeordnete Tabelle ändern (z. B. das Hinzufügen einer neuen Spalte), die auf Partitionen übertragen wird.

Themen

- Übersicht über die PostgreSQL-Erweiterung pg_partman
- Aktivieren der Erweiterung pg_partman
- Konfigurieren von Partitionen mit der create_parent-Funktion
- Konfigurieren der Partitionspflege mit der run_maintenance_proc-Funktion

Übersicht über die PostgreSQL-Erweiterung pg_partman

Sie können die PostgreSQL-Erweiterung `pg_partman` verwenden, um die Erstellung und Pflege von Tabellenpartitionen zu automatisieren. Weitere allgemeine Informationen finden Sie unter PG-Partitions-Manager in der `pg_partman`-Dokumentation.

> **Note**
>
> Die Erweiterung `pg_partman` wird auf den Aurora-PostgreSQL-Versionen 12.6 und höher unterstützt.

Anstatt jede Partition manuell erstellen zu müssen, konfigurieren Sie `pg_partman` mit den folgenden Einstellungen:

- Zu partitionierende Tabelle
- Partitionstyp
- Partitionsschlüssel
- Granularität der Partition
- Optionen für die Erstellung und Verwaltung von Partitionen

Nachdem Sie eine partitionierte PostgreSQL-Tabelle erstellt haben, registrieren Sie diese mit `pg_partman`, indem Sie die Funktion `create_parent` aufrufen. Dadurch werden die erforderlichen Partitionen basierend auf den Parametern erstellt, die Sie an die Funktion übergeben.

Die Erweiterung `pg_partman` bietet auch die Funktion `run_maintenance_proc`, die Sie planmäßig aufrufen können, um Partitionen automatisch zu verwalten. Planen Sie, dass diese Funktion regelmäßig (z. B. stündlich) ausgeführt wird, um sicherzustellen, dass die richtigen Partitionen nach Bedarf erstellt werden. Sie können auch sicherstellen, dass Partitionen automatisch gelöscht werden.

Aktivieren der Erweiterung pg_partman

Wenn Sie mehrere Datenbanken innerhalb derselben PostgreSQL-DB-Instance haben, für die Sie Partitionen verwalten möchten, aktivieren Sie die Erweiterung `pg_partman` für jede Datenbank separat. Um die Erweiterung `pg_partman` für eine bestimmte Datenbank zu aktivieren, erstellen Sie das Partitionswartungsschema und dann die Erweiterung `pg_partman` wie folgt:

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> Um die Erweiterung `pg_partman` zu erstellen, müssen Sie sicherstellen, dass Sie über `rds_superuser`-Berechtigungen verfügen.

Wenn Sie einen Fehler wie den folgenden erhalten, erteilen Sie dem Konto die Berechtigungen `rds_superuser` oder verwenden Sie Ihr Superuser-Konto.

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

Um die Berechtigungen `rds_superuser` zu erteilen, verbinden Sie sich mit Ihrem Superuser-Konto und führen Sie den folgenden Befehl aus.

```
GRANT rds_superuser TO user-or-role;
```

Für die Beispiele, die die Verwendung der Erweiterung pg_partman zeigen, verwenden wir die folgende Beispieldatenbanktabelle und Partition. Diese Datenbank verwendet eine partitionierte

Tabelle basierend auf einem Zeitstempel. Ein `data_mart`-Schema enthält eine Tabelle mit dem Namen `events` mit einer Spalte namens `created_at`. Die folgenden Einstellungen sind in der `events`-Tabelle enthalten:

- Primärschlüssel `event_id` und `created_at`, die die Spalte zur Führung der Partition verwenden müssen.
- Eine CHECK-Beschränkung `ck_valid_operation` zum Durchsetzen von Werten für eine `operation`-Tabellenspalte.
- Zwei Fremdschlüssel, wobei einer `fk_orga_membership)` auf die externe Tabelle `organization` verweist und der andere (`fk_parent_event_id`) ein selbst referenzierter Fremdschlüssel ist.
- Zwei Indizes, wobei einer (`idx_org_id`) für den Fremdschlüssel und der andere (`idx_event_type`) für den Ereignistyp steht.

Die folgenden DDL-Anweisungen erstellen diese Objekte, die automatisch auf jeder Partition enthalten sind.

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
        value           FLOAT(24),
        parent_event_id BIGINT,
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);
```

```
CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

Konfigurieren von Partitionen mit der create_parent-Funktion

Nachdem Sie die Erweiterung `pg_partman` aktiviert haben, verwenden Sie die `create_parent`-Funktion, um Partitionen innerhalb des Partitionswartungsschemas zu konfigurieren. Im folgenden Beispiel wird das `events`-Tabellenbeispiel verwendet, das in Aktivieren der Erweiterung pg_partman erstellt wurde. Rufen Sie die `create_parent`-Funktion wie folgt auf.

```
SELECT partman.create_parent(
 p_parent_table => 'data_mart.events',
 p_control      => 'created_at',
 p_type         => 'range',
 p_interval     => '1 day',
 p_premake      => 30);
```

Dabei werden die folgenden Parameter verwendet:

- `p_parent_table` – Die übergeordnete partitionierte Tabelle. Diese Tabelle muss bereits existieren und einschließlich des Schemas vollständig qualifiziert sein.
- `p_control` – Die Spalte, auf der die Partitionierung basieren soll. Der Datentyp muss ganzzahlig oder zeitbasiert sein.
- `p_type` – Der Typ ist entweder `'range'` oder `'list'`.
- `p_interval` – Das Zeitintervall oder der Ganzzahlbereich für jede Partition. Beispielwerte sind `1 day`, `1 hour` usw.
- `p_premake` – Die Anzahl der Partitionen, die im Voraus erstellt werden müssen, um neue Inserts zu unterstützen.

Eine vollständige Beschreibung der Funktion `create_parent` finden Sie in der `pg_partman`-Dokumentation unter Creation Functions (Erstellungsfunktionen).

Konfigurieren der Partitionspflege mit der run_maintenance_proc-Funktion

Sie können Partitionswartungsvorgänge ausführen, um automatisch neue Partitionen zu erstellen, Partitionen zu trennen oder alte Partitionen zu entfernen. Die Partitionspflege beruht

auf der Funktion `run_maintenance_proc` von der `pg_partman`-Erweiterung und der Erweiterung `pg_cron`, die einen internen Scheduler initiiert. Der `pg_cron`-Scheduler führt automatisch SQL-Anweisungen, -Funktionen und -Prozesse aus, die in Ihren Datenbanken definiert sind.

Im folgenden Beispiel wird das `events`-Tabellenbeispiel verwendet, das in Aktivieren der Erweiterung pg_partman erstellt wurde, um die automatische Ausführung der Partitionswartung festzulegen. Fügen Sie als Voraussetzung `pg_cron` zum Parameter `shared_preload_libraries` in der Parametergruppe der DB-Instance hinzu.

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

Nachfolgend finden Sie eine Schritt-für-Schritt-Erklärung für das vorangehende Beispiel:

1. Ändern Sie die mit Ihrer DB-Instance verknüpfte Parametergruppe und fügen Sie `pg_cron` dem `shared_preload_libraries`-Parameterwert hinzu. Diese Änderung erfordert einen Neustart der DB-Instance, damit sie wirksam wird. Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.
2. Führen Sie den Befehl `CREATE EXTENSION pg_cron;` mit einem Konto aus, das die `rds_superuser`-Berechtigungen besitzt. Dadurch wird die Erweiterung `pg_cron` aktiviert. Weitere Informationen finden Sie unter Planen der Wartung mit der PostgreSQL-Erweiterung pg_cron.
3. Führen Sie den Befehl `UPDATE partman.part_config` aus, um die `pg_partman`-Einstellungen für die Tabelle `data_mart.events` anzupassen.
4. Führen Sie den Befehl aus `SET . . .` um die `data_mart.events`-Tabelle mit den folgenden Klauseln zu konfigurieren:
   a. `infinite_time_partitions = true,` – Konfiguriert die Tabelle so, dass automatisch neue Partitionen ohne Begrenzung erstellt werden können.
   b. `retention = '3 months',` – Konfiguriert die Tabelle so, dass sie eine maximale Beibehaltung von drei Monaten hat.

c. `retention_keep_table=true` – Konfiguriert die Tabelle so, dass die Tabelle bei Fälligkeit der Aufbewahrungsfrist nicht automatisch gelöscht wird. Stattdessen werden Partitionen, die älter als die Aufbewahrungsfrist sind, nur von der übergeordneten Tabelle getrennt.

5. Führen Sie den Befehl aus `SELECT cron.schedule . . .` um einen `pg_cron`-Funktionsaufruf zu machen. Dieser Aufruf definiert, wie oft der Scheduler das `pg_partman`-Wartungsverfahren `partman.run_maintenance_proc` ausführt. In diesem Beispiel wird der Prozess stündlich ausgeführt.

Eine vollständige Beschreibung der `run_maintenance_proc`-Funktion finden Sie in der `pg_partman`-Dokumentation unter Maintenance Functions (Wartungsfunktionen).

Planen der Wartung mit der PostgreSQL-Erweiterung pg_cron

Sie können die PostgreSQL-Erweiterung `pg_cron` verwenden, um Wartungsbefehle innerhalb einer PostgreSQL-Datenbank zu planen. Weitere Informationen zu der Erweiterung finden Sie unter Was ist pg_cron? in der pg_cron-Dokumentation.

Die Erweiterung `pg_cron` wird von Aurora-PostgreSQL-Versionen 12.6 und höher unterstützt.

Weitere Informationen zur Verwendung von `pg_cron` finden Sie unter Planen von Aufträgen mit pg_cron auf RDS für PostgreSQL oder Ihren Datenbanken der mit Aurora PostgreSQL kompatiblen Edition.

> **Note**
>
> Die `pg_cron`-Erweiterungsversion wird in der Ansicht „pg_available_extensions“ als zweistellige Version angezeigt, beispielsweise 1.6. In einigen Kontexten werden möglicherweise dreistellige Versionen aufgeführt, z. B. 1.6.4 oder 1.6.5, Sie müssen jedoch die zweistellige Version angeben, wenn Sie ein Upgrade der Erweiterung durchführen.

Themen

- Einrichten der pg_con-Erweiterung
- Gewähren von Berechtigungen zur Verwendung von pg_cron für Datenbankbenutzer
- Planen von pg_cron-Aufträgen
- Referenz für die pg_cron-Erweiterung

Einrichten der pg_con-Erweiterung

Richten Sie die Erweiterung `pg_cron` wie folgt ein:

1. Ändern Sie die benutzerdefinierte Parametergruppe, die mit Ihrer PostgreSQL-DB-Instance verknüpft ist, indem Sie `pg_cron` dem Parameterwert `shared_preload_libraries` hinzufügen.

   Starten Sie die PostgreSQL-DB-Instance neu, damit die Änderungen an der Parametergruppe in Kraft treten. Weitere Informationen zum Arbeiten mit Parametergruppen finden Sie unter Amazon-Aurora-PostgreSQL-Parameter.
2. Nachdem die PostgreSQL-DB-Instance neu gestartet wurde, führen Sie den folgenden Befehl mit einem Konto aus, das über `rds_superuser`-Berechtigungen verfügt. Wenn Sie beispielsweise beim Erstellen Ihres Aurora-PostgreSQL-DB-Clusters die Standardeinstellungen verwendet haben, verbinden Sie sich als Benutzer `postgres` und erstellen Sie die Erweiterung.

   ```
   CREATE EXTENSION pg_cron;
   ```

   Der Scheduler `pg_cron` ist in der standardmäßigen PostgreSQL-Datenbank mit dem Namen `postgres` festgelegt. Die Objekte `pg_cron` werden in dieser `postgres`-Datenbank erstellt und alle Planungsaktionen werden in dieser Datenbank ausgeführt.
3. Sie können die Standardeinstellungen verwenden oder Aufgaben für die Ausführung in anderen Datenbanken innerhalb Ihrer PostgreSQL-DB-Instance planen. Informationen zum Planen von Aufgaben für andere Datenbanken innerhalb Ihrer PostgreSQL-DB-Instance finden Sie im Beispiel unter Planen einer Cron-Aufgabe für eine andere als die Standard-Datenbank.

Gewähren von Berechtigungen zur Verwendung von pg_cron für Datenbankbenutzer

Zum Installieren der Erweiterung `pg_cron` sind `rds_superuser`-Berechtigungen erforderlich. Berechtigungen zur Verwendung von `pg_cron` können anderen Datenbankbenutzern (von einem Mitglied der Gruppe/Rolle `rds_superuser`) gewährt werden, damit diese ihre eigenen Aufträge planen können. Wir empfehlen Ihnen, dem `cron`-Schema Berechtigungen nur nach Bedarf, wenn es die Abläufe in Ihrer Produktionsumgebung verbessert, zu gewähren.

Führen Sie den folgenden Befehl aus, um einem Datenbankbenutzer Berechtigungen im `cron`-Schema zu erteilen:

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

Dies gibt die *db-user* Erlaubnis, auf das `cron` Schema zuzugreifen, um Cron-Jobs für die Objekte zu planen, für die sie Zugriffsberechtigungen haben. Wenn der Datenbankbenutzer keine Berechtigungen hat, schlägt der Auftrag fehl, nachdem die Fehlermeldung in der `postgresql.log`-Datei angezeigt wurde, wie nachfolgend dargestellt:

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

Mit anderen Worten, stellen Sie sicher, dass Datenbankbenutzer, denen Berechtigungen für das `cron`-Schemas gewährt werden, auch über Berechtigungen für die Objekte (Tabellen, Schemas usw.) verfügen, die sie planen wollen.

Die Details des cron-Auftrags und dessen Erfolg oder Misserfolg werden ebenfalls in der Tabelle `cron.job_run_details` erfasst. Weitere Informationen finden Sie unter Tabellen zum Planen von Jobs und zur Erfassung des Status .

Planen von pg_cron-Aufträgen

Die folgenden Abschnitte zeigen, wie Sie verschiedene Verwaltungsaufgaben mit `pg_cron`-Aufträgen planen können.

> **Note**
>
> Wenn Sie `pg_cron`-Aufträge erstellen, überprüfen Sie, dass die Einstellung `max_worker_processes` größer ist als die Anzahl von `cron.max_running_jobs`. Ein `pg_cron`-Auftrag schlägt fehl, wenn keine Hintergrund-Workerprozesse ausgeführt werden. Die Standardanzahl von `pg_cron`-Aufträgen ist `5`. Weitere Informationen finden Sie unter Parameter für die Verwaltung der pg_cron-Erweiterung.

Themen

- Bereinigen von Tabellen
- Löschen der Verlaufstabelle pg_cron
- Protokollieren von Fehlern nur in der Datei postgresql.log
- Planen einer Cron-Aufgabe für eine andere als die Standard-Datenbank

Bereinigen von Tabellen

Autovacuum übernimmt die Entfernung für die meisten Fälle Möglicherweise möchten Sie jedoch eine Bereinigung für eine bestimmte Tabelle zu einem Zeitpunkt Ihrer Wahl planen.

Es folgt ein Beispiel für die Verwendung der Funktion `cron.schedule` zum Einrichten eines Auftrags, der jeden Tag um 22:00 Uhr (GMT) `VACUUM FREEZE` auf eine bestimmte Tabelle anwenden soll.

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

Nachdem das vorangehende Beispiel ausgeführt wurde, können Sie den Verlauf in der `cron.job_run_details`-Tabelle wie folgt überprüfen.

```
postgres=> SELECT * FROM cron.job_run_details;
jobid  | runid | job_pid | database | username | command                        |
 status    | return_message | start_time                    | end_time
-------+-------+---------+----------+----------+--------------------------------
+-----------+----------------+-------------------------------
+--------------------------------
 1     | 1     | 3395    | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

Nachfolgend finden Sie eine Abfrage der Tabelle `cron.job_run_details`, um fehlgeschlagene Aufträge anzuzeigen.

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                        | status
 | return_message                                  | start_time                    |
 end_time
------+-------+---------+----------+----------+--------------------------------+--------
+-------------------------------------------------+-------------------------------
+-------------------------------
 5    | 4     | 30339   | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
```

```
(1 row)
```

Weitere Informationen finden Sie unter Tabellen zum Planen von Jobs und zur Erfassung des Status .

Löschen der Verlaufstabelle pg_cron

Die `cron.job_run_details`-Tabelle enthält einen Verlauf von Cron-Aufgaben, die im Laufe der Zeit sehr groß werden können. Wir empfehlen Ihnen, eine Aufgabe zu planen, die diese Tabelle bereinigt. Beispielsweise kann es für die Fehlerbehebung ausreichen, die Eingaben einer Woche beizubehalten.

Im folgenden Beispiel wird die cron.schedule-Funktion verwendet, um eine Aufgabe zu planen, die jeden Tag um Mitternacht ausgeführt wird, um die `cron.job_run_details`-Tabelle zu bereinigen. Die Aufgabe behält nur die letzten sieben Tage. Verwenden Sie Ihr `rds_superuser`-Konto, um die Aufgabe wie folgt zu planen.

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

Weitere Informationen finden Sie unter Tabellen zum Planen von Jobs und zur Erfassung des Status .

Protokollieren von Fehlern nur in der Datei postgresql.log

Wenn Sie verhindern möchten, dass in die `cron.job_run_details`-Tabelle geschrieben wird, ändern Sie die mit der PostgreSQL-DB-Instance verknüpfte Parametergruppe und deaktivieren Sie den Parameter `cron.log_run`. Es werden keine Schreibvorgänge der `pg_cron`-Erweiterung in die Tabelle mehr durchgeführt und nur noch Fehler in der `postgresql.log`-Datei erfasst. Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.

Verwenden Sie den folgenden Befehl, um den Wert des `cron.log_run`-Parameters zu überprüfen.

```
postgres=> SHOW cron.log_run;
```

Weitere Informationen finden Sie unter Parameter für die Verwaltung der pg_cron-Erweiterung.

Planen einer Cron-Aufgabe für eine andere als die Standard-Datenbank

Die Metadaten für `pg_cron` werden alle in der PostgreSQL-Standarddatenbank mit dem Namen `postgres` gespeichert. Da Hintergrund-Worker für die Ausführung der Maintenance-Cron-Aufgaben

verwendet werden, können Sie eine Aufgabe in jeder Ihrer Datenbanken innerhalb der PostgreSQL-DB-Instance planen.

> **Note**
>
> Nur Benutzer mit der Rolle `rds_superuser` oder mit `rds_superuser`-Berechtigungen können alle cron-Aufträge in der Datenbank auflisten. Andere Benutzer können nur ihre eigenen Aufträge in der Tabelle `cron.job` anzeigen.

1. Planen Sie die Aufgabe in der Cron-Datenbank wie gewohnt mit cron.schedule.

```
postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
 freeze test_table');
```

2. Aktualisieren Sie als Benutzer mit der `rds_superuser`-Rolle die Datenbankspalte für die soeben erstellte Aufgabe, damit sie in einer anderen Datenbank innerhalb Ihrer PostgreSQL-DB-Instance ausgeführt wird.

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. Überprüfen Sie dies, indem Sie die `cron.job`-Tabelle abfragen.

```
postgres=> SELECT * FROM cron.job;
jobid | schedule    | command                        | nodename  | nodeport |
 database | username  | active | jobname
------+-------------+--------------------------------+-----------+----------
+----------+-----------+--------+------------------------
106   | 29 03 * * * | vacuum freeze test_table       | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1   | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> In einigen Fällen können Sie eine Cron-Aufgabe hinzufügen, die Sie in einer anderen Datenbank ausführen möchten. Bevor Sie die richtige Datenbankspalte aktualisieren, versucht die Aufgabe in solchen Fällen möglicherweise, in der Standarddatenbank

(`postgres`) ausgeführt zu werden. Wenn der Benutzername über Berechtigungen verfügt, wird die Aufgabe erfolgreich in der Standarddatenbank ausgeführt.

Referenz für die pg_cron-Erweiterung

Sie können die folgenden Parameter, Funktionen und Tabellen mit der `pg_cron`-Erweiterung verwenden. Weitere Informationen finden Sie unter Was ist pg_cron? in der pg_cron-Dokumentation.

Themen

- Parameter für die Verwaltung der pg_cron-Erweiterung
- Funktionsreferenz: cron.schedule
- Funktionsreferenz: cron.unschedule
- Tabellen zum Planen von Jobs und zur Erfassung des Status

Parameter für die Verwaltung der pg_cron-Erweiterung

Es folgt eine Liste der Parameter zur Steuerung des Erweiterungsverhaltens von `pg_cron`.

Parameter	Description
cron.database_name	Die Datenbank, in der `pg_cron`-Metadaten aufbewahrt werden.
`cron.host`	Der Hostname für die Verbindung mit PostgreSQL. Dieser Wert kann nicht verändert werden.
`cron.log_run`	Protokollieren Sie jeden ausgeführten Auftrag in der Tabelle `job_run_details` . Die Werte sind `on` oder `off`. Weitere Informationen finden Sie unter Tabellen zum Planen von Jobs und zur Erfassung des Status .
`cron.log_statement`	Protokolliert alle Cron-Anweisungen, bevor Sie ausgeführt werden. Die Werte sind `on` oder `off`.

Parameter	Description
`cron.max_running_jobs`	Die maximale Anzahl von Aufgaben, die gleichzeitig ausgeführt werden können.
`cron.use_background_workers`	Verwenden Sie Hintergrund-Worker anstelle von Client-Sitzungen. Dieser Wert kann nicht verändert werden.

Verwenden Sie den folgenden SQL-Befehl, um diese Parameter und ihre Werte anzuzeigen.

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

Funktionsreferenz: cron.schedule

Diese Funktion plant eine Cron-Aufgabe. Die Aufgabe ist anfänglich in der `postgres` Standarddatenbank geplant. Die Funktion gibt einen `bigint` Wert zurück, der den Aufgabenbezeichner darstellt. Informationen zum Planen von Aufgaben für die Ausführung in anderen Datenbanken innerhalb Ihrer PostgreSQL-DB-Instance finden Sie im Beispiel unter Planen einer Cron-Aufgabe für eine andere als die Standard-Datenbank.

Die Funktion hat zwei Syntaxformate.

Syntax

```
cron.schedule (job_name,
    schedule,
    command
);

cron.schedule (schedule,
    command
);
```

Parameter

Parameter	Beschreibung
`job_name`	Der Name der Cron-Aufgabe.
`schedule`	Text, der den Zeitplan für die Cron-Aufgabe angibt. Das Format ist das Standard-Cron-Format.
`command`	Text des auszuführenden Befehls.

Beispiele

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

Funktionsreferenz: cron.unschedule

Diese Funktion löscht eine Cron-Aufgabe. Sie können entweder den `job_name` oder die `job_id` angeben. Eine Richtlinie stellt sicher, dass Sie der Besitzer sind, um den Plan für die Aufgabe zu entfernen. Die Funktion gibt einen Booleschen Wert zurück, der Erfolg oder Misserfolg anzeigt.

Die Funktion weist die folgende Syntax auf.

Syntax

```
cron.unschedule (job_id);

cron.unschedule (job_name);
```

Parameter

Parameter	Beschreibung
`job_id`	Ein Aufgabenbezeichner, der von der `cron.schedule` Funktion zurückgegeben wurde, als die Cron-Aufgabe geplant wurde.
`job_name`	Der Name einer Cron-Aufgabe, die mit der `cron.schedule` Funktion geplant wurde.

Beispiele

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)

postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

Tabellen zum Planen von Jobs und zur Erfassung des Status

Die folgenden Tabellen werden verwendet, um die Cron-Aufgaben zu planen und aufzuzeichnen, wie die Aufgaben abgeschlossen wurden.

Tabelle	Beschreibung
`cron.job`	Enthält die Metadaten zu jeder geplanten Aufgabe. Die meisten Interaktionen mit dieser Tabelle sollten über die Funktionen `cron.schedule` und `cron.unschedule` erfolgen.

Tabelle	Beschreibung
	Important Wir empfehlen, dieser Tabelle keine Aktualisierungs- oder Einfügeberechtigungen zu gewähren. Dies würde es dem Benutzer ermöglichen, die `username`-Spalte zu aktualisieren, die als `rds-superuser` ausgeführt wird.
`cron.job_run_details`	Enthält Verlaufsdaten zu vergangenen geplanten Aufträgen , die ausgeführt wurden. Dies ist nützlich, um den Status, die Rückgabe von Nachrichten sowie die Start- und Endzeit des ausgeführten Auftrags zu untersuchen. **Note** Um zu verhindern, dass diese Tabelle ins Unendliche wächst, bereinigen Sie sie in regelmäßigen Abständen. Ein Beispiel finden Sie unter Löschen der Verlaufstabelle pg_cron.

Verwenden von pgAudit zur Protokollierung der Datenbankaktivität

Finanzinstitute, Behörden und viele Branchen müssen Audit-Protokolle aufbewahren, um die gesetzlichen Bestimmungen zu erfüllen. Durch die Verwendung der PostgreSQL-Audit-Erweiterung (pgAudit) mit Ihrem DB-Cluster von Aurora PostgreSQL können Sie die detaillierten Datensätze erfassen, die normalerweise von Prüfern oder zur Erfüllung gesetzlicher Bestimmungen benötigt werden. Sie können beispielsweise die pgAudit-Erweiterung einrichten, um Änderungen an bestimmten Datenbanken und Tabellen nachzuverfolgen, den Benutzer zu erfassen, der die Änderung vorgenommen hat, und viele andere Details.

Die pgAudit-Erweiterung baut auf der Funktionalität der nativen PostgreSQL-Protokollierungsinfrastruktur auf, indem sie die Protokollmeldungen um Details erweitert. Mit anderen Worten, Sie verwenden für die Anzeige Ihres Audit-Protokolls den gleichen Ansatz wie für die Anzeige von Protokollmeldungen. Weitere Informationen zur PostgreSQL-Protokollierung finden Sie unter Datenbank-Protokolldateien von Aurora PostgreSQL.

Die pgAudit-Erweiterung redigiert sensible Daten wie Klartext-Passwörter aus den Protokollen. Wenn Ihr DB-Cluster von Aurora PostgreSQL so konfiguriert ist, dass Data Manipulation Language (DML)-Anweisungen protokolliert werden, wie in Aktivieren der Abfrageprotokollierung für Ihren DB-Cluster von Aurora PostgreSQL beschrieben, können Sie das Klartext-Passwortproblem mithilfe der PostgreSQL-Audit-Erweiterung vermeiden.

Sie können das Auditing für Ihre Datenbank-Instances mit einem hohen Grad an Spezifität konfigurieren. Sie können alle Datenbanken und alle Benutzer überprüfen. Sie können auch festlegen, dass nur bestimmte Datenbanken, Benutzer und andere Objekte überprüft werden. Bestimmte Benutzer und Datenbanken können Sie auch explizit von der Prüfung ausschließen. Weitere Informationen finden Sie unter Benutzer oder Datenbanken von der Audit-Protokollierung ausschließen.

Angesichts der Menge an Details, die erfasst werden können, empfehlen wir, dass Sie bei Verwendung von pgAudit Ihren Speicherverbrauch überwachen.

Die pgAudit-Erweiterung wird von allen verfügbaren Aurora-PostgreSQL-Versionen unterstützt. Eine Liste der pgAudit-Versionen, die von Aurora PostgreSQL unterstützt werden, finden Sie unter Versionen der Erweiterungen für Amazon Aurora PostgreSQL in den Versionshinweisen für Aurora PostgreSQL.

Themen

- Einrichten der pgAudit-Erweiterung
- Überprüfen von Datenbankobjekten
- Benutzer oder Datenbanken von der Audit-Protokollierung ausschließen
- Referenz für die pgAudit-Erweiterung

Einrichten der pgAudit-Erweiterung

Wenn Sie die pgAudit-Erweiterung auf einrichten möchten, fügen Sie zunächst pgAudit zu den gemeinsam genutzten Bibliotheken in der , benutzerdefinierten DB-Cluster-Parametergruppe für Ihren DB-Cluster von Aurora PostgreSQL hinzu. Weitere Informationen über das Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe finden Sie unter Parametergruppen für Amazon Aurora. Als Nächstes installieren Sie die pgAudit-Erweiterung. Abschließend geben Sie die Datenbanken und Objekte an, die Sie überprüfen möchten. Die Schritte in diesem Abschnitt veranschaulichen die Vorgehensweise. Sie können die AWS-Managementkonsole oder die AWS CLI verwenden.

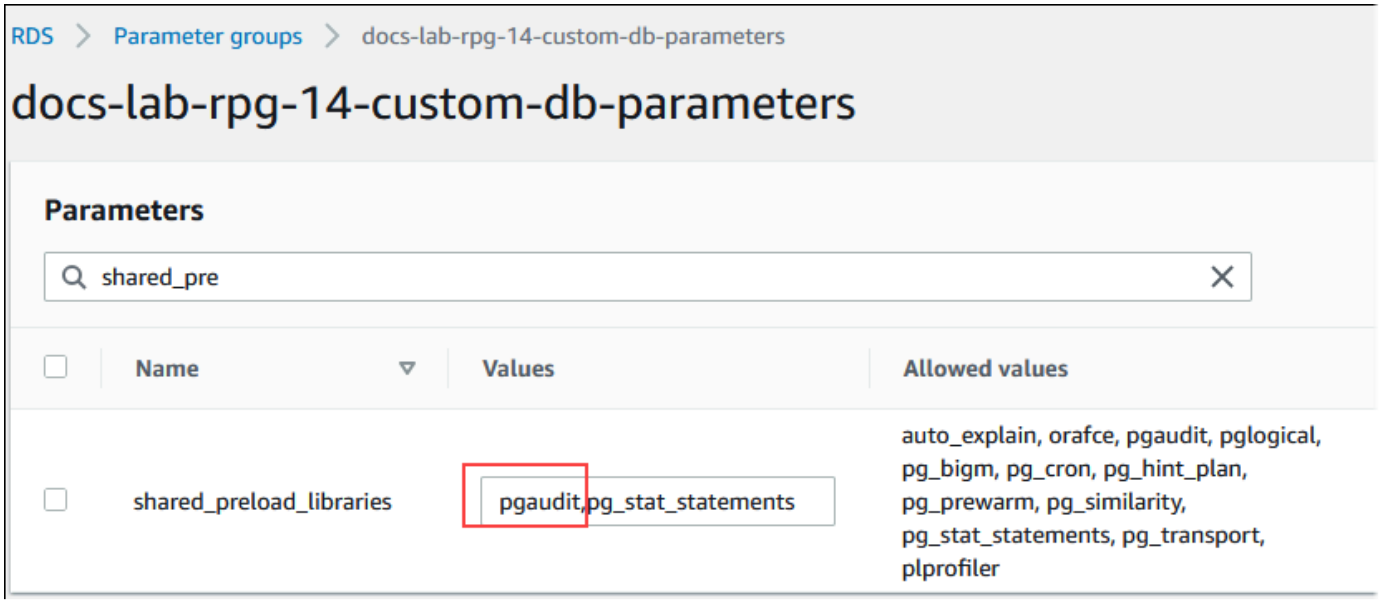
Sie müssen über Berechtigungen als `rds_superuser`-Rolle verfügen, um alle diese Aufgaben ausführen zu können.

Bei den folgenden Schritten wird davon ausgegangen, dass Ihr DB-Cluster von Aurora PostgreSQL einer benutzerdefinierten DB-Cluster -Parametergruppe zugeordnet ist.

Konsole

So richten Sie die pgAudit-Erweiterung ein

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL aus.
3. Öffnen Sie die Registerkarte Configuration (Konfiguration) für Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL. Suchen Sie in den Instance-Details den Link Parameter group (Parametergruppe).
4. Wählen Sie den Link aus, um die benutzerdefinierten Parameter zu öffnen, die Ihrem DB-Cluster von Aurora PostgreSQL zugeordnet sind.
5. Geben Sie in das Suchfeld Parameters (Parameter) `shared_pre` ein, um den `shared_preload_libraries`-Parameter zu finden.
6. Wählen Sie Edit parameters (Parameter bearbeiten) aus, um auf die Eigenschaftswerte zuzugreifen.
7. Fügen Sie `pgaudit` der Liste im Feld Values (Werte) hinzu. Verwenden Sie ein Komma, um Elemente in der Werteliste zu trennen.



8. Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu, damit Ihre Änderung des `shared_preload_libraries`-Parameters wirksam wird.
9. Wenn die Instance verfügbar ist, stellen Sie sicher, dass pgAudit initialisiert wurde. Stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her und führen Sie den folgenden Befehl aus.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. Wenn pgAudit initialisiert ist, können Sie jetzt die Erweiterung erstellen. Sie müssen die Erweiterung nach dem Initialisieren der Bibliothek erstellen, da die `pgaudit`-Erweiterung Ereignisauslöser für die Überwachung von Data Definition Language (DDL)-Anweisungen installiert.

```
CREATE EXTENSION pgaudit;
```

11. Schließen Sie die `psql`-Sitzung.

```
labdb=> \q
```

12. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

13. Suchen Sie den `pgaudit.log`-Parameter in der Liste und legen Sie den entsprechenden Wert für Ihren Anwendungsfall fest. Wenn Sie beispielsweise den `pgaudit.log`-Parameter auf `write` festlegen, wie in der folgenden Abbildung gezeigt, werden Einfügungen, Aktualisierungen, Löschungen und einige andere Typänderungen im Protokoll erfasst.

Sie können auch einen der folgenden Werte für den `pgaudit.log`-Parameter auswählen.

- „none“: Dies ist der Standardwert. Es werden keine Datenbankänderungen protokolliert.
- „all“: Es wird alles protokolliert (Lesen, Schreiben, Funktion, Rolle, DDL, Verschiedenes).
- „ddl“: Protokolliert alle Data Definition Language (DDL)-Anweisungen, die nicht in der `ROLE`-Klasse enthalten sind.
- „function“: Protokolliert Funktionsaufrufe und `DO`-Blöcke.
- „misc“: Protokolliert verschiedene Befehle wie `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM` und `SET`.
- „read“: Protokolliert `SELECT` und `COPY`, wenn die Quelle eine Beziehung (z. B. eine Tabelle) oder eine Abfrage ist.
- „role“: Protokolliert Anweisungen in Bezug auf Rollen und Berechtigungen wie `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE` und `DROP ROLE`.
- „write“: Protokolliert `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE` und `COPY`, wenn das Ziel eine Beziehung (Tabelle) ist.

14. Wählen Sie Änderungen speichern aus.
15. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

16. Wählen Sie Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL aus der Datenbankliste aus.

AWS CLI

So richten Sie pgAudit ein

Wenn Sie pgAudit mit der AWS CLI einrichten möchten, rufen Sie die Operation modify-db-parameter-group auf, um die Audit-Protokollparameter in Ihrer benutzerdefinierten Parametergruppe zu ändern, wie in den folgenden Schritten gezeigt.

1. Verwenden Sie den folgenden AWS CLI-Befehl, um dem `shared_preload_libraries`-Parameter `pgaudit` hinzuzufügen.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Verwenden Sie den folgendenAWS CLI Befehl, um die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu zu starten, sodass die pgaudit-Bibliothek initialisiert wird.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Wenn die Instance verfügbar ist, können Sie überprüfen, ob `pgaudit` initialisiert wurde. Stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her und führen Sie den folgenden Befehl aus.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

   Wenn pgAudit initialisiert ist, können Sie jetzt die Erweiterung erstellen.

```
CREATE EXTENSION pgaudit;
```

4. Schließen Sie die `psql`-Sitzung, damit Sie die AWS CLI verwenden können.

```
labdb=> \q
```

5. Verwenden Sie den folgenden AWS CLI-Befehl, um die Anweisungsklassen anzugeben, die von der Sitzungsüberwachungsprotokollierung erfasst werden sollen. Im Beispiel wird der `pgaudit.log`-Parameter auf `write` festgelegt, wodurch Einfügungen, Aktualisierungen und Löschungen im Protokoll erfasst werden.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

Sie können auch einen der folgenden Werte für den `pgaudit.log`-Parameter auswählen.

- „none“: Dies ist der Standardwert. Es werden keine Datenbankänderungen protokolliert.
- „all“: Es wird alles protokolliert (Lesen, Schreiben, Funktion, Rolle, DDL, Verschiedenes).
- „ddl“: Protokolliert alle Data Definition Language (DDL)-Anweisungen, die nicht in der `ROLE`-Klasse enthalten sind.
- „function“: Protokolliert Funktionsaufrufe und `DO`-Blöcke.
- „misc“: Protokolliert verschiedene Befehle wie `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM` und `SET`.
- „read“: Protokolliert `SELECT` und `COPY`, wenn die Quelle eine Beziehung (z. B. eine Tabelle) oder eine Abfrage ist.
- „role“: Protokolliert Anweisungen in Bezug auf Rollen und Berechtigungen wie `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE` und `DROP ROLE`.
- „write“: Protokolliert `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE` und `COPY`, wenn das Ziel eine Beziehung (Tabelle) ist.

Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL mit dem folgenden AWS CLI-Befehl neu.

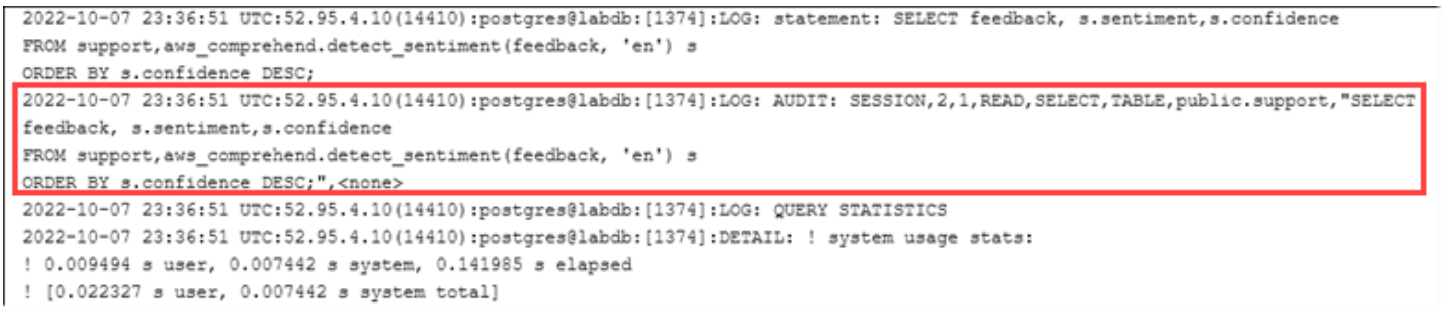
```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

Überprüfen von Datenbankobjekten

Wenn pgAudit auf Ihrem DB-Cluster von Aurora PostgreSQL eingerichtet und für Ihre Anforderungen konfiguriert ist, werden detailliertere Informationen im PostgreSQL-Protokoll erfasst. Während die PostgreSQL-Standardprotokollierungskonfiguration beispielsweise das Datum und die Uhrzeit angibt, zu der eine Änderung in einer Datenbanktabelle vorgenommen wurde, kann der Protokolleintrag mit der pgAudit-Erweiterung das Schema, den Benutzer, der die Änderung vorgenommen hat, und andere Details enthalten, je nachdem, wie die Erweiterungsparameter konfiguriert sind. Sie können das Auditing einrichten, um Änderungen wie folgt zu verfolgen.

- Für jede Sitzung, nach Benutzer. Auf der Sitzungsebene können Sie den vollständig qualifizierten Befehlstext erfassen.
- Für jedes Objekt, nach Benutzer und nach Datenbank.

Die Objektüberwachungsfunktion wird aktiviert, wenn Sie die `rds_pgaudit`-Rolle in Ihrem System erstellen und diese Rolle dann dem `pgaudit.role`-Parameter in Ihrer benutzerdefinierten Parametergruppe hinzufügen. Standardmäßig ist der `pgaudit.role`-Parameter nicht festgelegt und der einzig zulässige Wert ist `rds_pgaudit`. Bei den folgenden Schritten wird davon ausgegangen, dass `pgaudit` initialisiert wurde und Sie die `pgaudit`-Erweiterung gemäß den Schritten unter Einrichten der pgAudit-Erweiterung erstellt haben.



```
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: statement: SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: AUDIT: SESSION,2,1,READ,SELECT,TABLE,public.support,"SELECT
feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;",<none>
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: QUERY STATISTICS
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:DETAIL: ! system usage stats:
! 0.009494 s user, 0.007442 s system, 0.141985 s elapsed
! [0.022327 s user, 0.007442 s system total]
```

Wie in diesem Beispiel gezeigt, enthält die Zeile „LOG: AUDIT: SESSION“ unter anderem Informationen über die Tabelle und deren Schema.

So richten Sie die Objektüberwachung ein

1. Stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her.

```
psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
username=postgrespostgres --password --dbname=labdb
```

2. Erstellen Sie mithilfe des folgenden Befehls eine Datenbankrolle mit dem Namen `rds_pgaudit`.

```
labdb=> CREATE ROLE rds_pgaudit;
CREATE ROLE
labdb=>
```

3. Schließen Sie die `psql`-Sitzung.

```
labdb=> \q
```

   Verwenden Sie in den nächsten Schritten die AWS CLI, um die Audit-Protokollparameter in Ihrer benutzerdefinierten Parametergruppe zu ändern.

4. Verwenden Sie den folgenden AWS CLI-Befehl, um den `pgaudit.role`-Parameter auf `rds_pgaudit` festzulegen. Standardmäßig ist dieser Parameter leer und der einzig zulässige Wert ist `rds_pgaudit`.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot"
 \
   --region aws-region
```

5. Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL mit dem folgenden AWS CLI-Befehl neu, damit Ihre Änderungen der Parameter wirksam werden.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

6. Führen Sie den folgenden Befehl aus, um zu bestätigen, dass `pgaudit.role` auf `rds_pgaudit` festgelegt ist.

```
SHOW pgaudit.role;
pgaudit.role
------------------
rds_pgaudit
```

Um die pgAudit-Protokollierung zu testen, können Sie mehrere Beispielbefehle ausführen, die Sie überprüfen möchten. Sie könnten beispielsweise die folgenden Befehle ausführen.

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
(0 rows)
```

Die Datenbankprotokolle sollten dann einen Eintrag ähnlich dem folgenden enthalten.

```
...
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

Weitere Informationen zur Anzeige der Protokolle finden Sie unter Überwachen von Amazon Aurora-Protokolldateien.

Weitere Informationen zur pgAudit-Erweiterung finden Sie unter pgAudit auf GitHub.

Benutzer oder Datenbanken von der Audit-Protokollierung ausschließen

Wie unter Datenbank-Protokolldateien von Aurora PostgreSQL beschrieben, verbrauchen PostgreSQL-Protokolle Speicherplatz. Die Verwendung der pgAudit-Erweiterung erhöht die in Ihren Protokollen gesammelte Datenmenge je nach den von Ihnen verfolgten Änderungen in unterschiedlichem Maße. Möglicherweise müssen Sie nicht jeden Benutzer oder jede Datenbank in Ihrem DB-Cluster von Aurora PostgreSQL überwachen.

Sie können Benutzer und Datenbanken von der Prüfung ausschließen, um die Auswirkungen auf Ihren Speicher zu minimieren und die unnötige Erfassung von Audit-Datensätzen zu vermeiden. Sie können die Protokollierung auch innerhalb einer bestimmten Sitzung ändern. In den nachstehenden Beispielen wird die Vorgehensweise dazu veranschaulicht.

> **Note**
>
> Parametereinstellungen auf Sitzungsebene haben Vorrang vor den Einstellungen in der benutzerdefinierten DB-Cluster-Parametergruppe der Writer-Instance des DB-Clusters von Aurora PostgreSQL. Wenn Sie nicht möchten, dass Datenbankbenutzer Ihre Konfigurationseinstellungen für die Audit-Protokollierung umgehen, müssen Sie ihre Berechtigungen ändern.

Angenommen, Ihr DB-Cluster von Aurora PostgreSQL ist so konfiguriert, dass derselbe Aktivitätsgrad für alle Benutzer und Datenbanken überprüft wird. Sie entscheiden dann, dass Sie den Benutzer `myuser` nicht überprüfen möchten. Sie können das Auditing für `myuser` mit dem folgenden SQL-Befehl deaktivieren.

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

Anschließend können Sie die folgende Abfrage verwenden, um die Spalte `user_specific_settings` auf `pgaudit.log` zu überprüfen und zu bestätigen, dass der Parameter auf `NONE` festgelegt ist.

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
    usename = 'myuser';
```

Die Ausgabe sollte folgendermaßen aussehen.

```
 user_name | user_specific_settings
-----------+-----------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

Sie können die Protokollierung für einen bestimmten Benutzer während seiner Datenbanksitzung mit dem folgenden Befehl deaktivieren.

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

Verwenden Sie die folgende Abfrage, um die Einstellungsspalte für eine bestimmte Benutzer- und Datenbankkombination auf pgaudit.log zu überprüfen.

```
SELECT
    usename AS "user_name",
    datname AS "database_name",
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

Die Ausgabe entspricht weitgehend der folgenden.

```
  user_name | database_name |     settings
-----------+---------------+------------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

Nachdem Sie das Auditing für `myuser` deaktiviert haben, entscheiden Sie, dass Sie Änderungen an `mydatabase` nicht verfolgen möchten. Sie können das Auditing für diese spezifische Datenbank mit dem folgenden Befehl deaktivieren.

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

Verwenden Sie dann die folgende Abfrage, um die Spalte database_specific_settings zu überprüfen und zu bestätigen, dass pgaudit.log auf NONE festgelegt ist.

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
```

```
WHERE
a.datname = 'mydatabase';
```

Die Ausgabe sollte folgendermaßen aussehen.

```
 database_name | database_specific_settings
---------------+----------------------------
 mydatabase    | {pgaudit.log=NONE}
(1 row)
```

Verwenden Sie den folgenden Befehl, um die Einstellungen wieder auf die Standardeinstellung für myuser festzulegen:

```
ALTER USER myuser RESET pgaudit.log;
```

Verwenden Sie den folgenden Befehl, um die Einstellungen wieder auf die Standardeinstellung für eine Datenbank festzulegen.

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

Verwenden Sie den folgenden Befehl, um Benutzer und Datenbank wieder auf die Standardeinstellung festzulegen.

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

Sie können auch bestimmte Ereignisse im Protokoll erfassen, indem Sie `pgaudit.log` auf einen der anderen zulässigen Werte für den `pgaudit.log`-Parameter festlegen. Weitere Informationen finden Sie unter Liste der zulässigen Einstellungen für den `pgaudit.log`-Parameter.

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

Referenz für die pgAudit-Erweiterung

Sie können den gewünschten Detaillierungsgrad für Ihr Audit-Protokoll angeben, indem Sie einen oder mehrere der in diesem Abschnitt aufgeführten Parameter ändern.

Steuern des pgAudit-Verhaltens

Sie können die Audit-Protokollierung steuern, indem Sie einen oder mehrere der in der folgenden Tabelle aufgeführten Parameter ändern.

Parameter	Beschreibung
`pgaudit.log`	Gibt die Anweisungsklassen an, die durch die Sitzungs-Audit-Protokollierung erfasst werden. Zulässige Werte sind „ddl“, „function“, „misc“, „read“, „role“, „write“, „none“, „all“. Weitere Informationen finden Sie unter Liste der zulässigen Einstellungen für den `pgaudit.log` -Parameter.
`pgaudit.log_catalog`	Wenn diese Option aktiviert ist (auf 1 festgelegt), werden Anweisungen zum Audit-Trail hinzugefügt, wenn sich alle Beziehungen in einer Anweisung in pg_catalog befinden.
`pgaudit.log_level`	Gibt die Protokollstufe an, die für Protokolleinträge verwendet werden soll. Zulässige Werte: „debug5“, „debug4“, „debug3“, „debug2“, „debug1“, „info“, „notice“, „warning“, „log“
`pgaudit.log_parameter`	Wenn diese Option aktiviert ist (auf 1 festgelegt), werden die mit der Anweisung übergebenen Parameter im Audit-Protokoll erfasst.
`pgaudit.log_relation`	Wenn diese Option aktiviert ist (auf 1 festgelegt), erstellt das Audit-Protokoll für die Sitzung einen separaten Protokolleintrag für jede Beziehung (TABLE, VIEW usw.), auf die in einer SELECT- oder DML-Anweisung verwiesen wird.
`pgaudit.log_statement_once`	Gibt an, ob die Protokollierung den Anweisungstext und die Parameter mit dem ersten Protokolleintrag für eine Kombination aus Anweisung/Unteranweisung oder bei jedem Eintrag enthält.
`pgaudit.role`	Gibt die Hauptrolle an, die für die Objektüberwachungsprotokollierung verwendet werden soll. Der einzig zulässige Eintrag ist `rds_pgaudit` .

Liste der zulässigen Einstellungen für den **`pgaudit.log`**-Parameter

Wert	Beschreibung
none	Dies ist die Standardeinstellung. Es werden keine Datenbankänderungen protokolliert.
all	Protokolliert alles (Lesen, Schreiben, Funktion, Rolle, DDL, Verschied enes).
ddl	Protokolliert alle Data Definition Language (DDL)-Anweisungen, die nicht in der `ROLE`-Klasse enthalten sind.
function	Protokolliert Funktionsaufrufe und `DO`-Blöcke.
misc	Protokolliert verschiedene Befehle wie `DISCARD`, `FETCH`, `CHECKPOINT` , `VACUUM` und `SET`.
read	Protokolliert `SELECT` und `COPY`, wenn die Quelle eine Beziehung (z. B. eine Tabelle) oder eine Abfrage ist.
role	Protokolliert Anweisungen in Bezug auf Rollen und Berechtigungen wie `REVOKE`, `CREATE ROLE`, `ALTER ROLE` und `DROP ROLE`.
write	Protokolliert `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE` und `COPY`, wenn das Ziel eine Beziehung (Tabelle) ist.

Um mehrere Ereignistypen mit der Sitzungsüberwachung zu protokollieren, verwenden Sie eine kommagetrennte Liste. Um alle Ereignistypen zu protokollieren, legen Sie `pgaudit.log` auf `ALL` fest. Starten Sie Ihre DB-Instance neu, um die Änderungen zu übernehmen.

Mit der Objektüberwachung können Sie die Überwachungsprotokollierung verfeinern, um mit bestimmten Beziehungen zu arbeiten. Sie können z. B. angeben, dass Sie eine Audit-Protokollierung für `READ`-Vorgänge in einer oder mehreren Tabellen wünschen.

Verwenden von pglogical, um Daten zwischen Instances zu synchronisieren

Alle derzeit verfügbaren Aurora-PostgreSQL-Versionen unterstützen die `pglogical`-Erweiterung. Die Erweiterung pglogical ist älter als die funktionell ähnliche logische Replikationsfunktion, die von

PostgreSQL in Version 10 eingeführt wurde. Weitere Informationen finden Sie unter Überblick über die logische Replikation von PostgreSQL mit Aurora.

Die `pglogical`-Erweiterung unterstützt die logische Replikation zwischen zwei oder mehr DB-Cluster von Aurora PostgreSQL. Es unterstützt auch die Replikation zwischen verschiedenen PostgreSQL-Versionen und zwischen Datenbanken, die auf DB-Instances von RDS für PostgreSQL und DB-Clustern von Aurora PostgreSQL laufen. Die `pglogical`-Erweiterung verwendet ein Publish-Subscribe-Modell, um Änderungen an Tabellen und anderen Objekten, z. B. Sequenzen, von einem Herausgeber auf einen Abonnenten zu replizieren. Sie stützt sich auf einen Replikationsslot, um sicherzustellen, dass Änderungen von einem Herausgeberknoten zu einem Abonnentenknoten synchronisiert werden. Dies ist wie folgt definiert.

- Der Herausgeberknoten ist der DB-Cluster von Aurora PostgreSQL, der die Datenquelle ist, die auf andere Knoten repliziert werden soll. Der Herausgeberknoten definiert die Tabellen, die in einem Veröffentlichungssatz repliziert werden sollen.
- Der Abonnentenknoten ist der DB-Cluster von Aurora PostgreSQL, der WAL-Updates vom Herausgeber erhält. Der Abonnent erstellt ein Abonnement, um eine Verbindung zum Herausgeber herzustellen und die dekodierten WAL-Daten abzurufen. Wenn der Abonnent das Abonnement erstellt, wird der Replikationsslot auf dem Herausgeberknoten erstellt.

Im Folgenden erfahren Sie, wie Sie die `pglogical`-Erweiterung einrichten.

Themen

- Anforderungen und Einschränkungen für die pglogical-Erweiterung
- Einrichten der pglogical-Erweiterung
- Einrichten der logischen Replikation für den DB-Cluster von Aurora PostgreSQL
- Wiederherstellung der logischen Replikation nach einem Hauptversions-Upgrade
- Verwalten logischer Replikationsslots für Aurora PostgreSQL
- Parameterreferenz für die pglogical-Erweiterung

Anforderungen und Einschränkungen für die pglogical-Erweiterung

Alle derzeit verfügbaren Versionen von Aurora PostgreSQL unterstützen die `pglogical`-Erweiterung.

Sowohl der Herausgeberknoten als auch der Abonnentenknoten müssen für die logische Replikation eingerichtet sein.

Die Tabellen, die Sie vom Publisher zum Subscriber replizieren möchten, müssen dieselben Namen und dasselbe Schema haben. Diese Tabellen müssen außerdem dieselben Spalten enthalten und diese müssen dieselben Datentypen verwenden. Sowohl die Herausgeber- als auch die Abonnententabelle müssen dieselben Primärschlüssel haben. Wir empfehlen, nur den PRIMARY KEY als eindeutige Einschränkung zu verwenden.

Die Tabellen auf dem Abonnentenknoten können für CHECK-Einschränkungen und NOT NULL-Einschränkungen großzügigere Einschränkungen haben als die Tabellen auf dem Herausgeberknoten.

Die `pglogical`-Erweiterung bietet Funktionen wie die bidirektionale Replikation, die von der logischen Replikationsfunktion, die in PostgreSQL (Version 10 und höher) integriert ist, nicht unterstützt werden. Weitere Informationen finden Sie unter Die bidirektionale PostgreSQL-Replikation mit pglogical.

Einrichten der pglogical-Erweiterung

Wenn Sie die `pglogical`-Erweiterung auf Ihrem DB-Cluster von Aurora PostgreSQL einrichten möchten, fügen Sie zunächst `pglogical` zu den gemeinsam genutzten Bibliotheken in der benutzerdefinierten DB-Cluster-Parametergruppe für Ihren DB-Cluster von Aurora PostgreSQL hinzu. Sie müssen außerdem den Wert des `rds.logical_replication`-Parameters auf 1 festlegen, um die logische Dekodierung zu aktivieren. Abschließend erstellen Sie die Erweiterung in der Datenbank. Für diese Aufgabe können Sie die AWS-Managementkonsole oder die AWS CLI verwenden.

Sie müssen über Berechtigungen als `rds_superuser`-Rolle verfügen, um diese Aufgaben ausführen zu können.

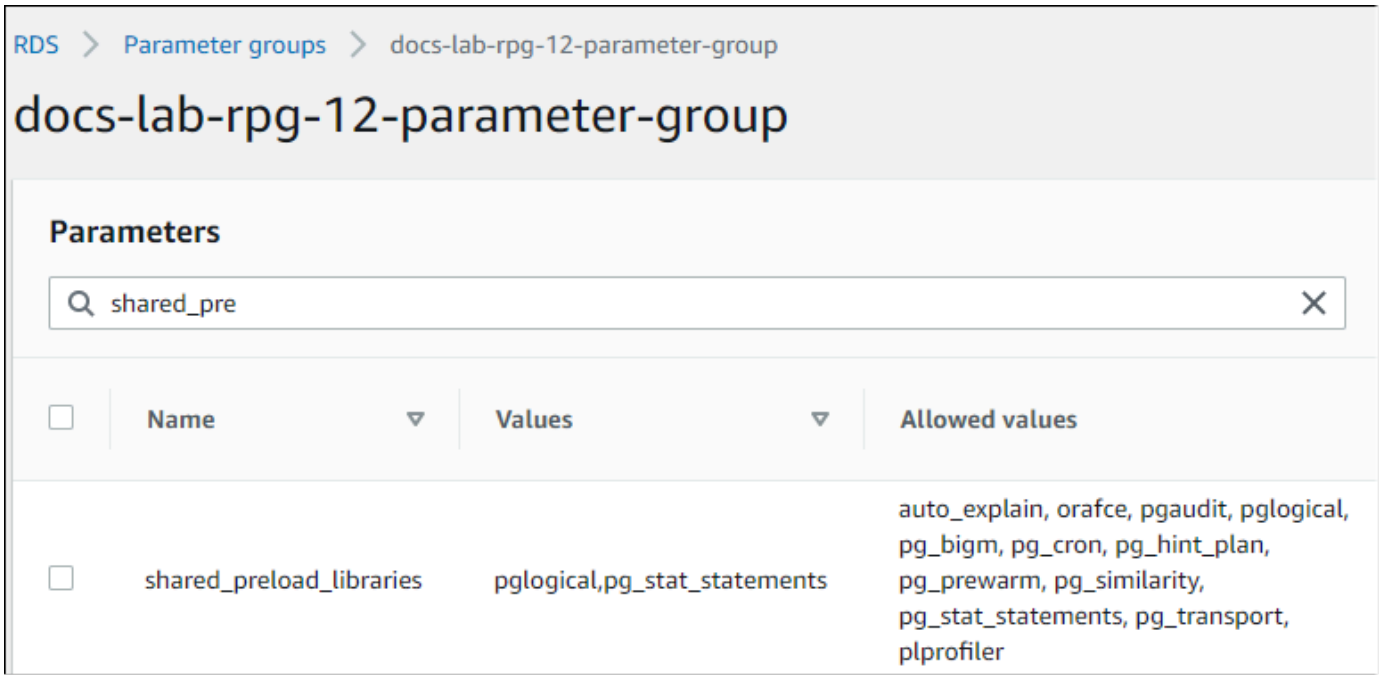
Bei den folgenden Schritten wird davon ausgegangen, dass Ihr DB-Cluster von Aurora PostgreSQL einer benutzerdefinierten DB-Cluster -Parametergruppe zugeordnet ist. Weitere Informationen über das Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe finden Sie unter Parametergruppen für Amazon Aurora.

Konsole

So richten Sie die pglogical-Erweiterung ein

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL aus.
3. Öffnen Sie die Registerkarte Configuration (Konfiguration) für Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL. Suchen Sie in den Instance-Details den Link Parameter group (Parametergruppe).
4. Wählen Sie den Link aus, um die benutzerdefinierten Parameter zu öffnen, die Ihrem DB-Cluster von Aurora PostgreSQL zugeordnet sind.
5. Geben Sie in das Suchfeld Parameters (Parameter) `shared_pre` ein, um den `shared_preload_libraries`-Parameter zu finden.
6. Wählen Sie Edit parameters (Parameter bearbeiten) aus, um auf die Eigenschaftswerte zuzugreifen.
7. Fügen Sie `pglogical` der Liste im Feld Values (Werte) hinzu. Verwenden Sie ein Komma, um Elemente in der Werteliste zu trennen.



8. Suchen Sie den `rds.logical_replication`-Parameter und legen Sie ihn auf 1 fest, um die logische Replikation zu aktivieren.
9. Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu, damit Ihre Änderungen wirksam werden.
10. Wenn die Instance verfügbar ist, können Sie über `psql` (oder pgAdmin) eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL herstellen.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. Führen Sie den folgenden Befehl aus, um zu überprüfen, dass pglogical initialisiert ist.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. Überprüfen Sie die Einstellung, die die logische Dekodierung ermöglicht, wie folgt.

```
SHOW wal_level;
wal_level
-----------
 logical
(1 row)
```

13. Erstellen Sie die Erweiterung wie folgt.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. Wählen Sie Änderungen speichern aus.
15. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
16. Wählen Sie aus der Liste der Datenbanken Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL und dann im Menü „Actions“ (Aktionen) die Option Reboot (Neustart) aus.

AWS CLI

So richten Sie die pglogical-Erweiterung ein

Wenn Sie pglogical mit der AWS CLI einrichten möchten, rufen Sie die Operation modify-db-parameter-group auf, um bestimmte Parameter in Ihrer benutzerdefinierten Parametergruppe zu ändern, wie in den folgenden Schritten gezeigt.

1. Verwenden Sie den folgenden AWS CLI-Befehl, um dem `shared_preload_libraries`-Parameter `pglogical` hinzuzufügen.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Verwenden Sie den folgenden AWS CLI-Befehl, um `rds.logical_replication` auf `1` festzulegen und die logische Dekodierungsfunktion für die Writer-Instance des DB-Clusters von Aurora PostgreSQL zu aktivieren.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. Verwenden Sie den folgenden AWS CLI-Befehl, um die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu zu starten, sodass die pglogical-Bibliothek initialisiert wird.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

4. Wenn die Instance verfügbar ist, stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. Erstellen Sie die Erweiterung wie folgt.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL mit dem folgenden AWS CLI-Befehl neu.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
```

```
    --region aws-region
```

Einrichten der logischen Replikation für den DB-Cluster von Aurora PostgreSQL

Das folgende Verfahren zeigt Ihnen, wie Sie die logische Replikation zwischen zwei DB-Clustern von Aurora PostgreSQL starten. Bei den Schritten wird davon ausgegangen, dass sowohl für die Quelle (Herausgeber) als auch für das Ziel (Abonnent) die `pglogical`-Erweiterung eingerichtet wurde, wie unter Einrichten der pglogical-Erweiterung beschrieben.

> **Note**
>
> Der `node_name` eines Subscriber-Knoten kann nicht mit `rds` beginnen.

So erstellen Sie den Herausgeberknoten und die zu replizierenden Tabellen

Bei diesen Schritten wird vorausgesetzt, dass Ihr DB-Cluster von Aurora PostgreSQL über eine Writer-Instance mit einer Datenbank mit einer oder mehreren Tabellen verfügt, die Sie auf einen anderen Knoten replizieren möchten. Sie müssen die Tabellenstruktur des Herausgebers für den Abonnenten neu erstellen. Rufen Sie daher bei Bedarf zunächst die Tabellenstruktur ab. Verwenden Sie dazu den `psql`-Metabefehl `\d` *`tablename`* und erstellen Sie dann dieselbe Tabelle auf der Abonnenten-Instance. Mit dem folgenden Verfahren wird zu Demonstrationszwecken eine Beispieltabelle für den Herausgeber (Quelle) erstellt.

1. Verwenden Sie `psql`, um eine Verbindung zu der Instance herzustellen, die die Tabelle enthält, die Sie als Quelle für Abonnenten verwenden möchten.

   ```
   psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password --dbname=labdb
   ```

   Wenn Sie noch keine vorhandene Tabelle haben, die Sie replizieren möchten, können Sie wie folgt eine Beispieltabelle erstellen.

   a. Erstellen Sie mit der folgenden SQL-Anweisung eine Beispieltabelle.

      ```
      CREATE TABLE docs_lab_table (a int PRIMARY KEY);
      ```

   b. Füllen Sie die Tabelle mit der folgenden SQL-Anweisung mit generierten Daten auf.

```
INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
INSERT 0 5000
```

c. Stellen Sie sicher, dass Daten in der Tabelle vorhanden sind, indem Sie die folgende SQL-Anweisung verwenden.

```
SELECT count(*) FROM docs_lab_table;
```

2. Identifizieren Sie diesen DB-Cluster von Aurora PostgreSQL wie folgt als Herausgeberknoten.

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_provider',
    dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 dbname=labdb');
 create_node
-------------
   3410995529
(1 row)
```

3. Fügen Sie die Tabelle, die Sie replizieren möchten, zum Standardreplikationssatz hinzu. Weitere Informationen zu Replikationssätzen finden Sie unter Replikationssätze in der Dokumentation zur pglogical-Erweiterung.

```
SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
 NULL, NULL);
 replication_set_add_table
  ---------------------------
  t
  (1 row)
```

Die Einrichtung des Herausgeberknotens ist abgeschlossen. Sie können jetzt den Abonnentenknoten einrichten, um die Updates vom Herausgeber zu erhalten.

So richten Sie den Abonnentenknoten ein und erstellen ein Abonnement für den Empfang von Updates

Bei diesen Schritten wird vorausgesetzt, dass der DB-Cluster von Aurora PostgreSQL mit der `pglogical`-Erweiterung eingerichtet wurde. Weitere Informationen finden Sie unter Einrichten der pglogical-Erweiterung.

1. Verwenden Sie `psql`, um sich mit der Instance zu verbinden, die Updates vom Herausgeber erhalten soll.

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. Erstellen Sie auf dem DB-Cluster von Aurora PostgreSQL des Abonnenten dieselbe Tabelle, die auf dem Herausgeber vorhanden ist. In diesem Beispiel heißt die Tabelle `docs_lab_table`. Sie können die Tabelle wie folgt erstellen.

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. Stellen Sie sicher, dass diese Tabelle leer ist.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. Identifizieren Sie diesen DB-Cluster von Aurora PostgreSQL wie folgt als Abonnentenknoten.

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
(1 row)
```

5. Erstellen Sie das Abonnement.

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
   replication_sets := ARRAY['default'],
   synchronize_data := true,
   forward_origins := '{}' );
 create_subscription
---------------------
```

```
1038357190
(1 row)
```

Wenn Sie diesen Schritt abschließen, werden die Daten aus der Tabelle des Herausgeberknotens in der Tabelle auf dem Abonnentenknoten erstellt. Mit der folgenden SQL-Abfrage können Sie überprüfen, ob dies der Fall ist.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

Ab diesem Zeitpunkt werden Änderungen, die an der Tabelle auf dem Herausgeberknoten vorgenommen wurden, auf die Tabelle auf dem Abonnentenknoten repliziert.

Wiederherstellung der logischen Replikation nach einem Hauptversions-Upgrade

Bevor Sie ein Hauptversions-Upgrade eines DB-Clusters von Aurora PostgreSQL durchführen können, der als Herausgeberknoten für die logische Replikation eingerichtet ist, müssen Sie alle Replikationsslots einschließlich der Slots löschen, die nicht aktiv sind. Wir empfehlen, Datenbanktransaktionen vorübergehend vom Herausgeberknoten umzuleiten, die Replikationsslots zu löschen, den DB-Cluster von Aurora PostgreSQL zu aktualisieren und dann die Replikation erneut einzurichten und neu zu starten.

Die Replikationsslots werden nur auf dem Herausgeberknoten gehostet. Der Aurora-PostgreSQL-Abonnentenknoten hat in einem logischen Replikationsszenario keine Slots, die gelöscht werden könnten. Der Upgrade-Prozess für die Hauptversion von Aurora PostgreSQL unterstützt das Upgrade des Abonnenten auf eine neue Hauptversion von PostgreSQL unabhängig vom Herausgeberknoten. Der Upgrade-Prozess unterbricht jedoch den Replikationsprozess und beeinträchtigt die Synchronisation der WAL-Daten zwischen dem Herausgeber- und dem Abonnentenknoten. Sie müssen die logische Replikation zwischen Herausgeber und Abonnent wiederherstellen, nachdem Sie den Herausgeber, den Abonnenten oder beide aktualisiert haben. Im folgenden Verfahren wird gezeigt, wie Sie feststellen, ob die Replikation unterbrochen wurde, und wie Sie das Problem beheben.

Feststellen, dass die logische Replikation unterbrochen wurde

Sie können feststellen, ob der Replikationsprozess unterbrochen wurde, indem Sie entweder den Herausgeberknoten oder den Abonnentenknoten wie folgt abfragen.

So überprüfen Sie den Herausgeberknoten

- Verwenden Sie `psql`, um eine Verbindung mit dem Herausgeberknoten herzustellen, und fragen Sie dann die `pg_replication_slots`-Funktion ab. Notieren Sie sich den Wert in der aktiven Spalte. Normalerweise wird `t` (true) zurückgegeben, was bedeutet, dass die Replikation aktiv ist. Wenn die Abfrage `f` (false) zurückgibt, ist dies ein Hinweis darauf, dass die Replikation an den Abonnenten gestoppt wurde.

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

So überprüfen Sie den Abonnentenknoten

Auf dem Abonnentenknoten können Sie den Status der Replikation auf drei verschiedene Arten überprüfen.

- Suchen Sie in den PostgreSQL-Protokollen auf dem Abonnentenknoten nach Fehlermeldungen. Das Protokoll identifiziert Fehler mit Meldungen, die den Exit-Code 1 enthalten, wie im Folgenden dargestellt.

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- Fragen Sie die `pg_replication_origin`-Funktion ab. Stellen Sie mithilfe von `psql` eine Verbindung mit der Datenbank auf dem Abonnentenknoten her und fragen Sie die `pg_replication_origin`-Funktion wie folgt ab.

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
```

```
(0 rows)
```

Die leere Ergebnismenge bedeutet, dass die Replikation unterbrochen wurde. Die Ausgabe sollte normalerweise folgendermaßen aussehen.

```
 roident |                      roname
---------+---------------------------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

- Fragen Sie die `pglogical.show_subscription_status`-Funktion wie im folgenden Beispiel veranschaulicht ab.

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
-------+----------------+--------+-----------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

Diese Ausgabe zeigt, dass die Replikation unterbrochen wurde. Ihr Status lautet down. Normalerweise zeigt die Ausgabe den Status `replicating` an.

Wenn Ihr logischer Replikationsprozess unterbrochen wurde, können Sie die Replikation wiederherstellen, indem Sie die folgenden Schritte ausführen.

So stellen Sie die logische Replikation zwischen Herausgeber- und Abonnentenknoten wieder her

Um die Replikation wiederherzustellen, trennen Sie zuerst den Abonnenten vom Herausgeberknoten und richten dann das Abonnement erneut ein, wie in diesen Schritten beschrieben.

1. Stellen Sie mit `psql` wie folgt eine Verbindung zum Abonnentenknoten her.

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. Deaktivieren Sie das Abonnement, indem Sie die `pglogical.alter_subscription_disable`-Funktion verwenden.

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
```

```
----------------------------
 t
(1 row)
```

3. Rufen Sie die ID des Herausgeberknotens ab, indem Sie `pg_replication_origin` wie folgt abfragen.

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+-------------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. Verwenden Sie die Antwort aus dem vorherigen Schritt mit dem `pg_replication_origin_create`-Befehl, um die ID zuzuweisen, die vom Abonnement verwendet werden kann, wenn es erneut eingerichtet wurde.

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
------------------------------
                            1
(1 row)
```

5. Aktivieren Sie das Abonnement, indem Sie seinen Namen mit dem Status `true` übergeben, wie im folgenden Beispiel veranschaulicht.

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
  alter_subscription_enable
---------------------------
 t
(1 row)
```

Prüfen Sie den Status des Knotens. Sein Status sollte `replicating` wie in diesem Beispiel gezeigt lauten.

```
SELECT subscription_name,status,slot_name
  FROM pglogical.show_subscription_status();
            subscription_name |   status    |              slot_name
------------------------------+-------------+-------------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
```

```
(1 row)
```

Überprüfen Sie den Status des Replikationsslots des Abonnenten auf dem Herausgeberknoten. Die `active`-Spalte des Slots sollte den Wert `t` (true) zurückgeben, was darauf hinweist, dass die Replikation wiederhergestellt wurde.

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                   slot_name              |      plugin      | slot_type | active
------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

Verwalten logischer Replikationsslots für Aurora PostgreSQL

Bevor Sie ein Hauptversions-Upgrade eines DB-Clusters von Aurora PostgreSQL durchführen können, der als Herausgeberknoten in einem logischen Replikationsszenario eingerichtet ist, müssen Sie alle Replikationsslots auf der Instance löschen. Bei der Vorabüberprüfung des Hauptversions-Upgrades werden Sie darüber informiert, dass das Upgrade erst fortgesetzt werden kann, wenn die Slots gelöscht wurden.

Um Replikationsslots zu identifizieren, die mit der `pglogical`-Erweiterung erstellt wurden, melden Sie sich bei jeder Datenbank an und rufen Sie die Namen der Knoten ab. Wenn Sie den Abonnentenknoten abfragen, erhalten Sie in der Ausgabe sowohl den Herausgeber- als auch den Abonnentenknoten, wie in diesem Beispiel gezeigt.

```
SELECT * FROM pglogical.node;
node_id   |     node_name
------------+-------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

Die Details zum Abonnement erhalten Sie mit der folgenden Abfrage.

```
SELECT sub_name,sub_slot_name,sub_target
  FROM pglogical.subscription;
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
```

```
(1 row)
```

Sie können das Abonnement jetzt wie folgt kündigen.

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

Nachdem Sie das Abonnement gekündigt haben, können Sie den Knoten löschen.

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
 t
(1 row)
```

Sie können wie folgt überprüfen, dass der Knoten nicht mehr existiert.

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

Parameterreferenz für die pglogical-Erweiterung

In der Tabelle finden Sie Parameter, die der `pglogical`-Erweiterung zugeordnet sind. Parameter wie `pglogical.conflict_log_level` und `pglogical.conflict_resolution` werden verwendet, um Aktualisierungskonflikte zu beheben. Konflikte können auftreten, wenn Änderungen lokal an denselben Tabellen vorgenommen werden, die Änderungen vom Herausgeber abonniert haben. Konflikte können auch in verschiedenen Szenarien auftreten, z. B. bei der bidirektionalen Replikation oder wenn mehrere Abonnenten vom selben Herausgeber replizieren. Weitere Informationen finden Sie unter Die bidirektionale PostgreSQL-Replikation mit pglogical.

Parameter	Beschreibung
pglogical.batch_inserts	Batch-Inserts wenn möglich Standardmäßig nicht festgelegt. In '1' ändern zum Einschalten, in '0' zum Ausschalten.

Parameter	Beschreibung
pglogical.conflict_log_level	Legt die Protokollstufe für die Protokollierung gelöster Konflikte fest. Unterstützte Zeichenfolgenwerte sind debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal, panic.
pglogical.conflict_resolution	Legt die Methode fest, die verwendet werden soll, um Konflikte zu lösen, die lösbar sind. Unterstützte Zeichenfolgenwerte sind error, apply_remote, keep_local, last_update_wins, first_upd ate_wins.
pglogical.extra_connection_ options	Verbindungsoptionen zum Hinzufügen zu allen Peer-Knot enverbindungen
pglogical.synchronous_commit	Spezifischer synchroner Commit-Wert für pglogical
pglogical.use_spi	Verwenden Sie zum Anwenden von Änderungen SPI (Server Programming Interface) anstelle der Low-Level-API. In '1' ändern zum Einschalten, in '0' zum Ausschalten. Weitere Informationen zu SPI finden Sie unter Server Programming Interface in der PostgreSQL-Dokumentation.

Arbeiten mit den unterstützten Fremddaten-Wrapper für Amazon Aurora PostgreSQL

Ein Fremddaten-Wrapper (FDW) ist eine bestimmte Art von Erweiterung, die Zugriff auf externe Daten ermöglicht. Zum Beispiel ermöglicht die Erweiterung `oracle_fdw` Ihrer Aurora-PostgreSQL-DB-Instance die Zusammenarbeit mit Oracle-Datenbanken.

Im Folgenden finden Sie Informationen zu mehreren unterstützten PostgreSQL-Fremddaten-Wrappern.

Themen

- Verwenden der Erweiterung log_fdw für den Zugriff auf das DB-Protokoll mithilfe von SQL
- Verwenden der postgres_fdw-Erweiterung für den Zugriff auf externe Daten
- Arbeiten mit MySQL-Datenbanken mithilfe der Erweiterung mysql_fdw

- Arbeiten mit Oracle-Datenbanken unter Verwendung der Erweiterung oracle_fdw
- Arbeiten mit SQL-Server-Datenbanken unter Verwendung der Erweiterung tds_fdw

Verwenden der Erweiterung log_fdw für den Zugriff auf das DB-Protokoll mithilfe von SQL

Das Aurora-PostgreSQL-DB-Cluster unterstützt die `log_fdw`-Erweiterung, mit der Sie über eine SQL-Schnittstelle auf Ihr Datenbank-Engine-Protokoll zugreifen können. Die `log_fdw`-Erweiterung umfasst zwei neue Funktionen, die das Erstellen von Fremdtabellen für Datenbankprotokolle erleichtern:

- `list_postgres_log_files` führt die Dateien im Datenbankprotokollverzeichnis und die Dateigröße in Bytes auf.
- `create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text)` erstellt eine Fremdtabelle für die angegebene Datei in der aktuellen Datenbank.

Alle durch `log_fdw` erstellten Funktionen gehören `rds_superuser`. Mitglieder der `rds_superuser`-Rolle können anderen Datenbankbenutzern Zugriff auf diese Funktionen gewähren.

Standardmäßig werden die Protokolldateien von Amazon Aurora im `stderr` (Standardfehler)-Format generiert, wie im `log_destination`-Parameter angegeben. Es gibt nur zwei Optionen für diesen Parameter: `stderr` und `csvlog` (Comma Separated Values, CSV). Wenn Sie dem Parameter die Option `csvlog` hinzufügen, generiert Amazon Aurora sowohl `stderr`- als auch `csvlog`-Protokolle. Dies kann die Speicherkapazität Ihres DB-Clusters beeinträchtigen. Daher müssen Sie sich der anderen Parameter bewusst sein, die sich auf die Protokollbehandlung auswirken. Weitere Informationen finden Sie unter Festlegen des Protokollziels (`stderr, csvlog`).

Ein Vorteil der Generierung von `csvlog`-Protokollen ist, dass Sie mit der `log_fdw`-Erweiterung Fremdtabellen erstellen können, bei denen die Daten ordentlich in verschiedene Spalten aufgeteilt sind. Dazu muss Ihre Instance einer benutzerdefinierten DB-Parametergruppe zugeordnet sein, damit Sie die Einstellung für `log_destination` ändern können. Weitere Informationen zur Vorgehensweise finden Sie unter Parametergruppen für Amazon Aurora.

Im folgenden Beispiel wird angenommen, dass der `log_destination`-Parameter `cvslog` enthält.

So verwenden Sie die Erweiterung log_fdw:

1. Installieren Sie die `log_fdw`-Erweiterung.

```
postgres=> CREATE EXTENSION log_fdw;
CREATE EXTENSION
```

2. Erstellen Sie den Protokollserver als Fremddaten-Wrapper.

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. Wählen Sie aus einer Liste an Protokolldateien "alle" aus.

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

   Beispiel-Antwort.

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. Erstellen Sie eine Tabelle mit einer einzigen 'log_entry'-Spalte für die ausgewählte Datei.

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

   Die Antwort liefert keine weiteren Details außer, dass die Tabelle jetzt existiert.

```
-----------------------------------
(1 row)
```

5. Wählen Sie ein Beispiel der Protokolldatei aus. Mit dem folgenden Code werden die Protokollzeit und die Fehlermeldungsbeschreibung abgerufen.

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

Beispiel-Antwort.

```
            log_time             |                                  message
---------------------------------
+-------------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

Verwenden der postgres_fdw-Erweiterung für den Zugriff auf externe Daten

Auf die Daten in einer Tabelle auf einem Remote-Datenbank-Server können Sie mit der Erweiterung postgres_fdw zugreifen. Wenn Sie eine Remote-Verbindung ausgehend von einer PostgreSQL-DB-Instance einrichten, ist der Zugriff auch für das Lesereplikat verfügbar.

Verwenden Sie postgres_fdw wie folgt für den Zugriff auf einen Remote-Datenbank-Server:

1. Installieren Sie die Erweiterung postgres_fdw.

   ```
   CREATE EXTENSION postgres_fdw;
   ```

2. Erstellen Sie einen Fremddaten-Server mit CREATE SERVER.

   ```
   CREATE SERVER foreign_server
   FOREIGN DATA WRAPPER postgres_fdw
   OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
   ```

3. Erstellen Sie ein Benutzer-Mapping, um die Rolle zu identifizieren, die auf dem Remote-Server verwendet werden soll.

> **Important**
>
> Um das Passwort so zu redigieren, dass es nicht in den Protokollen angezeigt wird, legen Sie es auf Sitzungsebene auf `log_statement=none` fest. Durch die Festlegung auf Parameterebene wird das Passwort nicht redigiert.

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. Erstellen Sie eine Tabelle, die der Tabelle auf dem Remote-Server zugewiesen ist.

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

Arbeiten mit MySQL-Datenbanken mithilfe der Erweiterung mysql_fdw

Um von Ihrem Aurora-PostgreSQL-DB-Cluster auf eine MySQL-kompatible Datenbank zuzugreifen, können Sie die `mysql_fdw`-Erweiterung installieren und verwenden. Mit diesem Fremddaten-Wrapper können Sie mit RDS für MySQL, Aurora MySQL, MariaDB und anderen MySQL-kompatiblen Datenbanken arbeiten. Die Verbindung des Aurora-PostgreSQL-DB-Clusters mit der MySQL-Datenbank wird je nach Client- und Serverkonfigurationen bestmöglich verschlüsselt. Sie können die Verschlüsselung jedoch erzwingen, wenn Sie möchten. Weitere Informationen finden Sie unter Verwenden der Verschlüsselung während der Übertragung mit der Erweiterung.

Die `mysql_fdw`-Erweiterung wird in Amazon Aurora PostgreSQL Version 15.4, 14.9, 13.12, 12.16 und neueren Versionen unterstützt. Es werden Auswahlen, Einfügungen, Updates und Löschungen einer RDS-für-PostgreSQL-DB in Tabellen einer MySQL-kompatiblen Datenbank-Instance unterstützt.

Themen

- Einrichten der Aurora-PostgreSQL-DB zur Verwendung der mysql_fdw-Erweiterung
- Beispiel: Arbeiten mit einer Aurora-MySQL--Datenbank von Aurora PostgreSQL

- Verwenden der Verschlüsselung während der Übertragung mit der Erweiterung

Einrichten der Aurora-PostgreSQL-DB zur Verwendung der mysql_fdw-Erweiterung

Das Einrichten der `mysql_fdw`-Erweiterung auf dem Aurora-PostgreSQL-DB-Cluster umfasst das Laden der Erweiterung in den DB-Cluster und das anschließende Erstellen des Verbindungspunkts mit der MySQL-DB-Instance. Für diese Aufgabe benötigen Sie folgende Details zur MySQL-DB-Instance:

- Hostname oder Endpunkt. Sie können den Endpunkt für einen Aurora-MySQL-DB-Cluster mithilfe der Konsole ermitteln. Wählen Sie die Registerkarte „Connectivity & security“ (Konnektivität und Sicherheit) aus und sehen Sie im Abschnitt „Endpoint and port“ (Endpunkt und Port) nach.
- Port-Nummer. Die Standardport-Nummer für MySQL ist 3306.
- Name der Datenbank. Die DB-ID.

Sie müssen auch Zugriff auf die Sicherheitsgruppe oder die Zugriffssteuerungsliste (ACL) für den MySQL-Port 3306 gewähren. Der Aurora-PostgreSQL-DB-Cluster und der Aurora-MySQL-DB-Cluster ebenso wie benötigen Zugriff auf Port 3306. Wenn der Zugriff nicht korrekt konfiguriert ist, wird beim Versuch, eine Verbindung mit einer MySQL-kompatiblen Tabelle herzustellen, eine Fehlermeldung ähnlich wie die folgende angezeigt:

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-
region.rds.amazonaws.com:3306' (110)
```

Im folgenden Verfahren erstellen Sie (als `rds_superuser`-Konto) den fremden Server. Anschließend gewähren Sie bestimmten Benutzern Zugriff auf den fremden Server. Diese Benutzer erstellen dann ihre eigenen Zuordnungen zu den entsprechenden MySQL-Benutzerkonten zur Zusammenarbeit mit der MySQL-DB-Instance.

So greifen Sie mit mysql_fdw auf einen MySQL-Datenbankserver zu

1. Stellen Sie über ein Konto, das die `rds_superuser`-Rolle enthält, eine Verbindung mit Ihrer PostgreSQL-DB-Instance her. Wenn Sie die Standardwerte beim Erstellen des Aurora-PostgreSQL-DB-Clusters akzeptiert haben, lautet der Benutzername `postgres`. Sie können sich mit dem `psql`-Befehlszeilen-Tool wie folgt verbinden:

```
psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. Installieren Sie die `mysql_fdw`-Erweiterung wie folgt:

```
postgres=> CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
```

Nachdem die Erweiterung auf dem Aurora-PostgreSQL-DB-Cluster installiert wurde, richten Sie den fremden Server ein, der die Verbindung mit einer MySQL-Datenbank bereitstellt.

So erstellen Sie den fremden Server

Führen Sie diese Aufgaben auf dem Aurora-PostgreSQL-DB-Cluster aus. Die Schritte setzen voraus, dass Sie als Benutzer mit `rds_superuser`-Berechtigungen wie `postgres` verbunden sind.

1. Erstellen Sie einen fremden Server auf dem Aurora-PostgreSQL-DB-Cluster:

```
postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-
name.111122223333.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Gewähren Sie den entsprechenden Benutzern Zugriff auf den fremden Server. Dies sollten keine Administratorbenutzer sein, d. h. Benutzer ohne `rds_superuser`-Rolle.

```
postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
GRANT
```

PostgreSQL-Benutzer erstellen und verwalten ihre eigenen Verbindungen mit der MySQL-Datenbank über den fremden Server.

Beispiel: Arbeiten mit einer Aurora-MySQL--Datenbank von Aurora PostgreSQL

Angenommen, Sie haben eine einfache Tabelle auf einer Aurora-PostgreSQL-DB-Instance. Ihre Aurora-PostgreSQL-Benutzer möchten (`SELECT`)-, `INSERT`-, `UPDATE`- und `DELETE`-Elemente in dieser Tabelle abfragen. Nehmen wir an, dass die `mysql_fdw`-Erweiterung auf Ihrer RDS-für-PostgreSQL-DB-Instance erstellt wurde, wie im vorherigen Verfahren beschrieben. Nachdem Sie

eine Verbindung mit der RDS-für-PostgreSQL-DB-Instance als Benutzer mit `rds_superuser`-Berechtigungen hergestellt haben, können Sie mit den folgenden Schritten fortfahren.

1. Erstellen Sie einen fremden Server in der Aurora-PostgreSQL-DB-Instance:

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-
DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Gewähren Sie einem Benutzer, der keine `rds_superuser`-Berechtigungen hat, die Nutzung, zum Beispiel `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. Stellen Sie eine Verbindung als *`user1`* (Benutzer 1) her und erstellen Sie dann eine Zuordnung zum MySQL-Benutzer:

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser',
 password 'mypassword');
CREATE USER MAPPING
```

4. Erstellen Sie eine fremde Tabelle, die mit der MySQL-Tabelle verknüpft ist:

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname
 'test', table_name '');
CREATE FOREIGN TABLE
```

5. Führen Sie eine einfache Abfrage mit der fremden Tabelle aus:

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. Sie können der MySQL-Tabelle Daten hinzufügen, ändern und daraus entfernen. Beispiel:

```
test=> INSERT INTO mytab values (2, 'mango');
INSERT 0 1
```

Führen Sie die SELECT-Abfrage noch einmal aus, um die Ergebnisse zu sehen:

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
(2 rows)
```

Verwenden der Verschlüsselung während der Übertragung mit der Erweiterung

Die Verbindung mit MySQL über Aurora PostgreSQL verwendet standardmäßig die Verschlüsselung während der Übertragung (TLS/SSL). Die Verbindung wird jedoch nicht verschlüsselt, wenn sich die Client- und Serverkonfiguration unterscheiden. Sie können die Verschlüsselung für alle ausgehenden Verbindungen erzwingen, indem Sie die Option `REQUIRE SSL` in den RDS-für-MySQL-Benutzerkonten angeben. Derselbe Ansatz funktioniert auch für MariaDB- und Aurora-MySQL-Benutzerkonten.

Für MySQL-Benutzerkonten, die für `REQUIRE SSL` konfiguriert sind, schlägt der Verbindungsversuch fehl, wenn keine sichere Verbindung hergestellt werden kann.

Um die Verschlüsselung für vorhandene MySQL-Datenbankbenutzerkonten durchzusetzen, können Sie den Befehl `ALTER USER` verwenden. Die Syntax variiert je nach MySQL-Version, wie in der folgenden Tabelle gezeigt. Weitere Informationen finden Sie unter ALTER USER im MySQL-Referenzhandbuch.

MySQL 5.7, MySQL 8.0	MySQL 5.6
`ALTER USER 'user'@'%' REQUIRE SSL;`	`GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;`

Weitere Informationen zur Erweiterung `mysql_fdw` finden Sie in der mysql_fdw-Dokumentation.

Arbeiten mit Oracle-Datenbanken unter Verwendung der Erweiterung oracle_fdw

Um von Ihrem Aurora-PostgreSQL-DB-Cluster auf eine Oracle-Datenbank zuzugreifen, können Sie die Erweiterung `oracle_fdw` installieren und verwenden. Diese Erweiterung ist ein Fremddaten-

Wrapper für Oracle-Datenbanken. Weitere Informationen zu dieser Erweiterung finden Sie in der oracle_fdw-Dokumentation.

Die Erweiterung `oracle_fdw` wird von Aurora PostgreSQL 12.7, Amazon Aurora Version 4.2) und höheren Versionen unterstützt.

Themen

- Aktivieren der Erweiterung oracle_fdw
- Beispiel: Verwendung eines fremden Servers, der mit einer Amazon RDS für Oracle Database verknüpft ist
- Datenverschlüsselung während der Übertragung
- Informationen zur Ansicht pg_user_mappings und zu Berechtigungen

Aktivieren der Erweiterung oracle_fdw

So verwenden Sie die Erweiterung oracle_fdw:

Aktivieren der Erweiterung oracle_fdw

- Führen Sie den folgenden Befehl mit einem Konto aus, das die `rds_superuser`-Berechtigungen besitzt.

```
CREATE EXTENSION oracle_fdw;
```

Beispiel: Verwendung eines fremden Servers, der mit einer Amazon RDS für Oracle Database verknüpft ist

Das folgende Beispiel zeigt die Verwendung eines fremden Servers, der mit einer Amazon-RDS-für-Oracle-Datenbank verknüpft ist.

So erstellen Sie einen fremden Server, der mit einer RDS-for-Oracle-Datenbank verknüpft ist:

1. Beachten Sie Folgendes auf der RDS-for-Oracle-DB-Instance:
    - Endpunkt
    - Port
    - Datenbankname

2. Erstellen Sie einen fremden Server.

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. Gewähren Sie einem Benutzer, der keine `rds_superuser`-Berechtigungen hat, die Nutzung, zum Beispiel `user1`.

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. Verbinden Sie sich als `user1` und erstellen Sie ein Mapping zu einem Oracle-Benutzer.

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. Erstellen einer fremden Tabelle, die mit einer Oracle-Tabelle verknüpft ist.

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. Fragen Sie die fremde Tabelle ab.

```
test=>  SELECT * FROM mytab;
a
---
1
(1 row)
```

Wenn die Abfrage den folgenden Fehler meldet, überprüfen Sie Ihre Sicherheitsgruppe und Zugriffssteuerungsliste (Access Control List, ACL), um sicherzustellen, dass beide Instances kommunizieren können.

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

Datenverschlüsselung während der Übertragung

Die PostgreSQL-zu-Oracle-Verschlüsselung bei der Übertragung basiert auf einer Kombination von Client- und Server-Konfigurationsparametern. Ein Beispiel für Oracle 21c finden Sie unter Informationen zu den Werten für Verschlüsselung und Integrität in der Oracle-Dokumentation. Der Client, der für oracle_fdw auf Amazon RDS verwendet wird, ist mit `ACCEPTED` konfiguriert, was bedeutet, dass die Verschlüsselung von der Konfiguration des Oracle-Datenbankservers abhängt und Oracle Security Library (libnnz) als Verschlüsselung verwendet wird.

Wenn sich Ihre Datenbank auf RDS für Oracle befindet, finden Sie weitere Informationen zum Konfigurieren der Verschlüsselung unter Oracle Native Network Encryption.

Informationen zur Ansicht pg_user_mappings und zu Berechtigungen

Der PostgreSQL-Katalog `pg_user_mapping` speichert das Mapping eines Benutzers von Aurora PostgreSQL für den Benutzer auf einem fremden Datenserver (Remote). Der Zugriff auf den Katalog ist eingeschränkt, aber Sie verwenden die Ansicht `pg_user_mappings`, um die Mappings zu sehen. Das folgende Beispiel zeigt, wie Berechtigungen für eine Oracle-Beispieldatenbank gelten. Diese Informationen gelten im Allgemeinen für jeden fremden Daten-Wrapper.

In der folgenden Ausgabe finden Sie Rollen und Berechtigungen, die drei verschiedenen Beispielbenutzern zugeordnet sind. Benutzer `rdssu1` und `rdssu2` sind Mitglieder der `rds_superuser`-Rolle und `user1` nicht. In dem Beispiel wird der `psql`-Metabefehl `\du` verwendet, um vorhandene Rollen aufzulisten.

```
test=>  \du
                                                    List of roles
    Role name    |                         Attributes                         |
                 Member of
-----------------+------------------------------------------------------------
+-----------------------------------------------------------
 rdssu1          |                                                            |
 {rds_superuser}
 rdssu2          |                                                            |
 {rds_superuser}
 user1           |                                                            | {}
```

Alle Benutzer, einschließlich Benutzer mit `rds_superuser`-Berechtigungen, dürfen ihre eigenen Benutzerzuordnungen (`umoptions`) in der `pg_user_mappings`-Tabelle anzeigen. Wie im folgenden Beispiel gezeigt, wird trotz `rdssu1rds_superuser`-Berechtigungen ein Fehler gemeldet, wenn `rdssu1` versucht, alle Benutzerzuordnungen abzurufen:

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

Im Folgenden sind einige Beispiele aufgeführt.

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
 (3 rows)

test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+-------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)
```

Aufgrund von Unterschieden in der Implementierung von `information_schema._pg_user_mappings` und `pg_catalog.pg_user_mappings` erfordert ein manuell erstelltes `rds_superuser` zusätzliche Berechtigungen zum Anzeigen von Passwörtern in `pg_catalog.pg_user_mappings`.

Es sind keine zusätzlichen Berechtigungen für `rds_superuser` nötig, um Kennwörter in `information_schema._pg_user_mappings` anzuzeigen.

Benutzer, die nicht über die `rds_superuser`-Rolle verfügen können Passwörter in `pg_user_mappings` nur unter den folgenden Bedingungen anzeigen:

- Der aktuelle Benutzer ist der zugeordnete Benutzer und besitzt den Server oder besitzt die `USAGE`-Berechtigung dafür.
- Der aktuelle Benutzer ist der Serverbesitzer und das Mapping ist für `PUBLIC`.

Arbeiten mit SQL-Server-Datenbanken unter Verwendung der Erweiterung tds_fdw

Sie können die PostgreSQL-Erweiterung `tds_fdw` für den Zugriff auf Datenbanken verwenden, die das TDS-Protokoll (Tabular Data Stream) unterstützen, wie Sybase- und Microsoft-SQL-Server-Datenbanken. Mit diesem Fremddaten-Wrapper können Sie sich von Ihrem Aurora-PostgreSQL-DB-Cluster aus mit Datenbanken verbinden, die das TDS-Protokoll verwenden, einschließlich Amazon RDS for Microsoft SQL Server. Weitere Informationen finden Sie in der tds-fdw/tds_fdw-Dokumentation auf GitHub.

Die Erweiterung `tds_fdw` wird von den Amazon-Aurora-PostgreSQL-Versionen 13.6 und höher unterstützt.

Einrichten Ihrer Aurora-PostgreSQL-DB zur Verwendung der tds_fdw-Erweiterung

In den folgenden Verfahren finden Sie ein Beispiel für die Einrichtung und Verwendung der Erweiterung `tds_fdw` mit einem Aurora-PostgreSQL-DB-Cluster. Bevor Sie eine Verbindung mit einer SQL-Server-Datenbank mithilfe von `tds_fdw` herstellen können, benötigen Sie die folgenden Details für die Instance:

- Hostname oder Endpunkt. Sie können den Endpunkt für eine RDS-for-SQL-Server-DB-Instance mithilfe der Konsole ermitteln. Wählen Sie die Registerkarte „Connectivity & security“ (Konnektivität und Sicherheit) aus und sehen Sie im Abschnitt „Endpoint and port“ (Endpunkt und Port) nach.
- Port-Nummer. Die Standardport-Nummer für Microsoft SQL Server ist 1433.
- Name der Datenbank. Die DB-ID.

Sie müssen auch Zugriff auf die Sicherheitsgruppe oder die Zugriffssteuerungsliste (ACL) für den SQL-Server-Port 1433 gewähren. Sowohl der Aurora-PostgreSQL-DB-Cluster als auch die RDS-for-SQL-Server-DB-Instance benötigen Zugriff auf Port 1433. Wenn der Zugriff nicht richtig konfiguriert ist, wird beim Versuch, den Microsoft SQL Server abzufragen, die folgende Fehlermeldung angezeigt:

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
```

```
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

So verbinden Sie sich mit tds_fdw mit einer SQL-Server-Datenbank

1. Verbinden Sie sich mit Ihrer primären Instance des Aurora-PostgreSQL-DB-Clusters über ein Konto mit der `rds_superuser`-Rolle:

   ```
   psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=test --password
   ```

2. Installieren Sie die `tds_fdw`-Erweiterung.

   ```
   test=> CREATE EXTENSION tds_fdw;
   CREATE EXTENSION
   ```

Nachdem die Erweiterung auf Ihrem Aurora-PostgreSQL-DB-Cluster installiert wurde, richten Sie den fremden Server ein.

So erstellen Sie den fremden Server

Führen Sie diese Aufgaben auf dem Aurora-PostgreSQL-DB-Cluster unter Verwendung eines Kontos mit `rds_superuser`-Berechtigungen aus.

1. Erstellen Sie einen fremden Server auf dem Aurora-PostgreSQL-DB-Cluster:

   ```
   test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
    (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
    'tds_fdw_testing');
   CREATE SERVER
   ```

   Um auf Nicht-ASCII-Daten auf der SQLServer-Seite zuzugreifen, erstellen Sie einen Serverlink mit der Option character_set im DB-Cluster von Aurora PostgreSQL:

   ```
   test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
    'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
    character_set 'UTF-8');
   CREATE SERVER
   ```

2. Gewähren Sie einem Benutzer, der keine `rds_superuser`-Rollenberechtigungen hat, Berechtigungen, zum Beispiel `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. Stellen Sie eine Verbindung als „user1“ (Benutzer 1) her und erstellen Sie dann eine Zuordnung zum SQL-Server-Benutzer:

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. Erstellen Sie eine fremde Tabelle, die mit einer SQL-Server-Tabelle verknüpft ist:

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. Fragen Sie die fremde Tabelle ab:

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

Verwenden der Verschlüsselung während der Übertragung für die Verbindung

Die Verbindung von Aurora PostgreSQL mit SQL Server verwendet je nach SQL-Server-Datenbankkonfiguration die Verschlüsselung während der Übertragung (TLS/SSL). Wenn der SQL Server nicht für die Verschlüsselung konfiguriert ist, bleibt der RDS-für-PostgreSQL-Client, der die Anforderung an die SQL-Server-Datenbank stellt, unverschlüsselt.

Sie können die Verschlüsselung für die Verbindung mit RDS-for-SQL-Server DB-Instances erzwingen, indem Sie den `rds.force_ssl`-Parameter festlegen. Weitere Informationen finden Sie unter Erzwingen von Verbindungen mit Ihrer DB-Instance, um SSL zu verwenden. Weitere Informationen zur SSL/TLS-Konfiguration für RDS for SQL Server finden Sie unter Verwenden von SSL mit einer Microsoft-SQL-Server-DB-Instance.

Arbeiten mit Trusted Language Extensions für PostgreSQL

Trusted Language Extensions für PostgreSQL ist ein Open-Source-Entwicklungskit für die Erstellung von PostgreSQL-Erweiterungen. Es ermöglicht Ihnen, leistungsstarke PostgreSQL-Erweiterungen zu erstellen und diese sicher auf Ihrem DB-Cluster von Aurora PostgreSQL auszuführen. Mithilfe von Trusted Language Extensions (TLE) für PostgreSQL können Sie PostgreSQL-Erweiterungen erstellen, die dem dokumentierten Ansatz zur Erweiterung der PostgreSQL-Funktionalität folgen. Weitere Informationen finden Sie unter Packaging Related Objects in a Extension in der PostgreSQL-Dokumentation.

Ein wesentlicher Vorteil von TLE besteht darin, dass Sie es in Umgebungen verwenden können, die keinen Zugriff auf das der PostgreSQL-Instance zugrunde liegende Dateisystem bieten. Bisher war für die Installation einer neuen Erweiterung Zugriff auf das Dateisystem erforderlich. Mit TLE entfällt diese Einschränkung. Es bietet eine Entwicklungsumgebung für die Erstellung neuer Erweiterungen für jede PostgreSQL-Datenbank, einschließlich solcher, die auf Ihren DB-Clustern von Aurora PostgreSQL ausgeführt werden.

TLE wurde entwickelt, um den Zugriff auf unsichere Ressourcen für die Erweiterungen zu verhindern, die Sie mit TLE erstellen. Die Laufzeitumgebung begrenzt die Auswirkungen eines Erweiterungsdefekts auf eine einzelne Datenbankverbindung. TLE verleiht Datenbankadministratoren auch eine detaillierte Kontrolle darüber, wer Erweiterungen installieren kann, und bietet ein Berechtigungsmodell für deren Ausführung.

TLE wird von Aurora PostgreSQL Version 14.5 und höheren Versionen unterstützt.

Die Entwicklungsumgebung und Laufzeit von Trusted Language Extensions sind als `pg_tle`-PostgreSQL-Erweiterung, Version 1.0.1, verpackt. Das Paket unterstützt das Erstellen von Erweiterungen in JavaScript, Perl, Tcl, PL/pgSQL und SQL. Sie installieren die `pg_tle`-Erweiterung in Ihrem DB-Cluster von Aurora PostgreSQL auf die gleiche Weise wie andere PostgreSQL-Erweiterungen. Nach der Einrichtung von `pg_tle` können Entwickler damit neue PostgreSQL-Erweiterungen, sogenannte TLE-Erweiterungen, erstellen.

In den folgenden Themen finden Sie Informationen darüber, wie Sie Trusted Language Extensions einrichten und Ihre eigenen TLE-Erweiterungen erstellen.

Themen

- Terminologie
- Anforderungen für die Verwendung von Trusted Language Extensions für PostgreSQL

- Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL
- Übersicht über Trusted Language Extensions für PostgreSQL
- Erstellen von TLE-Erweiterungen für Aurora PostgreSQL
- Löschen Ihrer TLE-Erweiterungen aus einer Datenbank
- Deinstallieren von Trusted Language Extensions für PostgreSQL
- Verwenden von PostgreSQL-Haken mit Ihren TLE-Erweiterungen
- Funktionsreferenz für Trusted Language Extensions für PostgreSQL
- Hakenreferenz für Trusted Language Extensions für PostgreSQL

Terminologie

Sehen Sie sich das folgende Glossar und die in diesem Thema verwendeten Begriffe an, damit Sie Trusted Language Extensions besser verstehen.

Trusted Language Extensions für PostgreSQL

: Trusted Language Extensions für PostgreSQL ist der offizielle Name des Open-Source-Entwicklungskits, das als `pg_tle`-Erweiterung verpackt ist. Es ist für die Verwendung in jedem PostgreSQL-System verfügbar. Weitere Informationen finden Sie unter aws/pg_tle auf GitHub.

Trusted Language Extensions

: Trusted Language Extensions ist der Kurzname für Trusted Language Extensions für PostgreSQL. Dieser verkürzte Name und seine Abkürzung (TLE) werden ebenfalls in dieser Dokumentation verwendet.

Vertrauenswürdige Sprache

: Eine vertrauenswürdige Sprache ist eine Programmier- oder Skriptsprache mit bestimmten Sicherheitsattributen. Beispielsweise schränken vertrauenswürdige Sprachen in der Regel den Zugriff auf das Dateisystem und die Verwendung bestimmter Netzwerkeigenschaften ein. Das TLE-Entwicklungskit wurde entwickelt, um vertrauenswürdige Sprachen zu unterstützen. PostgreSQL unterstützt verschiedene Sprachen, die verwendet werden, um vertrauenswürdige oder nicht vertrauenswürdige Erweiterungen zu erstellen. Ein Beispiel finden Sie unter Trusted and Untrusted PL/Perl in der PostgreSQL-Dokumentation. Wenn Sie eine Erweiterung mithilfe von Trusted Language Extensions erstellen, verwendet die Erweiterung von sich aus vertrauenswürdige Sprachmechanismen.

TLE-Erweiterung

Eine TLE-Erweiterung ist eine PostgreSQL-Erweiterung, die mithilfe des Trusted Language Extensions (TLE)-Entwicklungskits erstellt wurde.

Anforderungen für die Verwendung von Trusted Language Extensions für PostgreSQL

Beachten Sie die folgenden Anforderungen für die Einrichtung und Verwendung des TLE-Entwicklungskits.

- Aurora-PostgreSQL Versionen – Trusted Language Extensions wird nur von Aurora-PostgreSQL Version 14.5, und höheren Versionen unterstützt.
  - Wenn Sie Ihren DB-Cluster von Aurora PostgreSQL aktualisieren müssen, finden Sie weitere Informationen unter Upgrade von DB-Clustern von Amazon Aurora PostgreSQL.
  - Wenn Sie noch keinen Aurora-DB-Cluster für die Ausführung von PostgreSQL haben, können Sie eine/n erstellen. Weitere Informationen finden Sie unter Erstellen eines DB-Clusters von Aurora PostgreSQL und Herstellen einer Verbindung.
- Erfordert **`rds_superuser`**-Berechtigungen – Um die `pg_tle`-Erweiterung einzurichten und zu konfigurieren, muss Ihre Datenbankbenutzerrolle über die Berechtigungen der `rds_superuser`-Rolle verfügen. Diese Rolle wird standardmäßig dem `postgres`-Benutzer zugewiesen, der den DB-Cluster von Aurora PostgreSQL erstellt.
- Erfordert eine benutzerdefinierte DB-Parametergruppe – Ihr DB-Cluster von Aurora PostgreSQL muss mit einer benutzerdefinierten DB-Parametergruppe konfiguriert sein. Sie verwenden die benutzerdefinierte DB-Parametergruppe für die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL.
  - Wenn Ihr DB-Cluster von Aurora PostgreSQL nicht mit einer benutzerdefinierten DB-Parametergruppe konfiguriert ist, sollten Sie eine erstellen und sie der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL zuordnen. Eine kurze Zusammenfassung der Schritte finden Sie unter Erstellen und Anwenden einer benutzerdefinierten DB-Parametergruppe.
  - Wenn Ihr DB-Cluster von Aurora PostgreSQL bereits mit einer benutzerdefinierten DB-Parametergruppe konfiguriert ist, können Sie Trusted Language Extensions einrichten. Details hierzu finden Sie unter Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL.

Erstellen und Anwenden einer benutzerdefinierten DB-Parametergruppe

Verwenden Sie die folgenden Schritte, um eine benutzerdefinierte DB-Parametergruppe zu erstellen und Ihren DB-Cluster von Aurora PostgreSQL für ihre Verwendung zu konfigurieren.

Konsole

So erstellen Sie eine benutzerdefinierte DB-Parametergruppe und verwenden Sie mit Ihrem DB-Cluster von Aurora PostgreSQL

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Menü von Amazon RDS „Parameter groups“ (Parametergruppen) aus.
3. Wählen Sie Create parameter group (Parametergruppe erstellen).
4. Geben Sie auf der Seite Parameter group details Parametergruppendetails die folgenden Informationen ein.
   - Wählen Sie unter Parameter group family (Parametergruppenfamilie) die Option aurora-postgresql14 aus.
   - Wählen Sie für Type (Typ) die Option DB Parameter Group (DB-Parametergruppe) aus.
   - Geben Sie Ihrer Parametergruppe unter Group name (Gruppenname) im Kontext Ihrer Operationen einen aussagekräftigen Namen.
   - Geben Sie unter Description (Beschreibung) eine nützliche Beschreibung ein, damit andere Mitglieder Ihres Teams sie leicht finden können.
5. Wählen Sie Erstellen aus. Ihre benutzerdefinierte DB-Parametergruppe wird in Ihrer AWS-Region erstellt. Sie können jetzt Ihren DB-Cluster von Aurora PostgreSQL ändern, um sie zu verwenden, indem Sie die nächsten Schritte ausführen.
6. Wählen Sie Databases (Datenbanken) im Amazon-RDS-Menü aus.
7. Wählen Sie aus der Liste den DB-Cluster von Aurora PostgreSQL für die Verwendung mit TLE aus und klicken Sie dann auf Modify (Ändern).
8. Suchen Sie auf der Seite zum Ändern der DB-Cluster-Einstellungen nach Database options (Datenbankoptionen) und verwenden Sie die Auswahl, um Ihre benutzerdefinierte DB-Parametergruppe auszuwählen.
9. Wählen Sie Continue (Weiter) aus, um die Änderung zu speichern.
10. Wählen Sie Apply immediately (Sofort anwenden) aus, damit Sie die Einrichtung DB-Cluster von Aurora PostgreSQL zur Verwendung von TLE fortsetzen können.

Informationen zum weiteren Einrichten Ihres Systems für Trusted Language Extensions finden Sie unter Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL.

Weitere Informationen zum Arbeiten mit DB-Clustern und DB-Parametergruppen finden Sie unter DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster.

AWS CLI

Sie können die Angabe des `--region`-Arguments vermeiden, wenn Sie CLI-Befehle verwenden, indem Sie Ihre AWS CLI mit Ihrer standardmäßigen AWS-Region konfigurieren. Weitere Informationen finden Sie unter Konfigurationsgrundlagen im AWS Command Line Interface-Benutzerhandbuch.

So erstellen Sie eine benutzerdefinierte DB-Parametergruppe und verwenden Sie mit Ihrem DB-Cluster von Aurora PostgreSQL

1. Verwenden Sie den AWS CLI-Befehl create-db-parameter-group, um eine benutzerdefinierte DB-Parametergruppe basierend auf aurora-postgresql14 für Ihre AWS-Region zu erstellen. Beachten Sie, dass Sie in diesem Schritt eine DB-Parametergruppe zum Anwenden auf die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL erstellen.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family aurora-postgresql14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Für Windows:

   ```
   aws rds create-db-parameter-group ^
     --region aws-region ^
     --db-parameter-group-name custom-params-for-pg-tle ^
     --db-parameter-group-family aurora-postgresql14 ^
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Ihre benutzerdefinierte DB-Parametergruppe ist in Ihrer AWS-Region verfügbar. Sie können also die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL ändern, um sie zu verwenden.

2. Verwenden Sie den AWS CLI-Befehl modify-db-Instance, um Ihre benutzerdefinierte DB-Parametergruppe auf die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL anzuwenden. Mit diesem Befehl wird die aktive Instance sofort neu gestartet.

   Für Linux, macOS oder Unix:

```
aws rds modify-db-instance \
  --region aws-region \
  --db-instance-identifier your-writer-instance-name \
  --db-parameter-group-name custom-params-for-pg-tle \
  --apply-immediately
```

   Für Windows:

```
aws rds modify-db-instance ^
  --region aws-region ^
  --db-instance-identifier your-writer-instance-name ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --apply-immediately
```

Informationen zum weiteren Einrichten Ihres Systems für Trusted Language Extensions finden Sie unter Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL.

Weitere Informationen finden Sie unter DB-Parametergruppen für DB-Instances von Amazon Aurora .

Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL

Bei den folgenden Schritten wird davon ausgegangen, dass Ihr DB-Cluster von Aurora PostgreSQL einer benutzerdefinierten DB-Cluster -Parametergruppe zugeordnet ist. Für diese Schritte können Sie die AWS-Managementkonsole oder die AWS CLI verwenden.

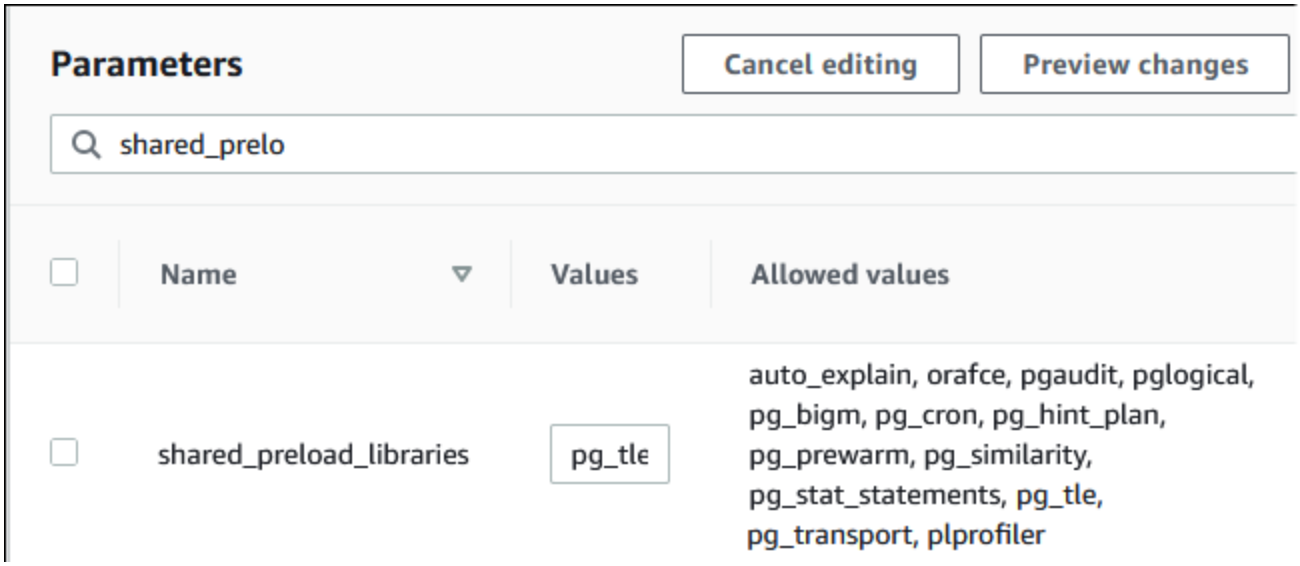
Wenn Sie Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL einrichten, installieren Sie sie in einer bestimmten Datenbank, damit sie von den Datenbankbenutzern verwendet werden kann, die über Berechtigungen für diese Datenbank verfügen.

Konsole

So richten Sie Trusted Language Extensions ein

Führen Sie die folgenden Schritte mit einem Konto aus, das Mitglied der `rds_superuser`-Gruppe (Rolle) ist.

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL aus.
3. Öffnen Sie die Registerkarte Configuration (Konfiguration) für Ihre Writer-Instance des DB-Clusters von Aurora PostgreSQL. Suchen Sie in den Instance-Details den Link Parameter group (Parametergruppe).
4. Wählen Sie den Link aus, um die benutzerdefinierten Parameter zu öffnen, die Ihrem DB-Cluster von Aurora PostgreSQL zugeordnet sind.
5. Geben Sie in das Suchfeld Parameters (Parameter) `shared_pre` ein, um den `shared_preload_libraries`-Parameter zu finden.
6. Wählen Sie Edit parameters (Parameter bearbeiten) aus, um auf die Eigenschaftswerte zuzugreifen.
7. Fügen Sie `pg_tle` der Liste im Feld Values (Werte) hinzu. Verwenden Sie ein Komma, um Elemente in der Werteliste zu trennen.



8. Starten Sie die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu, damit Ihre Änderung des `shared_preload_libraries`-Parameters wirksam wird.

9. Wenn die Instance verfügbar ist, überprüfen Sie, ob `pg_tle` initialisiert wurde. Stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her und führen Sie den folgenden Befehl aus.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. Wenn die `pg_tle`-Erweiterung initialisiert ist, können Sie jetzt die Erweiterung erstellen.

```
CREATE EXTENSION pg_tle;
```

Sie können überprüfen, ob die Erweiterung installiert wurde, indem Sie den folgenden `psql`-Metabefehl verwenden.

```
labdb=> \dx
                          List of installed extensions
  Name   | Version |   Schema   |                Description
---------+---------+------------+-------------------------------------------
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. Weisen Sie der `pgtle_admin`-Rolle dem primären Benutzernamen zu, den Sie bei der Einrichtung für Ihren DB-Cluster von Aurora PostgreSQL erstellt haben. Wenn Sie die Standardeinstellung akzeptiert haben, lautet der Wert `postgres`.

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

Wie in folgendem Beispiel veranschaulicht, können Sie anhand des `psql`-Metabefehls überprüfen, ob die Gewährung erfolgt ist. In der Ausgabe werden nur die Rollen `pgtle_admin` und `postgres` angezeigt. Weitere Informationen finden Sie unter Grundlegendes zu PostgreSQL-Rollen und -Berechtigungen.

```
labdb=> \du
                          List of roles
    Role name    |           Attributes           |              Member of
```

```
-----------------+----------------------------------
+----------------------------------
pgtle_admin     | Cannot login                      | {}
postgres        | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                | Password valid until infinity    |...
```

12. Schließen Sie die `psql`-Sitzung mit dem `\q`-Metabefehl.

```
\q
```

Informationen zum Erstellen von TLE-Erweiterungen finden Sie unter Beispiel: Erstellen einer Trusted Language Extension mit SQL.

AWS CLI

Sie können die Angabe des `--region`-Arguments vermeiden, wenn Sie CLI-Befehle verwenden, indem Sie Ihre AWS CLI mit Ihrer standardmäßigen AWS-Region konfigurieren. Weitere Informationen finden Sie unter Konfigurationsgrundlagen im AWS Command Line Interface Benutzerhandbuch.

So richten Sie Trusted Language Extensions ein

1. Verwenden Sie den AWS CLI-Befehl modify-db-parameter-group, um dem `shared_preload_libraries`-Parameter `pg_tle` hinzuzufügen.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Verwenden Sie den AWS CLI-Befehl reboot-db-instance, um die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL neu zu starten und die `pg_tle`-Bibliothek zu initialisieren.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Wenn die Instance verfügbar ist, können Sie überprüfen, ob `pg_tle` initialisiert wurde. Stellen Sie über `psql` eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her und führen Sie den folgenden Befehl aus.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

   Sobald `pg_tle` initialisiert ist, können Sie die Erweiterung erstellen.

```
CREATE EXTENSION pg_tle;
```

4. Weisen Sie der `pgtle_admin`-Rolle dem primären Benutzernamen zu, den Sie bei der Einrichtung für Ihren DB-Cluster von Aurora PostgreSQL erstellt haben. Wenn Sie die Standardeinstellung akzeptiert haben, lautet der Wert `postgres`.

```
GRANT pgtle_admin TO postgres;
GRANT ROLE
```

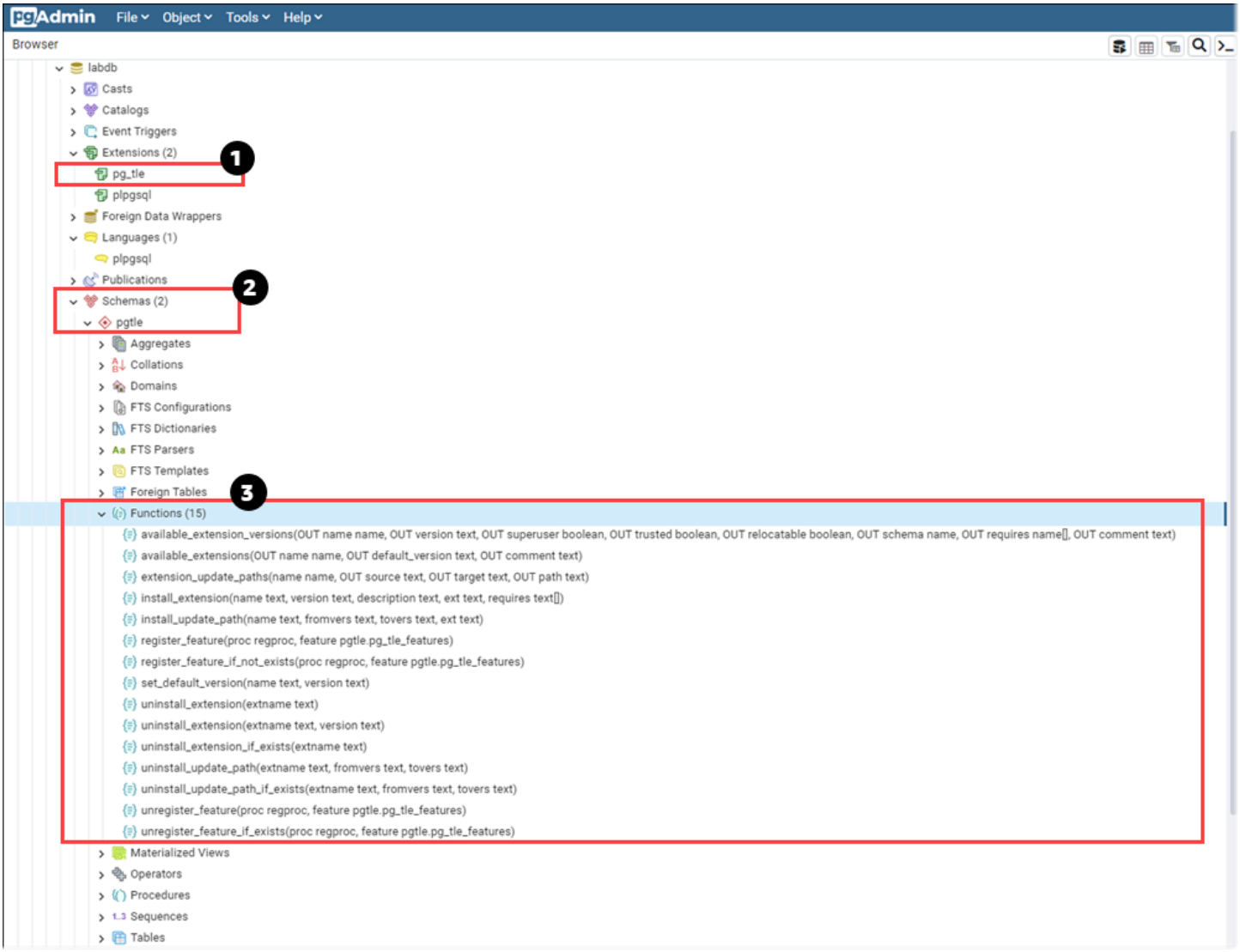
5. Schließen Sie die `psql`-Sitzung wie folgt.

```
labdb=> \q
```

Informationen zum Erstellen von TLE-Erweiterungen finden Sie unter Beispiel: Erstellen einer Trusted Language Extension mit SQL.

Übersicht über Trusted Language Extensions für PostgreSQL

Trusted Language Extensions für PostgreSQL ist eine PostgreSQL-Erweiterung, die Sie in Ihrem DB-Cluster von Aurora PostgreSQL auf die gleiche Weise installieren, wie Sie andere PostgreSQL-Erweiterungen einrichten. In der folgenden Abbildung einer Beispieldatenbank im pgAdmin-Client-Tool können Sie einige der Komponenten sehen, aus denen die `pg_tle`-Erweiterung besteht.



Sie können die folgenden Details sehen.

1. Das Entwicklungskit von Trusted Language Extensions (TLE) für PostgreSQL ist als `pg_tle`-Erweiterung verpackt. Daher wird `pg_tle` den verfügbaren Erweiterungen für die Datenbank hinzugefügt, in der es installiert ist.
2. TLE hat ein eigenes Schema, `pgtle`. Dieses Schema enthält Hilfsfunktionen (3) für die Installation und Verwaltung der von Ihnen erstellten Erweiterungen.
3. TLE bietet über ein Dutzend Hilfsfunktionen für die Installation, Registrierung und Verwaltung Ihrer Erweiterungen. Weitere Informationen zu diesen Funktionen finden Sie unter Funktionsreferenz für Trusted Language Extensions für PostgreSQL.

Das `pg_tle`-Erweiterungspaket umfasst außerdem folgende Komponenten:

- Die **`pgtle_admin`**-Rolle – Die `pgtle_admin`-Rolle wird erstellt, wenn die `pg_tle`-Erweiterung installiert wird. Diese Rolle ist privilegiert und sollte entsprechend behandelt werden. Es wird dringend empfohlen, bei der Gewährung der `pgtle_admin`-Rolle an Datenbankbenutzer dem Prinzip der geringsten Berechtigung zu folgen. Mit anderen Worten, weisen Sie die `pgtle_admin`-Rolle nur Datenbankbenutzern zu, die berechtigt sind, neue TLE-Erweiterungen zu erstellen, zu installieren und zu verwalten, wie z. b. `postgres`.
- Die **`pgtle.feature_info`**-Tabelle – Die `pgtle.feature_info`-Tabelle ist eine geschützte Tabelle, die Informationen über Ihre TLEs, Haken und die von ihnen verwendeten benutzerdefinierten gespeicherten Prozeduren und Funktionen enthält. Wenn Sie über `pgtle_admin`-Berechtigungen verfügen, verwenden Sie die folgenden Funktionen von Trusted Language Extensions, um diese Informationen in der Tabelle hinzuzufügen und zu aktualisieren.
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

Erstellen von TLE-Erweiterungen für Aurora PostgreSQL

Sie können alle Erweiterungen, die Sie mit TLE erstellen, in jedem beliebigen DB-Cluster von Aurora PostgreSQL installieren, sofern die die `pg_tle`-Erweiterung darauf installiert ist. Die `pg_tle`-Erweiterung ist auf die PostgreSQL-Datenbank beschränkt, in der sie installiert ist. Die Erweiterungen, die Sie mit TLE erstellen, sind auf dieselbe Datenbank ausgelegt.

Verwenden Sie die verschiedenen `pgtle`-Funktionen, um den Code zu installieren, aus dem Ihre TLE-Erweiterung besteht. Die folgenden Funktionen von Trusted Language Extensions erfordern alle die `pgtle_admin`-Rolle.

- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)

- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

Beispiel: Erstellen einer Trusted Language Extension mit SQL

Das folgende Beispiel zeigt Ihnen, wie Sie eine TLE-Erweiterung namens `pg_distance` erstellen, die einige SQL-Funktionen für die Berechnung von Entfernungen mit verschiedenen Formeln enthält. In der Liste finden Sie die Funktion zur Berechnung der Manhattan-Distanz und die Funktion zur Berechnung des euklidischen Abstands. Weitere Informationen zum Unterschied zwischen diesen Formeln finden Sie unter Taxi-Geometerie und Euklidische Geometrie in Wikipedia.

Sie können dieses Beispiel in Ihrem eigenen DB-Cluster von Aurora PostgreSQL verwenden, wenn Sie die `pg_tle`-Erweiterung wie unter Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL beschrieben eingerichtet haben.

> **Note**
>
> Sie benötigen die Rechte der `pgtle_admin`-Rolle, um dieses Verfahren ausführen zu können.

So erstellen Sie die TLE-Beispielerweiterung

In den folgenden Schritten wird eine Beispieldatenbank namens `labdb` verwendet. Diese Datenbank gehört dem `postgres`-Hauptbenutzer. Die `postgres`-Rolle verfügt auch über die Berechtigungen der `pgtle_admin`-Rolle.

1. Verwenden Sie `psql`, um eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL herzustellen.

   ```
   psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
   --port=5432 --username=postgres --password --dbname=labdb
   ```

2. Erstellen Sie eine TLE-Erweiterung mit dem Namen `pg_distance`, indem Sie den folgenden Code kopieren und in Ihre `psql`-Sitzungskonsole einfügen.

```
SELECT pgtle.install_extension
(
 'pg_distance',
 '0.1',
  'Distance functions for two points',
$_pg_tle_$
    CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;

    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

Die Ausgabe sollte folgendermaßen aussehen.

```
install_extension
---------------
 t
(1 row)
```

Die Artefakte, aus denen die `pg_distance`-Erweiterung besteht, sind jetzt in Ihrer Datenbank installiert. Zu diesen Artefakten gehören die Steuerdatei und der Code für die Erweiterung. Diese Elemente müssen vorhanden sein, damit die Erweiterung mit dem `CREATE EXTENSION`-Befehl erstellt werden kann. Mit anderen Worten, Sie müssen die Erweiterung nach wie vor erstellen, um ihre Funktionen Datenbankbenutzern zur Verfügung zu stellen.

3. Um die Erweiterung zu erstellen, verwenden Sie den `CREATE EXTENSION`-Befehl wie für jede andere Erweiterung. Wie bei anderen Erweiterungen muss der Datenbankbenutzer über die `CREATE`-Berechtigungen in der Datenbank verfügen.

```
CREATE EXTENSION pg_distance;
```

4. Um die `pg_distance`-TLE-Erweiterung zu testen, können Sie sie verwenden, um die Manhattan-Distanz zwischen vier Punkten zu berechnen.

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
8
```

Um den euklidischen Abstand zwischen derselben Menge von Punkten zu berechnen, können Sie Folgendes verwenden.

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
5.656854249492381
```

Die `pg_distance`-Erweiterung lädt die Funktionen in die Datenbank und stellt sie allen Benutzern mit Berechtigungen für die Datenbank zur Verfügung.

Ändern Ihrer TLE-Erweiterung

Um die Abfrageleistung für die in dieser TLE-Erweiterung enthaltenen Funktionen zu verbessern, fügen Sie ihren Spezifikationen die beiden folgenden PostgreSQL-Attribute hinzu.

- `IMMUTABLE` – Das `IMMUTABLE`-Attribut stellt sicher, dass der Abfrageoptimierer Optimierungen verwenden kann, um die Antwortzeiten von Abfragen zu verbessern. Weitere Informationen finden Sie im Abschnitt Volatilitätskategorien von Funktionen der PostgreSQL-Dokumentation.
- `PARALLEL SAFE` – Das `PARALLEL SAFE`-Attribut ist ein weiteres Attribut, das es PostgreSQL ermöglicht, die Funktion im Parallelmodus auszuführen. Weitere Informationen finden Sie im Abschnitt CREATE FUNCTION der PostgreSQL-Dokumentation.

Im folgenden Beispiel können Sie sehen, wie die `pgtle.install_update_path`-Funktion verwendet wird, um diese Attribute jeder Funktion hinzuzufügen, um eine Version `0.2` der `pg_distance`-TLE-Erweiterung zu erstellen. Weitere Informationen zu dieser Funktion finden

Sie unter pgtle.install_update_path. Sie benötigen die `pgtle_admin` Rolle, um diese Aufgabe auszuführen.

So aktualisieren Sie eine vorhandene TLE-Erweiterung und geben die Standardversion an

1. Stellen Sie über `psql` oder ein anderes Client-Tool wie pgAdmin eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Ändern Sie die vorhandene TLE-Erweiterung, indem Sie den folgenden Code kopieren und in Ihre `psql`-Sitzungskonsole einfügen.

```
SELECT pgtle.install_update_path
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

Es wird eine Antwort ähnlich dem folgenden Beispiel angezeigt.

```
install_update_path
--------------------
 t
(1 row)
```

Sie können diese Version der Erweiterung zur Standardversion machen, sodass Datenbankbenutzer keine Version angeben müssen, wenn sie die Erweiterung in ihrer Datenbank erstellen oder aktualisieren.

3. Um anzugeben, dass die modifizierte Version (Version 0.2) Ihrer TLE-Erweiterung die Standardversion ist, verwenden Sie die `pgtle.set_default_version`-Funktion wie im folgenden Beispiel gezeigt.

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

Weitere Informationen zu dieser Funktion finden Sie unter pgtle.set_default_version.

4. Wenn der Code vorhanden ist, können Sie die installierte TLE-Erweiterung wie gewohnt aktualisieren, indem Sie den `ALTER EXTENSION ... UPDATE`-Befehl verwenden, wie hier gezeigt:

```
ALTER EXTENSION pg_distance UPDATE;
```

Löschen Ihrer TLE-Erweiterungen aus einer Datenbank

Sie können Ihre TLE-Erweiterungen löschen, indem Sie den `DROP EXTENSION`-Befehl auf die gleiche Weise wie für andere PostgreSQL-Erweiterungen verwenden. Durch das Löschen der Erweiterung werden die Installationsdateien, aus denen die Erweiterung besteht, nicht entfernt, sodass Benutzer die Erweiterung neu erstellen können. Gehen Sie wie folgt in zwei Schritten vor, um die Erweiterung und ihre Installationsdateien zu entfernen.

So löschen Sie die TLE-Erweiterung und entfernen ihre Installationsdateien

1. Stellen Sie über `psql` oder ein anderes Client-Tool eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. Löschen Sie die Erweiterung so wie jede andere PostgreSQL-Erweiterung.

```
DROP EXTENSION your-TLE-extension
```

Wenn Sie die `pg_distance`-Erweiterung beispielsweise wie in beschrieben Beispiel: Erstellen einer Trusted Language Extension mit SQL erstellen, können Sie die Erweiterung wie folgt löschen.

```
DROP EXTENSION pg_distance;
```

Es wird eine Ausgabe angezeigt, die bestätigt, dass die Erweiterung gelöscht wurde, wie im Folgenden gezeigt.

```
DROP EXTENSION
```

Zu diesem Zeitpunkt ist die Erweiterung in der Datenbank nicht mehr aktiv. Die Installationsdateien und die Steuerdatei sind jedoch nach wie vor in der Datenbank verfügbar, sodass Datenbankbenutzer die Erweiterung erneut erstellen können, wenn sie möchten.

- Wenn Sie die Erweiterungsdateien intakt lassen möchten, damit Datenbankbenutzer Ihre TLE-Erweiterung erstellen können, können Sie an dieser Stelle aufhören.
- Wenn Sie alle Dateien, aus denen die Erweiterung besteht, entfernen möchten, fahren Sie mit dem nächsten Schritt fort.

3. Verwenden Sie die `pgtle.uninstall_extension`-Funktion, um alle Installationsdateien für Ihre Erweiterung zu entfernen. Diese Funktion entfernt dne gesamten Code und die Steuerdateien für Ihre Erweiterung.

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

Verwenden Sie beispielsweise den folgenden Befehl, um alle `pg_distance`-Installationsdateien zu entfernen.

```
SELECT pgtle.uninstall_extension('pg_distance');
```

```
 uninstall_extension
---------------------
 t
(1 row)
```

Deinstallieren von Trusted Language Extensions für PostgreSQL

Wenn Sie keine eigenen TLE-Erweiterungen mehr mit TLE erstellen möchten, können Sie die `pg_tle`-Erweiterung löschen und alle Artefakte entfernen. Diese Aktion beinhaltet das Löschen aller TLE-Erweiterungen in der Datenbank und das Entfernen des`pgtle`-Schemas.

So löschen Sie die **`pg_tle`**-Erweiterung und ihr Schema aus einer Datenbank

1. Stellen Sie über `psql` oder ein anderes Client-Tool eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL her.

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password --dbname=dbname
   ```

2. Löschen Sie die `pg_tle`-Erweiterung aus der Datenbank. Wenn Ihre eigenen TLE-Erweiterungen noch in der Datenbank laufen, müssen Sie diese Erweiterungen ebenfalls löschen. Dazu können Sie das `CASCADE`-Schlüsselwort verwenden, wie im Folgenden gezeigt.

   ```
   DROP EXTENSION pg_tle CASCADE;
   ```

   Wenn die `pg_tle`-Erweiterung in der Datenbank nicht mehr aktiv ist, müssen Sie das `CASCADE`-Schlüsselwort nicht verwenden.

3. Löschen Sie das `pgtle`-Schema. Mit dieser Aktion werden alle Verwaltungsfunktionen aus der Datenbank entfernt.

   ```
   DROP SCHEMA pgtle CASCADE;
   ```

   Der Befehl gibt nach Abschluss des Vorgangs Folgendes zurück.

   ```
   DROP SCHEMA
   ```

   Die `pg_tle`-Erweiterung, ihr Schema und ihre Funktionen sowie alle Artefakte werden entfernt. Um neue Erweiterungen mit TLE zu erstellen, führen Sie den Einrichtungsvorgang erneut durch.

Weitere Informationen finden Sie unter Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL.

Verwenden von PostgreSQL-Haken mit Ihren TLE-Erweiterungen

Ein Haken ist ein in PostgreSQL verfügbarer Callback-Mechanismus, der es Entwicklern ermöglicht, benutzerdefinierte Funktionen oder andere Routinen während regulärer Datenbankoperationen aufzurufen. Das TLE-Entwicklungskit unterstützt PostgreSQL-Haken, sodass Sie benutzerdefinierte Funktionen zur Laufzeit in das PostgreSQL-Verhalten integrieren können. Sie können beispielsweise einen Haken verwenden, um den Authentifizierungsprozess mit Ihrem eigenen benutzerdefinierten Code zu verknüpfen oder um den Planungs- und Ausführungsprozess für Abfragen Ihren spezifischen Bedürfnissen entsprechend anzupassen.

Ihre TLE-Erweiterungen können Haken verwenden. Wenn ein Haken einen globalen Gültigkeitsbereich hat, gilt er für alle Datenbanken. Wenn Ihre TLE-Erweiterung einen globalen Haken verwendet, müssen Sie Ihre TLE-Erweiterung daher in allen Datenbanken erstellen, auf die Ihre Benutzer zugreifen können.

Wenn Sie die `pg_tle`-Erweiterung verwenden, um Ihre eigenen Trusted Language Extensions zu erstellen, können Sie die verfügbaren Haken einer SQL-API verwenden, um die Funktionen Ihrer Erweiterung zu erstellen. Sie sollten alle Haken bei `pg_tle` registrieren. Für einige Haken müssen Sie möglicherweise auch verschiedene Konfigurationsparameter festlegen. Der `passcode`-Prüfungshaken kann beispielsweise auf ein, aus oder erforderlich festgelegt werden. Weitere Hinweise zu den spezifischen Anforderungen für verfügbare `pg_tle`-Haken finden Sie unter Hakenreferenz für Trusted Language Extensions für PostgreSQL.

Beispiel: Erstellen einer Erweiterung, die einen PostgreSQL-Haken verwendet

Das in diesem Abschnitt besprochene Beispiel verwendet einen PostgreSQL-Haken, um das bei bestimmten SQL-Vorgängen angegebene Passwort zu überprüfen, und verhindert, dass Datenbankbenutzer ihre Passwörter auf eines der in der `password_check.bad_passwords`-Tabelle enthaltenen Passwörter festlegen. Die Tabelle enthält die zehn am häufigsten verwendeten, aber leicht zu knackenden Optionen für Passwörter.

Um dieses Beispiel in Ihrem DB-Cluster von Aurora PostgreSQL einzurichten, müssen Sie Trusted Language Extensions bereits installiert haben. Details hierzu finden Sie unter Einrichten von Trusted Language Extensions in Ihrem DB-Cluster von Aurora PostgreSQL.

So richten Sie das Beispiel für einen Haken für die Passwortüberprüfung ein

1. Verwenden Sie `psql`, um eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL herzustellen.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Kopieren Sie den Code von Liste der Hakencodes für die Passwortüberprüfung und fügen Sie ihn in Ihre Datenbank ein.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
```

```
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

Wenn die Erweiterung in Ihre Datenbank geladen wurde, sehen Sie eine Ausgabe ähnlich wie die folgende.

```
 install_extension
-------------------
 t
(1 row)
```

3. Während Sie noch mit der Datenbank verbunden sind, können Sie jetzt die Erweiterung erstellen.

```
CREATE EXTENSION my_password_check_rules;
```

4. Mit dem folgenden `psql`-Metabefehl können Sie bestätigen, dass die Erweiterung in der Datenbank erstellt wurde.

```
\dx
                        List of installed extensions
          Name           | Version |   Schema   |
 Description
-------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. Öffnen Sie eine weitere Terminalsitzung, um mit der AWS CLI zu arbeiten. Sie müssen Ihre benutzerdefinierte DB-Parametergruppe ändern, um den Haken für die Passwortüberprüfung zu aktivieren. Verwenden Sie dazu den CLI-Befehl modify-db-parameter-group, wie im folgenden Beispiel gezeigt.

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

   Es kann einige Minuten dauern, bis die Änderung der Parametergruppe wirksam wird. Dieser Parameter ist jedoch dynamisch, sodass Sie die Writer-Instance des DB-Clusters von Aurora PostgreSQL nicht neu starten müssen, damit die Einstellung wirksam wird.

6. Öffnen Sie die `psql`-Sitzung und fragen Sie die Datenbank ab, um zu überprüfen, ob der password_check-Haken aktiviert wurde.

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
----------------------------
on
(1 row)
```

Der Haken für die Passwortüberprüfung ist jetzt aktiv. Sie können dies testen, indem Sie eine neue Rolle erstellen und eines der schwachen Passwörter verwenden, wie im folgenden Beispiel gezeigt.

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

Die Ausgabe wurde zur besseren Lesbarkeit formatiert.

Das folgende Beispiel zeigt, dass das interaktive `pgsql`-Metabefehlverhalten `\password` auch vom password_check-Haken beeinflusst wird.

```
postgres=> SET password_encryption TO 'md5';
SET
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

Sie können diese TLE-Erweiterung löschen und ihre Quelldateien deinstallieren, wenn Sie möchten. Weitere Informationen finden Sie unter Löschen Ihrer TLE-Erweiterungen aus einer Datenbank.

Liste der Hakencodes für die Passwortüberprüfung

Der hier gezeigte Beispielcode definiert die Spezifikation für die `my_password_check_rules`-TLE-Erweiterung. Wenn Sie diesen Code kopieren und in Ihre Datenbank einfügen, wird der Code für die `my_password_check_rules`-Erweiterung in die Datenbank geladen und der `password_check`-Haken für die Verwendung durch die Erweiterung registriert.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
```

```
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
```

```
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

Funktionsreferenz für Trusted Language Extensions für PostgreSQL

Sehen Sie sich die folgende Referenzdokumentation zu den Funktionen an, die in Trusted Language Extensions für PostgreSQL verfügbar sind. Verwenden Sie diese Funktionen, um Ihre TLE-Erweiterungen, d. h. die PostgreSQL-Erweiterungen, die Sie mit dem Trusted Language Extensions Development Kit entwickeln, zu installieren, zu registrieren, zu aktualisieren und zu verwalten.

Funktionen

- pgtle.available_extensions
- pgtle.available_extension_versions
- pgtle.extension_update_paths
- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

pgtle.available_extensions

Die `pgtle.available_extensions`-Funktion ist eine Mengenrückgabefunktion. Sie gibt alle verfügbaren TLE-Erweiterungen in der Datenbank zurück. Jede zurückgegebene Zeile enthält Informationen zu einer einzelnen TLE-Erweiterung.

Funktionsprototyp

```
pgtle.available_extensions()
```

Rolle

Keine.

Argumente

Keine.

Output

- `name` – Der Name der TLE-Erweiterung.
- `default_version` – Die Version der TLE-Erweiterung, die verwendet werden soll, wenn `CREATE EXTENSION` ohne Angabe einer Version aufgerufen wird.
- `description` – Eine ausführlichere Beschreibung der TLE-Erweiterung.

Verwendungsbeispiel

```
SELECT * FROM pgtle.available_extensions();
```

pgtle.available_extension_versions

Die `available_extension_versions`-Funktion ist eine Mengenrückgabefunktion. Sie gibt eine Liste aller verfügbaren TLE-Erweiterungen und deren Versionen zurück. Jede Zeile enthält Informationen zu einer bestimmten Version der angegebenen TLE-Erweiterung, einschließlich der Frage, ob für sie eine bestimmte Rolle erforderlich ist.

Funktionsprototyp

```
pgtle.available_extension_versions()
```

Rolle

Keine.

Argumente

Keine.

Output

- `name` – Der Name der TLE-Erweiterung.
- `version` – Die Version der TLE-Erweiterung.
- `superuser` – Dieser Wert ist für Ihre TLE-Erweiterungen immer `false`. Die Berechtigungen, die zum Erstellen oder Aktualisieren der TLE-Erweiterung erforderlich sind, entsprechen denen, die für die Erstellung anderer Objekte in der angegebenen Datenbank notwendig sind.
- `trusted` – Dieser Wert ist für eine TLE-Erweiterung immer `false`.
- `relocatable` – Dieser Wert ist für eine TLE-Erweiterung immer `false`.
- `schema` – Gibt den Namen des Schemas an, in dem die TLE-Erweiterung installiert ist.
- `requires` – Ein Array, das die Namen anderer Erweiterungen enthält, die für diese TLE-Erweiterung benötigt werden.
- `description` – Eine ausführliche Beschreibung der TLE-Erweiterung.

Weitere Informationen zu Ausgabewerten finden Sie unter Packaging Related Objects into an Extension > Extension Files in der PostgreSQL-Dokumentation.

Verwendungsbeispiel

```
SELECT * FROM pgtle.available_extension_versions();
```

pgtle.extension_update_paths

Die `extension_update_paths`-Funktion ist eine Mengenrückgabefunktion. Sie gibt eine Liste aller möglichen Aktualisierungspfade für eine TLE-Erweiterung zurück. Jede Zeile enthält die verfügbaren Upgrades oder Downgrades für diese TLE-Erweiterung.

Funktionsprototyp

```
pgtle.extension_update_paths(name)
```

Rolle

Keine.

Argumente

`name` – Der Name der TLE-Erweiterung, von der die Upgrade-Pfade abgerufen werden sollen.

Output

- `source` – Die Quellversion für eine Aktualisierung.
- `target` – Die Zielversion für eine Aktualisierung.
- `path` – Der Upgrade-Pfad, der verwendet wird, um eine TLE-Erweiterung von der `source`-Version auf die `target`-Version zu aktualisieren, zum Beispiel `0.1--0.2`.

Verwendungsbeispiel

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

pgtle.install_extension

Mit dieser `install_extension`-Funktion können Sie die Artefakte, aus denen Ihre TLE-Erweiterung besteht, in der Datenbank installieren. Anschließend können sie mit dem `CREATE EXTENSION`-Befehl erstellt werden.

Funktionsprototyp

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

Rolle

Keine.

Argumente

- `name` – Der Name der TLE-Erweiterung. Dieser Wert wird beim Aufrufen von `CREATE EXTENSION` verwendet.
- `version` – Die Version der TLE-Erweiterung.

- `description` – Eine ausführliche Beschreibung der TLE-Erweiterung. Diese Beschreibung wird im Feld `comment` in `pgtle.available_extensions()` angezeigt.
- `ext` – Der Inhalt der TLE-Erweiterung. Dieser Wert enthält Objekte wie Funktionen.
- `requires` – Ein optionaler Parameter, der Abhängigkeiten für diese TLE-Erweiterung angibt. Die `pg_tle`-Erweiterung wird automatisch als Abhängigkeit hinzugefügt.

Viele dieser Argumente entsprechen denen, die in einer Erweiterungskontrolldatei für die Installation einer PostgreSQL-Erweiterung im Dateisystem einer PostgreSQL-Instance enthalten sind. Weitere Informationen finden Sie unter Extension Files in Packaging Related Objects in a Extension in der PostgreSQL-Dokumentation.

Output

Diese Funktion gibt bei Erfolg `OK` und bei Fehler `NULL` zurück.

- `OK` – Die TLE-Erweiterung wurde erfolgreich in der Datenbank installiert.
- `NULL` – Die TLE-Erweiterung wurde nicht erfolgreich in der Datenbank installiert.

Verwendungsbeispiel

```
SELECT pgtle.install_extension(
 'pg_tle_test',
 '0.1',
 'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

pgtle.install_update_path

Die `install_update_path`-Funktion stellt einen Aktualisierungspfad zwischen zwei verschiedenen Versionen einer TLE-Erweiterung bereit. Mit dieser Funktion können Benutzer Ihrer TLE-Erweiterung ihre Version mithilfe der `ALTER EXTENSION ... UPDATE`-Syntax aktualisieren.

Funktionsprototyp

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

Rolle

`pgtle_admin`

Argumente

- `name` – Der Name der TLE-Erweiterung. Dieser Wert wird beim Aufrufen von `CREATE EXTENSION` verwendet.
- `fromvers` – Die Quellversion der TLE-Erweiterung für das Upgrade.
- `tovers` – Die Zielversion der TLE-Erweiterung für das Upgrade.
- `ext` – Der Inhalt der Aktualisierung. Dieser Wert enthält Objekte wie Funktionen.

Output

Keine.

Verwendungsbeispiel

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

pgtle.register_feature

Die `register_feature`-Funktion fügt der `pgtle.feature_info`-Tabelle das angegebene interne PostgreSQL-Feature hinzu. PostgreSQL-Haken sind ein Beispiel für ein internes PostgreSQL-Feature. Das Trusted Language Extensions Development Kit unterstützt die Verwendung von PostgreSQL-Haken. Derzeit unterstützt diese Funktion das folgende Feature.

- `passcheck` – Registriert den Haken für die Passwortüberprüfung bei Ihrer Prozedur oder Funktion, die das Verhalten von PostgreSQL bei der Passwortüberprüfung anpasst.

Funktionsprototyp

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

Rolle

`pgtle_admin`

Argumente

- `proc` – Der Name einer gespeicherten Prozedur oder Funktion, die für das Feature verwendet werden soll.
- `feature` – Der Name des `pg_tle`-Features (z. B. `passcheck`), das für die Funktion registriert werden soll.

Output

Keine.

Verwendungsbeispiel

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

pgtle.register_feature_if_not_exists

Die `pgtle.register_feature_if_not_exists`-Funktion fügt der `pgtle.feature_info`-Tabelle das angegebene PostgreSQL-Feature hinzu und identifiziert die TLE-Erweiterung oder eine andere Prozedur oder Funktion, die das Feature verwendet. Weitere Informationen zu Haken und Trusted Language Extensions finden Sie unter Verwenden von PostgreSQL-Haken mit Ihren TLE-Erweiterungen.

Funktionsprototyp

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

Rolle

`pgtle_admin`

Argumente

- `proc` – Der Name einer gespeicherten Prozedur oder Funktion, die die Logik (Code) enthält, die als Feature für Ihre TLE-Erweiterung verwendet werden soll. Beispielsweise der `pw_hook`-Code.
- `feature` – Der Name des PostgreSQL-Features, das für die TLE-Funktion registriert werden soll. Derzeit ist der `passcheck`-Haken das einzige verfügbare Feature. Weitere Informationen finden Sie unter Haken zur Passwortüberprüfung (Passcheck).

Output

Gibt `true` zurück, nachdem das Feature für die angegebene Erweiterung registriert wurde. Gibt `false` zurück, wenn das Feature bereits registriert ist.

Verwendungsbeispiel

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

pgtle.set_default_version

Mit der `set_default_version`-Funktion können Sie eine `default_version` für Ihre TLE-Erweiterung angeben. Mit dieser Funktion können Sie einen Upgrade-Pfad definieren und die Version als Standard für Ihre TLE-Erweiterung festlegen. Wenn Datenbankbenutzer Ihre TLE-Erweiterung in den Befehlen `CREATE EXTENSION` und `ALTER EXTENSION ... UPDATE` angeben, wird diese Version Ihrer TLE-Erweiterung in der Datenbank für diesen Benutzer erstellt.

Diese Funktion gibt bei Erfolg `true` zurück. Wenn die im `name`-Argument angegebene TLE-Erweiterung nicht vorhanden ist, gibt die Funktion einen Fehler zurück. Entsprechend wird ein Fehler zurückgegeben, wenn die `version` der TLE-Erweiterung nicht existiert.

Funktionsprototyp

```
pgtle.set_default_version(name text, version text)
```

Rolle

`pgtle_admin`

Argumente

- `name` – Der Name der TLE-Erweiterung. Dieser Wert wird beim Aufrufen von `CREATE EXTENSION` verwendet.
- `version` – Die Version der TLE-Erweiterung, für die die Standardeinstellung festgelegt werden soll.

Output

- `true`– Wenn die Standardversion erfolgreich festgelegt wurde, gibt die Funktion `true` zurück.
- `ERROR` – Gibt eine Fehlermeldung zurück, wenn eine TLE-Erweiterung mit dem angegebenen Namen oder der angegebenen Version nicht existiert.

Verwendungsbeispiel

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

pgtle.uninstall_extension(name)

Die `uninstall_extension`-Funktion entfernt alle Versionen einer TLE-Erweiterung aus einer Datenbank. Diese Funktion verhindert, dass die TLE-Erweiterung durch künftige Aufrufe von `CREATE EXTENSION` installiert wird. Wenn die TLE-Erweiterung in der Datenbank nicht existiert, wird ein Fehler ausgelöst.

Die `uninstall_extension`-Funktion löscht die TLE-Erweiterung jedoch nicht, wenn diese derzeit in der Datenbank aktiv ist. Zum Löschen einer TLE-Erweiterung, die derzeit aktiv ist, müssen Sie explizit `DROP EXTENSION` aufrufen, um sie zu entfernen.

Funktionsprototyp

```
pgtle.uninstall_extension(extname text)
```

Rolle

`pgtle_admin`

Argumente

- `extname` – Der Name der zu deinstallierenden TLE-Erweiterung. Dieser Name ist derselbe, der mit `CREATE EXTENSION` verwendet wurde, um die TLE-Erweiterung zur Verwendung in einer bestimmten Datenbank zu laden.

Output

Keine.

Verwendungsbeispiel

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

pgtle.uninstall_extension(name, version)

Die `uninstall_extension(name, version)`-Funktion entfernt die angegebene Version der TLE-Erweiterung aus der Datenbank. Diese Funktion verhindert, dass `CREATE EXTENSION` und `ALTER EXTENSION` eine TLE-Erweiterung installieren oder auf die angegebene Version aktualisieren. Diese Funktion entfernt außerdem alle Aktualisierungspfade für die angegebene Version der TLE-Erweiterung. Diese Funktion deinstalliert die TLE-Erweiterung jedoch nicht, wenn diese derzeit in der Datenbank aktiv ist. Sie müssen explizit `DROP EXTENSION` aufrufen, um die TLE-Erweiterung zu entfernen. Informationen zur Deinstallation aller Versionen einer TLE-Erweiterung finden Sie unter pgtle.uninstall_extension(name).

Funktionsprototyp

```
pgtle.uninstall_extension(extname text, version text)
```

Rolle

`pgtle_admin`

Argumente

- `extname` – Der Name der TLE-Erweiterung. Dieser Wert wird beim Aufrufen von `CREATE EXTENSION` verwendet.
- `version` – Die Version der TLE-Erweiterung, die in der Datenbank deinstalliert werden soll.

Output

Keine.

Verwendungsbeispiel

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

pgtle.uninstall_extension_if_exists

Die `uninstall_extension_if_exists`-Funktion entfernt alle Versionen einer TLE-Erweiterung aus einer bestimmten Datenbank. Wenn die TLE-Erweiterung nicht existiert, bleibt die Funktion im Hintergrund (es wird keine Fehlermeldung ausgegeben). Wenn die angegebene Erweiterung derzeit in einer Datenbank aktiv ist, wird sie von dieser Funktion nicht gelöscht. Sie müssen `DROP EXTENSION` explizit aufrufen, um die TLE-Erweiterung zu entfernen, bevor Sie diese Funktion verwenden, um ihre Artefakte zu deinstallieren.

Funktionsprototyp

```
pgtle.uninstall_extension_if_exists(extname text)
```

Rolle

`pgtle_admin`

Argumente

- `extname` – Der Name der TLE-Erweiterung. Dieser Wert wird beim Aufrufen von `CREATE EXTENSION` verwendet.

Output

Die `uninstall_extension_if_exists`-Funktion gibt `true` nach der Deinstallation der angegebenen Erweiterung zurück. Wenn die angegebene Erweiterung nicht vorhanden ist, gibt die Funktion `false` zurück.

- `true` – Gibt `true` nach der Deinstallation der TLE-Erweiterung zurück.
- `false` – Gibt `false` zurück, wenn die TLE-Erweiterung in der Datenbank nicht existiert.

Verwendungsbeispiel

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

pgtle.uninstall_update_path

Die `uninstall_update_path`-Funktion entfernt den angegebenen Aktualisierungspfad einer TLE-Erweiterung. Dadurch wird verhindert, dass `ALTER EXTENSION ... UPDATE TO` diesen Pfad als Aktualisierungspfad verwendet.

Wenn die TLE-Erweiterung derzeit von einer der Versionen in diesem Aktualisierungspfad verwendet wird, verbleibt sie in der Datenbank.

Wenn der angegebene Aktualisierungspfad nicht vorhanden ist, gibt diese Funktion einen Fehler aus.

Funktionsprototyp

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

Rolle

`pgtle_admin`

Argumente

- `extname` – Der Name der TLE-Erweiterung. Dieser Wert wird beim Aufrufen von `CREATE EXTENSION` verwendet.
- `fromvers` – Die Quellversion der TLE-Erweiterung, die im Aktualisierungspfad verwendet wird.
- `tovers` – Die Zielversion der TLE-Erweiterung, die im Aktualisierungspfad verwendet wird.

Output

Keine.

Verwendungsbeispiel

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

pgtle.uninstall_update_path_if_exists

Die `uninstall_update_path_if_exists`-Funktion ähnelt `uninstall_update_path` insofern, als sie den angegebenen Aktualisierungspfad aus einer TLE-Erweiterung entfernt. Wenn der Aktualisierungspfad jedoch nicht existiert, löst diese Funktion keine Fehlermeldung aus. Stattdessen gibt die Funktion `false` zurück.

Funktionsprototyp

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

Rolle

`pgtle_admin`

Argumente

- `extname` – Der Name der TLE-Erweiterung. Dieser Wert wird beim Aufrufen von `CREATE EXTENSION` verwendet.
- `fromvers` – Die Quellversion der TLE-Erweiterung, die im Aktualisierungspfad verwendet wird.
- `tovers` – Die Zielversion der TLE-Erweiterung, die im Aktualisierungspfad verwendet wird.

Output

- `true` – Die Funktion hat den Pfad für die TLE-Erweiterung erfolgreich aktualisiert.
- `false` – Die Funktion konnte den Pfad für die TLE-Erweiterung nicht aktualisieren.

Verwendungsbeispiel

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

pgtle.unregister_feature

Die `unregister_feature`-Funktion bietet eine Möglichkeit, Funktionen zu entfernen, die für die Verwendung von `pg_tle`-Features wurden, wie z. B. Haken. Weitere Informationen zur Registrierung eines Features erhalten Sie unter pgtle.register_feature.

Funktionsprototyp

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

Rolle

`pgtle_admin`

Argumente

- `proc` – Der Name einer gespeicherten Funktion, für die ein `pg_tle`-Feature registriert werden soll.
- `feature` – Der Name des `pg_tle`-Features, das für die Funktion registriert werden soll. Beispielsweise ist `passcheck` ein Feature, das für die Verwendung durch die vertrauenswürdigen Spracherweiterungen, die Sie entwickeln, registriert werden kann. Weitere Informationen finden Sie unter Haken zur Passwortüberprüfung (Passcheck).

Output

Keine.

Verwendungsbeispiel

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

pgtle.unregister_feature_if_exists

Die `unregister_feature`-Funktion bietet eine Möglichkeit, Funktionen zu entfernen, die für die Verwendung von `pg_tle`-Features wurden, wie z. B. Haken. Weitere Informationen finden Sie unter Verwenden von PostgreSQL-Haken mit Ihren TLE-Erweiterungen. Gibt `true` zurück, nachdem die Registrierung der Funktion erfolgreich aufgehoben wurde. Gibt `false` zurück, wenn das Feature nicht registriert wurde.

Informationen zur Registrierung von `pg_tle`-Features für Ihre TLE-Erweiterungen finden Sie unter pgtle.register_feature.

Funktionsprototyp

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

Rolle

`pgtle_admin`

Argumente

- `proc` – Der Name der gespeicherten Funktion, für die ein `pg_tle`-Feature registriert wurde.
- `feature` – Der Name des `pg_tle`-Features, das mit der vertrauenswürdigen Spracherweiterung registriert wurde.

Output

Gibt `true` oder `false` wie folgt zurück.

- `true` – Die Funktion hat die Registrierung des Features in der Erweiterung erfolgreich aufgehoben.
- `false` – Die Funktion konnte die Registrierung des Features in der TLE-Erweiterung nicht aufheben.

Verwendungsbeispiel

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

Hakenreferenz für Trusted Language Extensions für PostgreSQL

Trusted Language Extensions für PostgreSQL unterstützt PostgreSQL-Haken. Ein Haken ist ein interner Callback-Mechanismus, der Entwicklern zur Erweiterung der Kernfunktionalität von PostgreSQL zur Verfügung steht. Durch die Verwendung von Haken können Entwickler ihre eigenen Funktionen oder Verfahren zur Verwendung bei verschiedenen Datenbankoperationen implementieren und so das Verhalten von PostgreSQL in gewisser Weise ändern. Sie können beispielsweise einen `passcheck`-Haken verwenden, um anzupassen, wie PostgreSQL die Passwörter behandelt, die beim Erstellen oder Ändern von Passwörtern für Benutzer (Rollen) angegeben werden.

In der folgenden Dokumentation erfahren Sie mehr über die Passcheck-Hooks, die für Ihre TLE-Erweiterungen verfügbar sind. Weitere Informationen zu den verfügbaren Hooks, einschließlich des Client-Authentifizierungs-Hooks, finden Sie unter Trusted Language Extensions-Hooks.

Haken zur Passwortüberprüfung (Passcheck)

Der `passcheck`-Haken wird verwendet, um das PostgreSQL-Verhalten während der Passwortüberprüfung für die folgenden SQL-Befehle und `psql`-Metabefehle anzupassen.

- `CREATE ROLE` *`username`* `...PASSWORD` – Weitere Informationen finden Sie im Abschnitt CREATE ROLE der PostgreSQL-Dokumentation.
- `ALTER ROLE` *`username`*`...PASSWORD` – Weitere Informationen finden Sie im Abschnitt ALTER ROLE der PostgreSQL-Dokumentation.
- `\password` *`username`* – Dieser interaktive `psql`-Metabefehl ändert das Passwort für den angegebenen Benutzer auf sichere Weise, indem er das Passwort hasht, bevor die `ALTER ROLE ... PASSWORD`-Syntax transparent verwendet wird. Der Metabefehl ist ein sicherer Wrapper für den `ALTER ROLE ... PASSWORD`-Befehl, daher gilt der Haken für das Verhalten des`psql`-Metabefehls.

Ein Beispiel finden Sie unter Liste der Hakencodes für die Passwortüberprüfung.

Inhalt

- Funktionsprototyp
- Argumente
- Konfiguration
- Nutzungshinweise

Funktionsprototyp

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

Argumente

Eine `passcheck`-Hakenfunktion verwendet die folgenden Argumente.

- `username` – Der Name (als Text) der Rolle (Benutzername), die ein Passwort festlegt.
- `password` – Das Klartext- oder Hash-Passwort. Das eingegebene Passwort sollte dem im `password_type` angegebenen Typ entsprechen.

- `password_type` – Geben Sie das `pgtle.password_type`-Format des Passworts an. Dieses Format kann einer der folgenden Optionen entsprechen.
  - `PASSWORD_TYPE_PLAINTEXT` – Ein Klartext-Passwort.
  - `PASSWORD_TYPE_MD5` – Ein Passwort, das mit dem MD5-Algorithmus (Message Digest 5) gehasht wurde.
  - `PASSWORD_TYPE_SCRAM_SHA_256` – Ein Passwort, das mit dem SCRAM-SHA-256-Algorithmus gehasht wurde.
- `valid_until` – Geben Sie den Zeitpunkt an, am dem das Passwort ungültig wird. Dieses Argument ist optional. Wenn Sie dieses Argument verwenden, geben Sie die Uhrzeit als `timestamptz`-Wert an.
- `valid_null` – Wenn dieser boolesche Wert auf `true` festgelegt ist, wird die `valid_until`-Option auf `NULL` eingestellt.

Konfiguration

Die Funktion `pgtle.enable_password_check` steuert, ob der Passcheck-Haken aktiv ist. Der Passcheck-Haken hat drei mögliche Einstellungen.

- `off` – Schaltet den `passcheck`-Haken für die Passwortüberprüfung aus. Dies ist der Standardwert.
- on – Aktiviert den `passcode`-Haken für die Passwortüberprüfung, sodass Passwörter mit der Tabelle verglichen werden.
- `require` – Erfordert, dass ein Passwortüberprüfungshaken definiert wird.

Nutzungshinweise

Um den `passcheck`-Haken ein- oder auszuschalten, müssen Sie die benutzerdefinierte DB-Parametergruppe für die Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL-DB-Clusters ändern.

Für Linux, macOS oder Unix:

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
    --db-parameter-group-name your-custom-parameter-group ^
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Amazon-Aurora-PostgreSQL-Referenz

In den folgenden Themen finden Sie Informationen über Kollationen, Funktionen, Parameter und Warteereignisse in Amazon Aurora PostgreSQL.

Themen

- Aurora-PostgreSQL-Kollationen für EBCDIC- und andere Mainframe-Migrationen
- In Aurora PostgreSQL unterstützte Sortierungen
- Aurora-PostgreSQL-Funktionsreferenz
- Amazon-Aurora-PostgreSQL-Parameter
- Amazon-Aurora-PostgreSQL-Warteereignisse

Aurora-PostgreSQL-Kollationen für EBCDIC- und andere Mainframe-Migrationen

Migration von Mainframe-Anwendungen auf neue Plattformen wie AWS bewahrt idealerweise das Anwendungsverhalten. Damit das Anwendungsverhalten auf einer neuen Plattform genauso erhalten bleibt, wie es auf dem Mainframe war, müssen migrierte Daten mit denselben Kollations- und Sortierregeln zusammengetragen werden. Beispielsweise verschieben viele Db2-Migrationslösungen Nullwerte auf u0180 (Unicode-Position 0180), sodass diese Sortierungen zuerst u0180 sortieren. Dies ist ein Beispiel dafür, wie Kollationen von ihrer Mainframe-Quelle abweichen können und warum es notwendig ist, eine Kollation zu wählen, die besser der ursprünglichen EBCDIC-Kollation entspricht.

Aurora PostgreSQL 14.3 und höhere Versionen bieten viele ICU- und EBCDIC-Kollationen, um eine solche Migration nach AWS unter Verwendung der AWS-Mainframe-Modernisierung zu unterstützen. Weitere Informationen zu diesem Service finden Sie unter Was ist AWS Mainframe-Modernisierung?

In der folgenden Tabelle finden Sie von Aurora PostgreSQL bereitgestellte Sortierungen. Diese Sortierungen folgen den EBCDIC-Regeln und stellen sicher, dass Mainframe-Anwendungen auf dieselbe Weise auf AWS funktionieren, wie sie es in der Mainframe-Umgebung getan haben. Der Kollationsname beinhaltet die entsprechende Codepage, (cpnnnn), sodass Sie die passende Sortierung für Ihre Mainframe-Quelle auswählen können. Verwenden Sie z. B. `en-US-cp037-x-icu`, um das Kollationsverhalten für EBCDIC-Daten zu erreichen, die von einer Mainframe-Anwendung stammen, die Codepage 037 verwendete.

EBCDIC-Kollationen	AWS Blue Age	AWS Micro-Focus-Kollationen
da-DK-cp1142-x-icu	da-DK-cp1142b-x-icu	da-DK-cp1142m-x-icu
da-DK-cp277-x-icu	da-DK-cp277b-x-icu	–
de-DE-cp1141-x-icu	de-DE-cp1141b-x-icu	de-DE-cp1141m-x-icu
de-DE-cp273-x-icu	de-DE-cp273b-x-icu	–
en-GB-cp1146-x-icu	en-GB-cp1146b-x-icu	en-GB-cp1146m-x-icu
en-GB-cp285-x-icu	en-GB-cp285b-x-icu	–
en-US-cp037-x-icu	en-US-cp037b-x-icu	–
en-US-cp1140-x-icu	en-US-cp1140b-x-icu	en-US-cp1140m-x-icu
es-ES-cp1145-x-icu	es-ES-cp1145b-x-icu	es-ES-cp1145m-x-icu
es-ES-cp284-x-icu	es-ES-cp284b-x-icu	–
fi-FI-cp1143-x-icu	fi-FI-cp1143b-x-icu	fi-FI-cp1143m-x-icu
fi-FI-cp278-x-icu	fi-FI-cp278b-x-icu	–
fr-FR-cp1147-x-icu	fr-FR-cp1147b-x-icu	fr-FR-cp1147m-x-icu
fr-FR-cp297-x-icu	fr-FR-cp297b-x-icu	–
it-IT-cp1144-x-icu	it-IT-cp1144b-x-icu	it-IT-cp1144m-x-icu
it-IT-cp280-x-icu	it-IT-cp280b-x-icu	–
nl-BE-cp1148-x-icu	nl-BE-cp1148b-x-icu	nl-BE-cp1148m-x-icu
nl-BE-cp500-x-icu	nl-BE-cp500b-x-icu	–

Weitere Informationen über AWS Blu Age finden Sie im Tutorial: Verwaltete Runtime für AWS Blu Age im AWSMainframe-Modernisierung-Benutzerhandbuch.

Weitere Informationen zur Arbeit mit AWS Mikrofokus finden Sie im Tutorial: Verwaltete Runtime für Micro Focus im AWS-Mainframe-Modernisierung-Benutzerhandbuch.

Weitere Informationen zu PostgreSQL und Kollationen finden Sie unter Support für Kollationen in der PostgreSQL-Dokumentation.

In Aurora PostgreSQL unterstützte Sortierungen

Sortierungen sind eine Reihe von Regeln, die bestimmen, wie in der Datenbank gespeicherte Zeichenfolgen sortiert und verglichen werden. Sortierungen spielen eine grundlegende Rolle im Computersystem und sind Teil des Betriebssystems. Sortierungen ändern sich im Laufe der Zeit, wenn neue Zeichen zu Sprachen hinzugefügt werden oder wenn sich die Sortierregeln ändern.

Sortierungsbibliotheken definieren spezifische Regeln und Algorithmen für eine Sortierung. Die beliebtesten Sortierungsbibliotheken, die in PostgreSQL verwendet werden, sind GNU C (glibc) und Internationalization Components for Unicode (ICU). Standardmäßig verwendet Aurora PostgreSQL die Glibc-Sortierung, die Unicode-Zeichensortierreihenfolgen für Multibyte-Zeichensequenzen enthält.

Wenn Sie einen neuen DB-Cluster von Aurora PostgreSQL erstellen, wird das Betriebssystem auf die verfügbare Sortierung überprüft. Die PostgreSQL-Parameter des `CREATE DATABASE`-Befehls `LC_COLLATE` und `LC_CTYPE` werden verwendet, um eine Sortierung anzugeben, die in dieser Datenbank als Standardsortierung gilt. Alternativ können Sie auch den Parameter `LOCALE` in `CREATE DATABASE` verwenden, um diese Parameter festzulegen. Dieser bestimmt die Standardsortierung für Zeichenfolgen in der Datenbank und die Regeln für die Klassifizierung von Zeichen als Buchstaben, Zahlen oder Symbole. Sie können auch eine Sortierung auswählen, die für eine Spalte, einen Index oder eine Abfrage verwendet werden soll.

Aurora PostgreSQL benötigt für die Sortierungsunterstützung die Glibc-Bibliothek im Betriebssystem. Die Instance von Aurora PostgreSQL wird regelmäßig mit den neuesten Versionen des Betriebssystems aktualisiert. Diese Updates umfassen manchmal eine neuere Version der Glibc-Bibliothek. In seltenen Fällen ändern neuere Versionen von Glibc die Sortierreihenfolge oder Sortierung einiger Zeichen, was dazu führen kann, dass die Daten anders sortiert werden oder ungültige Indexeinträge entstehen. Wenn Sie während eines Updates bei der Sortierung Probleme mit der Sortierreihenfolge feststellen, müssen Sie möglicherweise die Indizes neu erstellen.

Damit mögliche Auswirkungen der Glibc-Updates reduziert werden, enthält Aurora PostgreSQL jetzt eine unabhängige Standard-Sortierungsbibliothek. Diese Sortierungsbibliothek ist in Aurora PostgreSQL 14.6, 13.9, 12.13, 11.18 und neueren Nebenversionen verfügbar. Sie ist mit Glibc

2.26-59.amzn2 kompatibel und bietet eine stabile Sortierreihenfolge, um falsche Abfrageergebnisse zu verhindern.

Aurora-PostgreSQL-Funktionsreferenz

Nachfolgend finden Sie eine Liste der Aurora PostgreSQL-Funktionen, die für Ihre Aurora-DB-Cluster verfügbar sind, welche die Aurora PostgreSQL-kompatible Edition-DB-Engine ausführen. Diese Aurora PostgreSQL-Funktionen sind zusätzlich zu den Standard-PostgreSQL-Funktionen. Weitere Informationen zu PostgreSQL-Standardfunktionen finden Sie unter PostgreSQL – Funktionen und Operatoren.

Übersicht

Sie können die folgenden Funktionen für Amazon-RDS-DB-Instances verwenden, auf denen Aurora PostgreSQL ausgeführt wird:

- aurora_db_instance_identifier
- aurora_ccm_status
- aurora_global_db_instance_status
- aurora_global_db_status
- aurora_list_builtins
- aurora_replica_status
- aurora_stat_activity
- aurora_stat_backend_waits
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans
- aurora_stat_reset_wal_cache
- aurora_stat_statements

- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

aurora_db_instance_identifier

Meldet den Namen der DB-Instance, mit der Sie verbunden sind.

Syntax

```
aurora_db_instance_identifier()
```

Argumente

Keine

Rückgabetyp

VARCHAR-Zeichenfolge

Nutzungshinweise

Diese Funktion zeigt den Namen der DB-Instance des Aurora-PostgreSQL-kompatiblen Edition-Clusters für Ihre Datenbank-Client- oder Anwendungsverbindung an.

Diese Funktion ist ab der Veröffentlichung der PostgreSQL-Versionen 13.7, 12.11, 11.16 und 10.21 und für alle späteren Versionen verfügbar.

Beispiele

Im folgenden Beispiel sehen Sie die Ergebnisse vom Aufruf der Funktion `aurora_db_instance_identifier`.

```
=> SELECT aurora_db_instance_identifier();
```

```
aurora_db_instance_identifier
------------------------------
 test-my-instance-name
```

Sie können die Ergebnisse dieser Funktion mit der `aurora_replica_status`-Funktion verbinden, um Details über die DB-Instance für Ihre Verbindung abzurufen. Die aurora_replica_status allein zeigt Ihnen nicht, welche DB-Instance Sie verwenden. Im folgenden Beispiel wird gezeigt, wie Sie dies tun.

```
=> SELECT *
    FROM aurora_replica_status() rt,
         aurora_db_instance_identifier() di
    WHERE rt.server_id = di;
-[ RECORD 1 ]----------------------+-----------------------
server_id                         | test-my-instance-name
session_id                        | MASTER_SESSION_ID
durable_lsn                       | 88492069
highest_lsn_rcvd                  |
current_read_lsn                  |
cur_replay_latency_in_usec        |
active_txns                       |
is_current                        | t
last_transport_error              | 0
last_error_timestamp              |
last_update_timestamp             | 2022-06-03 11:18:25+00
feedback_xmin                     |
feedback_epoch                    |
replica_lag_in_msec               |
log_stream_speed_in_kib_per_second | 0
log_buffer_sequence_number        | 0
oldest_read_view_trx_id           |
oldest_read_view_lsn              |
pending_read_ios                  | 819
```

aurora_ccm_status

Zeigt den Status des Cluster-Cache-Managers an.

Syntax

```
aurora_ccm_status()
```

Argumente

Keine.

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `buffers_sent_last_minute` – Die Anzahl der Puffer, die in der letzten Minute an den designierten Reader gesendet wurden.
- `buffers_found_last_minute` – Die Anzahl der Puffer, auf die in der letzten Minute häufig zugegriffen wurde.
- `buffers_sent_last_scan` – Die Anzahl der Puffer, die während des letzten vollständigen Scanvorgangs des Puffercaches an den designierten Reader gesendet wurden.
- `buffers_found_last_scan` – Die Anzahl der Puffer, auf die während des letzten vollständigen Scanvorgangs des Puffercaches häufig zugegriffen wurde. Puffer, die bereits im Cache des designierten Readers abgelegt wurden, werden nicht gesendet.
- `buffers_sent_current_scan` – Die Anzahl der Puffer, die während des aktuellen Scanvorgangs gesendet wurden.
- `buffers_found_current_scan` – Die Anzahl der häufig aufgerufenen Puffer, die im aktuellen Scan identifiziert wurden.
- `current_scan_progress` – Die Anzahl der Puffer, die während des aktuellen Scanvorgangs aufgerufen wurden.

Nutzungshinweise

Sie können diese Funktion verwenden, um die Funktion zur Cluster-Cache-Verwaltung (CCM) zu überprüfen und zu überwachen. Diese Funktion ist nur verfügbar, wenn CCM auf Ihrem Aurora-PostgreSQL-DB-Cluster aktiv ist. Zum Verwenden dieser Funktion stellen Sie eine Verbindung mit der Write-DB-Instance Ihres Aurora-PostgreSQL-DB-Clusters her.

Sie aktivieren CCM für einen Aurora-PostgreSQL-DB-Cluster, indem Sie den Wert `apg_ccm_enabled` in der benutzerdefinierten DB-Cluster-Parametergruppe des Clusters auf 1 festlegen. Um zu erfahren wie dies geht, vgl. Konfigurieren der Cluster-Cache-Verwaltung.

Die Cluster-Cache-Verwaltung ist auf einem Aurora-PostgreSQL-DB-Cluster aktiv, wenn der Cluster über eine Aurora-Reader-Instance verfügt, die wie folgt konfiguriert ist:

- Die Aurora-Reader-Instance verwendet den gleichen Typ und die gleiche Größe der DB-Instance-Klasse wie die Writer-Instance des Clusters.
- Die Aurora-Reader-Instance ist für den Cluster als Tier-0 konfiguriert. Wenn der Cluster über mehr als einen Reader verfügt, ist dies der einzige Tier-0-Reader.

Wenn Sie mehr als einen Reader auf Tier-0 einstellen, wird CCM deaktiviert. Wenn CCM deaktiviert ist, wird beim Aufruf dieser Funktion die folgende Fehlermeldung zurückgegeben:

```
ERROR: Cluster Cache Manager is disabled
```

Sie können auch die PostgreSQL-Erweiterung pg_buffercache verwenden, um den Puffercache zu analysieren. Weitere Informationen finden Sie unter pg_buffercache in der PostgreSQL-Dokumentation.

Weitere Informationen finden Sie unter Einführung in die Aurora-PostgreSQL-Cluster-Cache-Verwaltung.

Beispiele

Im folgenden Beispiel sehen Sie die Ergebnisse vom Aufruf der Funktion `aurora_ccm_status`. Dieses erste Beispiel zeigt CCM-Statistiken.

```
=> SELECT * FROM aurora_ccm_status();
 buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
 buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
 current_scan_progress
--------------------------+---------------------------+------------------------
+-------------------------+---------------------------+---------------------------
+------------------------
                  2242000 |                   2242003 |               17920442 |
         17923410 |                  14098000 |                   14100964 |
   15877443
```

Für ausführlichere Details können Sie die erweiterte Anzeige verwenden, wie im Folgenden gezeigt:

```
\x
Expanded display is on.
SELECT *  FROM aurora_ccm_status();
[ RECORD 1 ]----------------------+---------
buffers_sent_last_minute          | 2242000
```

```
buffers_found_last_minute          | 2242003
buffers_sent_last_scan             | 17920442
buffers_found_last_scan            | 17923410
buffers_sent_current_scan          | 14098000
buffers_found_current_scan         | 14100964
current_scan_progress              | 15877443
```

Dieses Beispiel zeigt, wie die aktive Rate und der aktive Prozentsatz überprüft werden.

```
=> SELECT buffers_sent_last_minute * 8/60 AS warm_rate_kbps,
100 * (1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
FROM aurora_ccm_status ();
 warm_rate_kbps | warm_percent
----------------+--------------
 16523 |        100.0
```

aurora_global_db_instance_status

Zeigt den Status aller Aurora-Instances an, einschließlich Replikaten in einem globalen Aurora-DB-Cluster.

Syntax

```
aurora_global_db_instance_status()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `server_id` – Die ID der DB-Instance.
- `session_id` – Eindeutiger Bezeichner für die aktuelle Sitzung. Der Wert `MASTER_SESSION_ID` bezeichnet die (primäre) Writer-DB-Instance.
- `aws_region` – Die AWS-Region, in der diese globale DB-Instance ausgeführt wird. Eine Liste der Regionen finden Sie unter Verfügbarkeit in Regionen.
- `durable_lsn` – Die Log-Sequenznummer (LSN), die dauerhaft gespeichert wurde. Eine Log-Sequenznummer (LSN) ist eine eindeutige fortlaufende Nummer, die einen Datensatz im

Datenbank-Transaktionsprotokoll identifiziert. LSNs werden so angeordnet, dass eine größere LSN eine spätere Transaktion darstellt.

- `highest_lsn_rcvd` – Die höchste LSN, die die DB-Instance von der Writer-DB-Instance empfangen hat.
- `feedback_epoch` – Die Epoche, die die DB-Instance beim Erzeugen der Hot-Standby-Informationen verwendet. Ein Hot Standby ist eine DB-Instance, die Verbindungen und Abfragen unterstützt, während sich die primäre DB im Wiederherstellungs- oder Standby-Modus befindet. Die Hot-Standby-Informationen umfassen die Epoche (bestimmter Zeitpunkt) und andere Details über die DB-Instance, die als Hot-Standby verwendet wird. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation zu Hot Standby.
- `feedback_xmin` – Die minimale (älteste) aktive Transaktions-ID, die von der DB-Instance verwendet wird.
- `oldest_read_view_lsn` – Die älteste LSN, die von der DB-Instance zum Lesen aus dem Speicher verwendet wird.
- `visibility_lag_in_msec` – Gibt an (in Millisekunden), wie weit die DB-Instance hinter der Writer-DB-Instance zurückgeblieben ist.

Nutzungshinweise

Diese Funktion gibt Aufschluss über Replikationsstatistiken für einen Aurora-DB-Cluster. Für jede Aurora-PostgreSQL-DB-Instance im Cluster zeigt die Funktion eine Datenreihe an, die regionsübergreifende Replikate in einer globalen Datenbankkonfiguration enthält.

Sie können diese Funktion von jeder Instance in einem Aurora-PostgreSQL-DB-Cluster oder einer globalen Aurora-PostgreSQL-Datenbank ausführen. Die Funktion gibt Details zur Verzögerung für alle Replikat-Instances zurück.

Weitere Informationen über die Überwachung der Verzögerung mithilfe dieser Funktion (`aurora_global_db_instance_status`) oder mit `aurora_global_db_status` finden Sie unter Überwachen von Aurora-PostgreSQL-basierten globalen Datenbanken.

Informationen über globale Aurora-Datenbanken finden Sie unter Übersicht über Amazon Aurora Global Database.

Informationen zu den ersten Schritten mit globalen Aurora-Datenbanken finden Sie unter Erste Schritte mit Amazon Aurora Global Database oder Häufig gestellte Fragen zu Amazon Aurora.

Beispiele

Dieses Beispiel zeigt regionsübergreifende Instance-Statistiken.

```
=> SELECT *
  FROM aurora_global_db_instance_status();
               server_id                |              session_id              |
  aws_region  | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------+--------------------------------------
+-------------+-------------+------------------+----------------+---------------
+----------------------+------------------------
 db-119-001-instance-01                  | MASTER_SESSION_ID                    | eu-
west-1    |  2534560273 |           [NULL] |         [NULL] |        [NULL] |
     [NULL] |                [NULL]
 db-119-001-instance-02                  | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                    6
 db-119-001-instance-03                  | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                    6
 db-119-001-instance-04                  | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                   23
 db-119-001-instance-05                  | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                   23
 db-119-001-global-instance-1            | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                  996
 db-119-001-global-instance-1            | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                   14
 db-119-001-global-instance-1-eu-west-2a  | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                    0
```

Dieses Beispiel veranschaulicht, wie die globale Replikatverzögerung in Millisekunden überprüft wird.

```
=> SELECT CASE
          WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
         ELSE 'Secondary'
      END AS global_role,
```

```
      aws_region,
      server_id,
      visibility_lag_in_msec
  FROM aurora_global_db_instance_status()
  ORDER BY 1, 2, 3;
   global_role |  aws_region  |               server_id                |
 visibility_lag_in_msec
-------------+--------------+----------------------------------------
+------------------------
 Primary     | eu-west-1    | db-119-001-instance-01                 |
 [NULL]
 Secondary   | eu-central-1 | db-119-001-global-instance-1           |
     13
 Secondary   | eu-west-1    | db-119-001-instance-02                 |
     10
 Secondary   | eu-west-1    | db-119-001-instance-03                 |
      9
 Secondary   | eu-west-1    | db-119-001-instance-04                 |
      2
 Secondary   | eu-west-1    | db-119-001-instance-05                 |
     18
 Secondary   | eu-west-2    | db-119-001-global-instance-1           |
     14
 Secondary   | eu-west-2    | db-119-001-global-instance-1-eu-west-2a |
     13
```

Dieses Beispiel zeigt, wie die minimale, maximale und durchschnittliche Verzögerung je AWS-Region im Hinblick auf die globale Datenbankkonfiguration überprüft wird.

```
=> SELECT 'Secondary' global_role,
       aws_region,
       min(visibility_lag_in_msec) min_lag_in_msec,
       max(visibility_lag_in_msec) max_lag_in_msec,
       round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
  FROM aurora_global_db_instance_status()
 WHERE aws_region NOT IN (SELECT   aws_region
                            FROM aurora_global_db_instance_status()
                           WHERE session_id='MASTER_SESSION_ID')
                        GROUP BY aws_region
UNION ALL
SELECT  'Primary' global_role,
        aws_region,
        NULL,
```

```
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
   WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
 global_role |  aws_region  | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
-------------+--------------+-----------------+-----------------+-----------------
 Primary     | eu-west-1    |          [NULL] |          [NULL] |          [NULL]
 Secondary   | eu-central-1 |             133 |             133 |             133
 Secondary   | eu-west-2    |               0 |             495 |             248
```

aurora_global_db_status

Zeigt Informationen über verschiedene Aspekte der globalen Aurora-Datenbankverzögerung an, insbesondere die Verzögerung des zugrunde liegenden Aurora-Speichers (sogenannte Haltbarkeitsverzögerung) und die Verzögerung zwischen dem Recovery Point Objective (RPO).

Syntax

```
aurora_global_db_status()
```

Argumente

Keine.

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `aws_region` – Die AWS-Region, in der sich dieser DB-Cluster befindet. Eine vollständige Auflistung von AWS-Regionen nach Engine finden Sie unter Regionen und Availability Zones.
- `highest_lsn_written` – Die höchste Log-Sequenznummer (LSN), die derzeit auf diesem DB-Cluster vorhanden ist. Eine Log-Sequenznummer (LSN) ist eine eindeutige fortlaufende Nummer, die einen Datensatz im Datenbank-Transaktionsprotokoll identifiziert. LSNs werden so angeordnet, dass eine größere LSN eine spätere Transaktion darstellt.
- `durability_lag_in_msec` – Die Differenz zwischen den Zeitstempelwerten der `highest_lsn_written` auf einem sekundären DB-Cluster und der `highest_lsn_written` auf dem primären DB-Cluster. Der Wert -1 identifiziert den primären DB-Cluster der globalen Aurora-Datenbank.

- `rpo_lag_in_msec` – Die RPO-Verzögerung (Recovery Point Objective). Die RPO-Verzögerung ist die Zeit, die benötigt wird, bis die letzte Benutzertransaktion COMMIT auf einem sekundären DB-Cluster gespeichert wird, nachdem sie auf dem primären DB-Cluster einer globalen Aurora-Datenbank abgelegt wurde. Der Wert -1 bezeichnet den primären DB-Cluster (die Verzögerung ist daher nicht relevant).

  Einfach ausgedrückt berechnet diese Metrik das Recovery Point Objective für jeden Aurora-PostgreSQL-DB-Cluster in der globalen Aurora-Datenbank, d. h., wie viele Daten bei einem Ausfall verloren gehen könnten. Wie die Verzögerung wird auch RPO zeitlich gemessen.
- `last_lag_calculation_time` – Der Zeitstempel, der angibt, wann die Werte für `durability_lag_in_msec` und `rpo_lag_in_msec` zuletzt berechnet wurden. Ein Zeitwert wie `1970-01-01 00:00:00+00` bedeutet, dass dies der primäre DB-Cluster ist.
- `feedback_epoch` – Die Epoche, die der sekundäre DB-Cluster beim Erzeugen von Hot-Standby-Informationen verwendet. Ein Hot Standby ist eine DB-Instance, die Verbindungen und Abfragen unterstützt, während sich die primäre DB im Wiederherstellungs- oder Standby-Modus befindet. Die Hot-Standby-Informationen umfassen die Epoche (bestimmter Zeitpunkt) und andere Details über die DB-Instance, die als Hot-Standby verwendet wird. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation zu Hot Standby.
- `feedback_xmin` – Die minimale (älteste) aktive Transaktions-ID, die von einem sekundären DB-Cluster verwendet wird.

Nutzungshinweise

Alle derzeit verfügbaren Aurora-PostgreSQL-Versionen unterstützen diese Funktion. Diese Funktion zeigt Replikationsstatistiken für eine globale Aurora-Datenbank. Sie zeigt eine Zeile für jeden DB-Cluster in einer globalen Aurora-PostgreSQL-Datenbank. Sie können diese Funktion von jeder Instance in Ihrer globalen Aurora-PostgreSQL-Datenbank ausführen.

Informationen zur Auswertung der Replikationsverzögerung der globalen Aurora-Datenbank, die die sichtbare Datenverzögerung darstellt, finden Sie unter aurora_global_db_instance_status.

Weitere Informationen über die Verwendung von `aurora_global_db_status` und `aurora_global_db_instance_status` zur Überwachung der globalen Aurora-Datenbankverzögerung finden Sie unter Überwachen von Aurora-PostgreSQL-basierten globalen Datenbanken. Informationen über globale Aurora-Datenbanken finden Sie unter Übersicht über Amazon Aurora Global Database.

Beispiele

Dieses Beispiel zeigt, wie regionsübergreifende Speicherstatistiken angezeigt werden.

```
=> SELECT CASE
          WHEN '-1' = durability_lag_in_msec THEN 'Primary'
          ELSE 'Secondary'
       END AS global_role,
       *
  FROM aurora_global_db_status();
 global_role | aws_region | highest_lsn_written | durability_lag_in_msec |
 rpo_lag_in_msec | last_lag_calculation_time  | feedback_epoch | feedback_xmin
-------------+------------+---------------------+------------------------
+-----------------+----------------------------+----------------+--------------
 Primary     | eu-west-1  |           131031557 |                     -1 |
 -1 | 1970-01-01 00:00:00+00     |             0 |              0
 Secondary   | eu-west-2  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
 Secondary   | eu-west-3  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
```

aurora_list_builtins

Listet alle verfügbaren integrierten Aurora-PostgreSQL-Funktionen mit kurzen Beschreibungen und Funktionsdetails auf.

Syntax

```
aurora_list_builtins()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz

Beispiele

Im folgenden Beispiel sehen Sie Ergebnisse vom Aufruf der Funktion `aurora_list_builtins`.

```
=> SELECT *
```

```
FROM aurora_list_builtins();

               Name                | Result data type |                    Argument data
 types                     | Type | Volatility |  Parallel  | Security |
             Description
-----------------------------------+-----------------
+-------------------------------------------------------------+------+------------
+------------+----------
+-------------------------------------------------------------------------
 aurora_version                    | text             |
                           | func | stable     | safe       | invoker  | Amazon Aurora
 PostgreSQL-Compatible Edition version string
 aurora_stat_wait_type             | SETOF record     | OUT type_id smallint, OUT
 type_name text                 | func | volatile   | restricted | invoker  | Lists all
 supported wait types
 aurora_stat_wait_event            | SETOF record     | OUT type_id smallint, OUT
 event_id integer, OUT event_na.| func | volatile   | restricted | invoker  | Lists all
 supported wait events
                                   |                  |.me text
                          |      |            |            |          |
 aurora_list_builtins              | SETOF record     | OUT "Name" text, OUT "Result
 data type" text, OUT "Argum.| func | stable     | safe       | invoker  | Lists all
 Aurora built-in functions
                                   |                  |.ent data types" text, OUT
 "Type" text, OUT "Volatility" .|      |            |            |          |
                                   |                  |.text, OUT "Parallel" text, OUT
 "Security" text, OUT "Des.|      |            |            |          |
                                   |                  |.cription" text
                          |      |            |            |          |
 .
 .
 .
 aurora_stat_file                  | SETOF record     | OUT filename text, OUT
 allocated_bytes bigint, OUT used_.| func | stable     | safe       | invoker  | Lists
 all files present in Aurora storage
                                   |                  |.bytes bigint
                          |      |            |            |          |
 aurora_stat_get_db_commit_latency | bigint           | oid
                           | func | stable     | restricted | invoker  | Per DB commit
 latency in microsecs
```

aurora_replica_status

Zeigt den Status aller Aurora-PostgreSQL-Reader-Knoten an.

Syntax

```
aurora_replica_status()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `server_id` – Die DB-Instance-ID (Kennung).
- `session_id` – Eine eindeutige ID für die aktuelle Sitzung, die wie folgt für die primäre Instance und Reader-Instances zurückgegeben wird:
  - Für die primäre Instance ist `session_id` immer `` `MASTER_SESSION_ID' ``.
  - Für Reader-Instances ist `session_id` immer die `UUID` (Universally Unique Identifier, eindeutige Kennung) der Reader-Instance.
- `durable_lsn` – Die Log-Sequenznummer (LSN), die im Speicher abgelegt wurde.
  - Für das primäre Volume die dauerhafte LSN des primären Volumes (VDL), die derzeit in Kraft ist.
  - Für alle sekundären Volumes die VDL des primären Volumes, auf das das sekundäre Volume erfolgreich angewendet wurde.

  > **Note**
  >
  > Eine Log-Sequenznummer (LSN) ist eine eindeutige fortlaufende Nummer, die einen Datensatz im Datenbank-Transaktionsprotokoll identifiziert. LSNs werden so angeordnet, dass eine größere LSN eine Transaktion darstellt, die später in der Sequenz auftritt.

- `highest_lsn_rcvd` – Die höchste (neueste) LSN, die die DB-Instance von der Writer-DB-Instance empfangen hat.
- `current_read_lsn` – Die LSN des letzten Snapshots, die auf alle Reader angewendet wurde.
- `cur_replay_latency_in_usec` – Die Anzahl der Mikrosekunden, die voraussichtlich benötigt werden, um das Protokoll auf der sekundären Instance wiederzugeben.

- `active_txns` – Die Anzahl der derzeit aktiven Transaktionen.
- `is_current` – Nicht verwendet.
- `last_transport_error` – Fehlercode für die letzte Replikation.
- `last_error_timestamp` – Zeitstempel des letzten Replikationsfehlers.
- `last_update_timestamp` – Zeitstempel der letzten Aktualisierung auf den Replikatstatus. Ab Aurora PostgreSQL 13.9 ist der `last_update_timestamp`-Wert für die DB-Instance, mit der Sie verbunden sind, auf `NULL` festgelegt.
- `feedback_xmin` – Der Hot Standby feedback_xmin des Replikats. Die minimale (älteste) aktive Transaktions-ID, die von der DB-Instance verwendet wird.
- `feedback_epoch` – Die Epoche, die die DB-Instance beim Erzeugen der Hot-Standby-Informationen verwendet.
- `replica_lag_in_msec` – Zeit, die die Leser-Instance in Millisekunden hinter der Schreiber-Instance zurückbleibt
- `log_stream_speed_in_kib_per_second` – Der Durchsatz des Protokollstreams in Kilobyte pro Sekunde.
- `log_buffer_sequence_number` – Die Sequenznummer des Protokollpuffers.
- `oldest_read_view_trx_id` – Nicht verwendet.
- `oldest_read_view_lsn` – Die älteste LSN, die von der DB-Instance zum Lesen aus dem Speicher verwendet wird.
- `pending_read_ios` – Die ausstehenden Seiten-Lesevorgänge, die noch im Replikat anstehen.
- `read_ios` – Die Gesamtzahl der Lesevorgänge auf dem Replikat.
- `iops` – Nicht verwendet.
- `cpu` – CPU-Auslastung des Aurora-Speicher-Daemon für jeden Knoten im Cluster. Weitere Informationen zur CPU-Auslastung durch die Instance finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

Nutzungshinweise

Alle derzeit verfügbaren Aurora-PostgreSQL-Versionen unterstützen diese Funktion. Die Funktion `aurora_replica_status` gibt Werte aus dem Replikatstatus-Manager eines Aurora-PostgreSQL-DB-Clusters zurück. Sie können diese Funktion verwenden, um Informationen über den Status der Replikation auf Ihrem Aurora-PostgreSQL-DB-Cluster zu erhalten, einschließlich Metriken für alle DB-Instances in Ihrem Aurora-DB-Cluster. Sie können z. B. Folgendes tun:

- Informationen über die Art der Instance (Writer, Reader) im Aurora-PostgreSQL-DB-Cluster abrufen – Diese Informationen erhalten Sie, indem Sie sich die Werte der folgenden Spalten ansehen:
  - `server_id` – Enthält den Namen der Instance, die Sie beim Erstellen der Instance festgelegt haben. In einigen Fällen, z. B. für die primäre (Writer-)Instance, wird der Name normalerweise durch Anhängen von -instance-1 an den Namen, den Sie für Ihren Aurora-PostgreSQL-DB-Cluster festlegen, für Sie erstellt.
  - `session_id` – Das Feld `session_id` gibt an, ob es sich bei der Instance um einen Reader oder einen Writer handelt. Für eine Writer-Instance ist `session_id` immer auf `"MASTER_SESSION_ID"` festgelegt. Für eine Reader-Instance ist `session_id` auf die `UUID` des spezifischen Readers festgelegt.
- Häufige Replikationsprobleme wie Replikatverzögerung, diagnostizieren – Replikatverzögerung ist die Zeit in Millisekunden, die der Seiten-Cache einer Reader-Instance hinter dem Cache der Writer-Instance zurückbleibt. Diese Verzögerung tritt auf, weil Aurora-Cluster eine asynchrone Replikation verwenden, wie unter Replikation mit Amazon Aurorabeschrieben. Diese ist in der Spalte `replica_lag_in_msec` in den von dieser Funktion zurückgegebenen Ergebnissen zu sehen. Verzögerungen können auch auftreten, wenn eine Abfrage aufgrund von Konflikten mit der Wiederherstellung auf einem Standby-Server abgebrochen wird. Unter `pg_stat_database_conflicts()` können Sie nachsehen, ob ein solcher Konflikt die Replikatverzögerung verursacht (oder nicht). Weitere Informationen finden Sie unter Die Statistikerfassung in der PostgreSQL-Dokumentation. Weitere Informationen zu Hochverfügbarkeit und Replikation finden Sie unter Häufig gestellte Fragen zu Amazon Aurora.

  Amazon CloudWatch speichert `replica_lag_in_msec`-Ergebnisse im Laufe der Zeit als `AuroraReplicaLag`-Metrik. Weitere Informationen zur Verwendung der CloudWatch-Metriken für Aurora finden Sie unter Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch.

Weitere Informationen zur Fehlerbehebung bei Aurora-Lesereplikaten und Neustarts finden Sie unter Warum ist mein Amazon-Aurora-Lesereplikat in den Rückstand geraten und wurde neu gestartet? im AWS -Support Center.

Beispiele

Im folgenden Beispiel wird veranschaulicht, wie Sie den Replikationsstatus aller Instances in einem Aurora-PostgreSQL-DB-Cluster abrufen:

```
=> SELECT *
```

```
FROM aurora_replica_status();
```

Das folgende Beispiel zeigt die Writer-Instance im Aurora-PostgreSQL-DB-Cluster `docs-lab-apg-main`:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
 db-119-001-instance-01 | writer
```

Im folgenden Beispiel werden alle Reader-Instances in einem Cluster aufgeführt:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
db-119-001-instance-02  | reader
db-119-001-instance-03  | reader
db-119-001-instance-04  | reader
db-119-001-instance-05  | reader
(4 rows)
```

Im folgenden Beispiel werden alle Instances aufgeführt und es wird angegeben, wie weit jede Instance hinter dem Writer zurückbleibt und wie lange seit dem letzten Update:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role,
    replica_lag_in_msec AS replica_lag_ms,
```

```
    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
 1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
       server_id        | instance_role | replica_lag_ms | last_update_age_ms
------------------------+---------------+----------------+--------------------
 db-124-001-instance-03 | writer        |         [NULL] |               1756
 db-124-001-instance-01 | reader        |             13 |               1756
 db-124-001-instance-02 | reader        |             13 |               1756
(3 rows)
```

aurora_stat_activity

Gibt eine Zeile pro Serverprozess zurück, in der Informationen zur aktuellen Aktivität dieses Prozesses angezeigt werden

Syntax

```
aurora_stat_activity();
```

Argumente

Keine

Rückgabetyp

Gibt eine Zeile pro Serverprozess zurück. Zusätzlich zu den `pg_stat_activity`-Spalten wird das folgende Feld hinzugefügt:

- planid – Plan-ID

Nutzungshinweise

Eine zusätzliche Ansicht zu `pg_stat_activity` mit denselben Spalten sowie einer zusätzlichen `plan_id`-Spalte, die den aktuellen Abfrageausführungsplan darstellt

`aurora_compute_plan_id` muss aktiviert sein, damit die Ansicht eine plan_id zurückgibt.

Diese Funktion ist in den Aurora-PostgreSQL-Versionen 14.10, 15.5 und für alle späteren Versionen verfügbar.

Beispiele

Die folgende Beispielabfrage aggregiert die Hauptlast nach query_id und plan_id.

```
db1=# select count(*), query_id, plan_id
db1-# from aurora_stat_activity() where state = 'active'
db1-# and pid <> pg_backend_pid()
db1-# group by query_id, plan_id
db1-# order by 1 desc;

count |  query_id             |  plan_id
-------+----------------------+-------------
 11    | -5471422286312252535 | -2054628807
 3     | -6907107586630739258 | -815866029
 1     | 5213711845501580017  |  300482084
(3 rows)
```

Wenn sich der für query_id verwendete Plan ändert, wird von aurora_stat_activity eine neue plan_id gemeldet.

```
count  |  query_id            |  plan_id
-------+----------------------+-------------
 10    | -5471422286312252535 | 1602979607
 1     | -6907107586630739258 | -1809935983
 1     | -2446282393000597155 | -207532066
(3 rows)
```

aurora_stat_backend_waits

Zeigt Statistiken für die Warteaktivität für einen bestimmten Backend-Prozess an.

Syntax

```
aurora_stat_backend_waits(pid)
```

Argumente

`pid` – Die ID für den Backend-Prozess. Sie können Prozess-IDs in der Ansicht `pg_stat_activity` abrufen.

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `type_id` – Eine Zahl, die die Art des Warteereignisses angibt, wie 1 für eine einfache Sperre (`LWLock`), 3 für eine Sperre oder 6 für eine Clientsitzung, um nur einige Beispiele zu nennen. Diese Werte werden aussagekräftig, wenn Sie die Ergebnisse dieser Funktion mit Spalten der Funktion `aurora_stat_wait_type` verbinden, wie in Beispiele gezeigt.
- `event_id` – Eine ID-Nummer für das Warteereignis. Verbinden Sie diesen Wert mit den Spalten von `aurora_stat_wait_event`, um aussagekräftige Ereignisnamen zu erhalten.
- `waits` – Die Anzahl der Warteereignisse, die für die angegebene Prozess-ID aufgelaufen sind.
- `wait_time` – Wartezeit in Millisekunden.

Nutzungshinweise

Sie können diese Funktion verwenden, um bestimmte Backend-Warteereignisse (Sitzung) zu analysieren, die seit dem Öffnen einer Verbindung aufgetreten sind. Aussagekräftigere Informationen über Warteereignisnamen und -typen erhalten Sie, wenn Sie diese Funktion mit `aurora_stat_wait_type` und `aurora_stat_wait_event` kombinieren, indem Sie JOIN verwenden, wie in den Beispielen gezeigt.

Beispiele

Dieses Beispiel zeigt alle Warteereignisse, Typen und Ereignisnamen für die Backend-Prozess-ID 3027.

```
=> SELECT type_name, event_name, waits, wait_time
        FROM aurora_stat_backend_waits(3027)
NATURAL JOIN aurora_stat_wait_type()
NATURAL JOIN aurora_stat_wait_event();
type_name |       event_name       | waits | wait_time
----------+------------------------+-------+------------
 LWLock    | ProcArrayLock          |     3 |         27
 LWLock    | wal_insert             |   423 |      16336
```

```
 LWLock    | buffer_content         | 11840 |    1033634
 LWLock    | lock_manager           | 23821 |    5664506
 Lock      | tuple                  | 10258 |  152280165
 Lock      | transactionid          | 78340 | 1239808783
 Client    | ClientRead             | 34072 |   17616684
 IO        | ControlFileSyncUpdate  |     2 |          0
 IO        | ControlFileWriteUpdate |     4 |         32
 IO        | RelationMapRead        |     2 |        795
 IO        | WALWrite               | 36666 |      98623
 IO        | XactSync               |  4867 |    7331963
```

Dieses Beispiel zeigt aktuelle und kumulative Wartetypen und -ereignisse für alle aktiven Sitzungen (`pg_stat_activity state <> 'idle'`) (aber ohne die aktuelle Sitzung, die die Funktion aufruft (`pid <> pg_backend_pid()`)).

```
=> SELECT a.pid,
            a.usename,
            a.app_name,
            a.current_wait_type,
            a.current_wait_event,
            a.current_state,
            wt.type_name AS wait_type,
            we.event_name AS wait_event,
            a.waits,
            a.wait_time
       FROM (SELECT pid,
                    usename,
                    left(application_name,16) AS app_name,
                    coalesce(wait_event_type,'CPU') AS current_wait_type,
                    coalesce(wait_event,'CPU') AS current_wait_event,
                    state AS current_state,
                    (aurora_stat_backend_waits(pid)).*
               FROM pg_stat_activity
              WHERE pid <> pg_backend_pid()
                AND state <> 'idle') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |       wait_event       | waits | wait_time
-------+----------+----------+-------------------+--------------------+---------------
+-----------+------------------------+-------+-----------
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1937 |     29975
```

```
30099 | postgres | pgbench  | Lock              | transactionid      | active        |
LWLock    | buffer_content         | 22903 |    760498
30099 | postgres | pgbench  | Lock              | transactionid      | active        |
LWLock    | lock_manager           | 10012 |    223207
30099 | postgres | pgbench  | Lock              | transactionid      | active        |
Lock      | tuple                  | 20315 |  63081529
.
.
.
30099 | postgres | pgbench  | Lock              | transactionid      | active        |
IO        | WALWrite               | 93293 |    237440
30099 | postgres | pgbench  | Lock              | transactionid      | active        |
IO        | XactSync               | 13010 |  19525143
30100 | postgres | pgbench  | Lock              | transactionid      | active        |
LWLock    | ProcArrayLock          |     6 |        53
30100 | postgres | pgbench  | Lock              | transactionid      | active        |
LWLock    | wal_insert             |  1913 |     25450
30100 | postgres | pgbench  | Lock              | transactionid      | active        |
LWLock    | buffer_content         | 22874 |    778005
.
.
.
30109 | postgres | pgbench  | IO                | XactSync           | active        |
LWLock    | ProcArrayLock          |     3 |        71
30109 | postgres | pgbench  | IO                | XactSync           | active        |
LWLock    | wal_insert             |  1940 |     27741
30109 | postgres | pgbench  | IO                | XactSync           | active        |
LWLock    | buffer_content         | 22962 |    776352
30109 | postgres | pgbench  | IO                | XactSync           | active        |
LWLock    | lock_manager           |  9879 |    218826
30109 | postgres | pgbench  | IO                | XactSync           | active        |
Lock      | tuple                  | 20401 |  63581306
30109 | postgres | pgbench  | IO                | XactSync           | active        |
Lock      | transactionid          | 50769 | 211645008
30109 | postgres | pgbench  | IO                | XactSync           | active        |
Client    | ClientRead             | 89901 |  44192439
```

Dieses Beispiel zeigt aktuelle und die drei wichtigsten (3) kumulativen Wartetypen und Warteereignisse für alle aktiven Sitzungen (`pg_stat_activity state <> 'idle'`) ohne die aktuelle Sitzung (`pid <>pg_backend_pid()`).

```
=> SELECT top3.*
       FROM (SELECT a.pid,
```

```
                    a.usename,
                    a.app_name,
                    a.current_wait_type,
                    a.current_wait_event,
                    a.current_state,
                    wt.type_name AS wait_type,
                    we.event_name AS wait_event,
                    a.waits,
                    a.wait_time,
                    RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)
                FROM (SELECT pid,
                             usename,
                             left(application_name,16) AS app_name,
                             coalesce(wait_event_type,'CPU') AS current_wait_type,
                             coalesce(wait_event,'CPU') AS current_wait_event,
                             state AS current_state,
                             (aurora_stat_backend_waits(pid)).*
                        FROM pg_stat_activity
                       WHERE pid <> pg_backend_pid()
                         AND state <> 'idle') a
          NATURAL JOIN aurora_stat_wait_type() wt
          NATURAL JOIN aurora_stat_wait_event() we) top3
 WHERE RANK <=3;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |   wait_event    |  waits  | wait_time  | rank
-------+----------+----------+-------------------+--------------------+---------------
+-----------+-----------------+---------+------------+------
 20567 | postgres | psql     | CPU               | CPU                | active        |
 LWLock    | wal_insert      |   25000 |   67512003 |    1
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | WALWrite        | 3071758 |    1016961 |    2
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | BufFileWrite    |   20750 |     184559 |    3
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237350 | 1265580011 |    1
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93641 |  341472318 |    2
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Client    | ClientRead      |  417556 |  204796837 |    3
 .
 .
 .
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | transactionid   |  238068 | 1265816822 |    1
```

```
 27745 | postgres | pgbench  | IO               | XactSync          | active        |
 Lock      | tuple           |   93210 |  338312247 |    2
 27745 | postgres | pgbench  | IO               | XactSync          | active        |
 Client    | ClientRead      |  419157 |  207836533 |    3
 27746 | postgres | pgbench  | Lock             | transactionid     | active        |
 Lock      | transactionid   |  237621 | 1264528811 |    1
 27746 | postgres | pgbench  | Lock             | transactionid     | active        |
 Lock      | tuple           |   93563 |  339799310 |    2
 27746 | postgres | pgbench  | Lock             | transactionid     | active        |
 Client    | ClientRead      |  417304 |  208372727 |    3
```

aurora_stat_bgwriter

`aurora_stat_bgwriter` ist eine Statistikanzeige mit Informationen über Cache-Schreibvorgänge für optimierte Lesevorgänge.

Syntax

```
aurora_stat_bgwriter()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit allen Spalten von `pg_stat_bgwriter` und den folgenden zusätzlichen Spalten. Weitere Informationen zu den Spalten von `pg_stat_bgwriter` finden Sie unter `pg_stat_bgwriter`.

Sie können die Statistiken für diese Funktion mit `pg_stat_reset_shared("bgwriter")` zurücksetzen.

- `orcache_blks_written` – Gesamtzahl der geschriebenen Cache-Datenblöcke für optimierte Lesevorgänge.
- `orcache_blk_write_time` – Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit, die für das Schreiben von Cache-Datendateiblöcken für optimierte Lesevorgänge aufgewendet wurde, in Millisekunden aufgezeichnet. Weitere Informationen finden Sie unter track_io_timing.

Nutzungshinweise

Diese Funktion steht in folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- 15.4 und höhere 15-Versionen
- 14.9 und höhere 14-Versionen

Beispiele

```
=> select * from aurora_stat_bgwriter();
-[ RECORD 1 ]-----------------+-----------
orcache_blks_written          | 246522
orcache_blk_write_time        | 339276.404
```

aurora_stat_database

Enthält alle Spalten von pg_stat_database und fügt am Ende neue Spalten hinzu.

Syntax

```
aurora_stat_database()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit allen Spalten von `pg_stat_database` und den folgenden zusätzlichen Spalten. Weitere Informationen zu den Spalten von `pg_stat_database` finden Sie unter `pg_stat_database`.

- `storage_blks_read`: Gesamtzahl der gemeinsam genutzten Blöcke, die aus dem Aurora-Speicher in dieser Datenbank gelesen wurden.
- `orcache_blks_hit`: Gesamtzahl der Treffer im Cache für optimierte Lesevorgänge in dieser Datenbank.
- `local_blks_read`: Gesamtzahl der in dieser Datenbank gelesenen lokalen Blöcke.

- `storage_blk_read_time`: Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit für das Lesen von Datendateiblöcken aus dem Aurora-Speicher in Millisekunden aufgezeichnet. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.
- `local_blk_read_time`: Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit für das Lesen von lokalen Datendateiblöcken in Millisekunden aufgezeichnet. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.
- `orcache_blk_read_time`: Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit für das Lesen von Datendateiblöcken aus dem Cache für optimierte Lesevorgänge in Millisekunden aufgezeichnet. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.

> **Note**
>
> Der Wert von `blks_read` ist die Summe von `storage_blks_read`, `orcache_blks_hit` und `local_blks_read`.
> Der Wert von `blk_read_time` ist die Summe von `storage_blk_read_time`, `orcache_blk_read_time` und `local_blk_read_time`.

Nutzungshinweise

Diese Funktion steht in folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- 15.4 und höhere 15-Versionen
- 14.9 und höhere 14-Versionen

Beispiele

Das folgende Beispiel zeigt, wie alle Spalten von `pg_stat_database` enthalten sind und am Ende 6 neue Spalten angefügt werden:

```
=> select * from aurora_stat_database() where datid=14717;
-[ RECORD 1 ]------------+------------------------------
datid                    | 14717
datname                  | postgres
numbackends              | 1
xact_commit              | 223
```

```
xact_rollback            | 4
blks_read                | 1059
blks_hit                 | 11456
tup_returned             | 27746
tup_fetched              | 5220
tup_inserted             | 165
tup_updated              | 42
tup_deleted              | 91
conflicts                | 0
temp_files               | 0
temp_bytes               | 0
deadlocks                | 0
checksum_failures        |
checksum_last_failure    |
blk_read_time            | 3358.689
blk_write_time           | 0
session_time             | 1076007.997
active_time              | 3684.371
idle_in_transaction_time | 0
sessions                 | 10
sessions_abandoned       | 0
sessions_fatal           | 0
sessions_killed          | 0
stats_reset              | 2023-01-12 20:15:17.370601+00
orcache_blks_hit         | 425
orcache_blk_read_time    | 89.934
storage_blks_read        | 623
storage_blk_read_time    | 3254.914
local_blks_read          | 0
local_blk_read_time      | 0
```

aurora_stat_dml_activity

Meldet kumulative Aktivitäten für jeden Data-Manipulation-Language-Vorgangstyp (DML) in einer Datenbank in einem Aurora-PostgreSQL-Cluster.

Syntax

```
aurora_stat_dml_activity(database_oid)
```

Argumente

database_oid

Die Objekt-ID (OID) der Datenbank im Aurora-PostgreSQL-Cluster.

Rückgabetyp

SETOF-Datensatz

Nutzungshinweise

Die Funktion `aurora_stat_dml_activity` ist nur mit Version 3.1 von Aurora PostgreSQL verfügbar, die mit der PostgreSQL-Engine 11.6 und höher kompatibel ist.

Verwenden Sie diese Funktion in Aurora-PostgreSQL-Clustern mit einer großen Anzahl von Datenbanken, um zu ermitteln, welche Datenbanken mehr oder langsamere DML-Aktivitäten oder beides haben.

Die Funktion `aurora_stat_dml_activity` gibt die Anzahl der Ausführungen und die kumulative Latenz in Mikrosekunden für SELECT-, INSERT-, UPDATE- und DELETE-Operationen zurück. Der Bericht enthält nur erfolgreiche DML-Operationen.

Mit der PostgreSQL-Statistikzugriffsfunktion `pg_stat_reset` können Sie diese Statistik zurücksetzen. Mit der Funktion `pg_stat_get_db_stat_reset_time` können Sie überprüfen, wann diese Statistik zuletzt zurückgesetzt wurde. Weitere Informationen zu PostgreSQL-Statistikzugriffsfunktionen finden Sie unter Die Statistikerfassung in der PostgreSQL-Dokumentation.

Beispiele

Im folgenden Beispiel wird gezeigt, wie Sie DML-Aktivitätsstatistiken für die verbundene Datenbank melden.

```
--Define the oid variable from connected database by using \gset
=> SELECT oid,
          datname
     FROM pg_database
    WHERE datname=(select current_database()) \gset
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
```

```
--------------+--------------------------+--------------+--------------------------
+--------------+--------------------------+--------------+--------------------------
       178957 |                 66684115 |       171065 |                 28876649 |
    519538 |            1454579206167 |            1 |                        53027


-- Showing the same results with expanded display on
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]------------+--------------
select_count             | 178957
select_latency_microsecs | 66684115
insert_count             | 171065
insert_latency_microsecs | 28876649
update_count             | 519538
update_latency_microsecs | 1454579206167
delete_count             | 1
delete_latency_microsecs | 53027
```

Im folgenden Beispiel sehen Sie DML-Aktivitätsstatistiken für alle Datenbanken im Aurora-PostgreSQL-Cluster. Dieser Cluster hat zwei Datenbanken, `postgres` und `mydb`. Die durch Komma getrennte Liste entspricht den Feldern `select_count`, `select_latency_microsecs`, `insert_count`, `insert_latency_microsecs`, `update_count`, `update_latency_microsecs`, `delete_count` und `delete_latency_microsecs`.

Aurora PostgreSQL erstellt und verwendet eine Systemdatenbank namens `rdsadmin`, um administrative Vorgänge wie Backups, Wiederherstellungen, Zustandsprüfungen, Replikationen usw. zu unterstützen. Diese DML-Operationen haben keine Auswirkungen auf Ihren Aurora-PostgreSQL-Cluster.

```
=> SELECT oid,
    datname,
    aurora_stat_dml_activity(oid)
    FROM pg_database;
oid  |    datname     |                   aurora_stat_dml_activity
-------+---------------
+-------------------------------------------------------------------
 14006 | template0      | (,,,,,,,)
 16384 | rdsadmin       | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1      | (,,,,,,,)
```

```
 14007 | postgres       |
(178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb           | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

Das folgende Beispiel zeigt DML-Aktivitätsstatistiken für alle Datenbanken, die aus Gründen der Lesbarkeit in Spalten angeordnet sind.

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '()') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '()') AS select_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '()') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '()') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '()') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '()') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '()') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '()') AS delete_latency_microsecs
FROM  (SELECT datname,
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
      ) AS db;

    datname     | select_count | select_latency_microsecs | insert_count |
 insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
 delete_latency_microsecs
----------------+--------------+--------------------------+--------------
+--------------------------+--------------+--------------------------+--------------
+--------------------------
 template0      |              |                          |              |
             |              |                          |              |
 rdsadmin       | 4206523      | 2478812333               | 7009414      | 1338482258
             | 0            | 0                        | 0            | 0
 template1      |              |                          |              |
             |              |                          |              |
 fault_test     | 66           | 452099                   | 0            | 0
             | 0            | 0                        | 0            | 0
 db_access_test | 1            | 5982                     | 0            | 0
             | 0            | 0                        | 0            | 0
 postgres       | 42035        | 95179203                 | 5752         | 2678832898
             | 21157        | 441883182488             | 2            | 1520
 mydb           | 71           | 453514                   | 0            | 0
             | 1            | 190                      | 1            | 152
```

Das folgende Beispiel zeigt die durchschnittliche kumulative Latenz (kumulative Latenz geteilt durch die Anzahl) jeder DML-Operation für die Datenbank mit der OID 16401.

```
=> SELECT select_count,
          select_latency_microsecs,
          select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
          insert_count,
          insert_latency_microsecs,
          insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
          update_count,
          update_latency_microsecs,
          update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
          delete_count,
          delete_latency_microsecs,
          delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
     FROM aurora_stat_dml_activity(16401);
-[ RECORD 1 ]------------+-------------
select_count             | 451312
select_latency_microsecs | 80205857
select_latency_per_exec  | 177
insert_count             | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec  | 274
update_count             | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec  | 148478
delete_count             | 12
delete_latency_microsecs | 448
delete_latency_per_exec  | 37
```

aurora_stat_get_db_commit_latency

Ruft die kumulative Commit-Latenz in Mikrosekunden für Aurora-PostgreSQL-Datenbanken ab. Die Commit-Latenz wird als Zeit zwischen dem Senden einer Commit-Anforderung und dem Empfang der Commit-Bestätigung gemessen.

Syntax

```
aurora_stat_get_db_commit_latency(database_oid)
```

Argumente

database_oid

Die Objekt-ID (OID) der Aurora-PostgreSQL-Datenbank.

Rückgabetyp

SETOF-Datensatz

Nutzungshinweise

Amazon CloudWatch verwendet diese Funktion, um die durchschnittliche Commit-Latenz zu berechnen. Weitere Informationen zu Amazon-CloudWatch-Metriken und zur Behebung einer hohen Commit-Latenz finden Sie unter Anzeigen von Metriken in der Amazon-RDS-Konsole und Making better decisions about Amazon RDS with Amazon CloudWatch metrics.

Mit der PostgreSQL-Statistikzugriffsfunktion `pg_stat_reset` können Sie diese Statistik zurücksetzen. Mit der Funktion `pg_stat_get_db_stat_reset_time` können Sie überprüfen, wann diese Statistik zuletzt zurückgesetzt wurde. Weitere Informationen zu PostgreSQL-Statistikzugriffsfunktionen finden Sie unter Die Statistikerfassung in der PostgreSQL-Dokumentation.

Beispiele

Im folgenden Beispiel wird die kumulative Commit-Latenz für jede Datenbank im Cluster `pg_database` abgerufen.

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database;

  oid  |    datname     | aurora_stat_get_db_commit_latency
-------+----------------+-----------------------------------
 14006 | template0      |                                 0
 16384 | rdsadmin       |                         654387789
     1 | template1      |                                 0
 16401 | mydb           |                            229556
 69768 | postgres       |                             22011
```

Im folgenden Beispiel wird die kumulative Commit-Latenz für die aktuell verbundene Datenbank abgerufen. Vor dem Aufruf der Funktion `aurora_stat_get_db_commit_latency` wird im Beispiel

zuerst `\gset` verwendet, um eine Variable für das Argument `oid` zu definieren und ihren Wert aus der verbundenen Datenbank zu beziehen.

```
--Get the oid value from the connected database before calling
 aurora_stat_get_db_commit_latency
=> SELECT oid
     FROM pg_database
    WHERE datname=(SELECT current_database()) \gset
=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

 aurora_stat_get_db_commit_latency
-----------------------------------
                        1424279160
```

Im folgenden Beispiel wird die kumulative Commit-Latenz für die `mydb`-Datenbank im Cluster `pg_database` abgerufen. Dann wird diese Statistik mit der Funktion `pg_stat_reset` zurückgesetzt und die Ergebnisse werden angezeigt. Abschließend wird die Funktion `pg_stat_get_db_stat_reset_time` ausgeführt, um zu überprüfen, wann diese Statistik zuletzt zurückgesetzt wurde.

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database
    WHERE datname = 'mydb';

  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                           3320370


=> SELECT pg_stat_reset();
 pg_stat_reset
---------------


=> SELECT oid,
          datname,
          aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';
```

```
  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                                 6


=> SELECT *
     FROM pg_stat_get_db_stat_reset_time(16427);

 pg_stat_get_db_stat_reset_time
--------------------------------
 2021-04-29 21:36:15.707399+00
```

aurora_stat_logical_wal_cache

Zeigt die Cache-Auslastung für das logische Write-Ahead-Protokoll (WAL) pro Slot an.

Syntax

```
SELECT * FROM aurora_stat_logical_wal_cache()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `name` – Der Name der Replikationsgruppe.
- `active_pid` – Die ID des Walsender-Prozesses.
- `cache_hit` – Die Gesamtzahl der WAL-Cache-Treffer seit dem letzten Reset.
- `cache_miss` – Die Gesamtzahl der WAL-Cache-Fehler seit dem letzten Reset.
- `blks_read` – Die Gesamtzahl der Wal-Cache-Leseanfragen.
- `hit_rate` – Die WAL-Cache-Trefferrate (cache_hit/blks_read).
- `last_reset_timestamp` – Das letzte Mal, dass der Zähler zurückgesetzt wurde.

Nutzungshinweise

Diese Funktion steht in folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- 15.2 und alle höheren Versionen
- 14.7 und höhere Versionen
- 13.8 und höhere Versionen
- 12.12 und höhere Versionen
- 11.17 und höhere Versionen

Beispiele

Das folgende Beispiel zeigt zwei Replikations-Slots mit nur einer aktiven `aurora_stat_logical_wal_cache`-Funktion.

```
=> SELECT *
     FROM aurora_stat_logical_wal_cache();
    name    | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
------------+------------+-----------+------------+-----------+----------
+-------------------------------
 test_slot1 |      79183 |        24 |          0 |        24 | 100.00%  | 2022-08-05
 17:39:56.830635+00
 test_slot2 |            |         1 |          0 |         1 | 100.00%  | 2022-08-05
 17:34:04.036795+00
(2 rows)
```

aurora_stat_memctx_usage

Meldet die Speicherkontextnutzung für jeden PostgreSQL-Prozess.

Syntax

```
aurora_stat_memctx_usage()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `pid` – die ID des Prozesses.
- `name` – der Name des Speicherkontexts.
- `allocated` – Die Anzahl der Byte, die der Speicherkontext aus dem zugrunde liegenden Speichersubsystem abgerufen hat.
- `used` – Die Anzahl der an Clients des Speicherkontextes übergebenen Byte.
- `instances` – Die Anzahl der derzeit existierenden Kontexte dieses Typs.

Nutzungshinweise

Diese Funktion zeigt die Speicherkontextnutzung für jeden PostgreSQL-Prozess an. Einige Prozesse sind mit `anonymous` beschriftet. Die Prozesse werden nicht offengelegt, da sie eingeschränkte Schlüsselwörter enthalten.

Diese Funktion steht ab den folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- 15.3 und höhere 15-Versionen
- 14.8 und höhere 14-Versionen
- 13.11 und höhere 13-Versionen
- 12.15 und höhere 12-Versionen
- 11.20 und höhere 11-Versionen

Beispiele

Im folgenden Beispiel sehen Sie die Ergebnisse vom Aufruf der Funktion `aurora_stat_memctx_usage`.

```
=> SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                             | allocated |   used  | instances
-------+----------------------------------+-----------+---------+-----------
123864 | Miscellaneous                    |     19520 |   15064 |         3
123864 | Aurora File Context              |      8192 |     616 |         1
123864 | Aurora WAL Context               |      8192 |     296 |         1
123864 | CacheMemoryContext               |    524288 |  422600 |         1
123864 | Catalog tuple context            |     16384 |   13736 |         1
123864 | ExecutorState                    |     32832 |   28304 |         1
123864 | ExprContext                      |      8192 |    1720 |         1
```

```
123864 | GWAL record construction        |      1024 |     832 |         1
123864 | MdSmgr                          |      8192 |     296 |         1
123864 | MessageContext                  |    532480 |  353832 |         1
123864 | PortalHeapMemory                |      1024 |     488 |         1
123864 | PortalMemory                    |      8192 |     576 |         1
123864 | printtup                        |      8192 |     296 |         1
123864 | RelCache hash table entries     |      8192 |    8152 |         1
123864 | RowDescriptionContext           |      8192 |    1344 |         1
123864 | smgr relation context           |      8192 |     296 |         1
123864 | Table function arguments        |      8192 |     352 |         1
123864 | TopTransactionContext           |      8192 |     632 |         1
123864 | TransactionAbortContext         |     32768 |     296 |         1
123864 | WAL record construction         |     50216 |   43904 |         1
123864 | hash table                      |     65536 |   52744 |         6
123864 | Relation metadata               |    191488 |  124240 |        87
104992 | Miscellaneous                   |      9280 |    7728 |         3
104992 | Aurora File Context             |      8192 |     376 |         1
104992 | Aurora WAL Context              |      8192 |     296 |         1
104992 ||Autovacuum Launcher             |      8192 |     296 |         1
104992 | Autovacuum database list        |     16384 |     744 |         2
104992 | CacheMemoryContext              |    262144 |  140288 |         1
104992 | Catalog tuple context           |      8192 |     296 |         1
104992 | GWAL record construction        |      1024 |     832 |         1
104992 | MdSmgr                          |      8192 |     296 |         1
104992 | PortalMemory                    |      8192 |     296 |         1
104992 | RelCache hash table entries     |      8192 |     296 |         1
104992 | smgr relation context           |      8192 |     296 |         1
104992 | Autovacuum start worker (tmp)   |      8192 |     296 |         1
104992 | TopTransactionContext           |     16384 |     592 |         2
104992 | TransactionAbortContext         |     32768 |     296 |         1
104992 | WAL record construction         |     50216 |   43904 |         1
104992 | hash table                      |     49152 |   34024 |         4
(39 rows)
```

Einige eingeschränkte Schlüsselwörter werden ausgeblendet und die Ausgabe sieht wie folgt aus:

```
postgres=>SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                            | allocated |   used  | instances
-------+---------------------------------+-----------+---------+-----------
  5482 | anonymous                       |      8192 |     456 |         1
  5482 | anonymous                       |      8192 |     296 |         1
```

aurora_stat_optimized_reads_cache

Diese Funktion zeigt mehrstufige Cache-Statistiken an.

Syntax

```
aurora_stat_optimized_reads_cache()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- `total_size` – Gesamtgröße des Cache für optimierte Lesevorgänge
- `used_size` – Verwendete Seitengröße im Cache für optimierte Lesevorgänge

Nutzungshinweise

Diese Funktion steht in folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- 15.4 und höhere 15-Versionen
- 14.9 und höhere 14-Versionen

Beispiele

Das folgende Beispiel zeigt die Ausgabe für eine r6gd.8xlarge-Instance:

```
=> select pg_size_pretty(total_size) as total_size, pg_size_pretty(used_size)
               as used_size from aurora_stat_optimized_reads_cache();
total_size | used_size
-----------+-----------
1054 GB    | 975 GB
```

aurora_stat_plans

Gibt eine Zeile für jeden nachverfolgten Ausführungsplan zurück

Syntax

```
aurora_stat_plans(
    showtext
)
```

Argumente

- showtext – Zeigt den Abfrage- und Plantext an. Gültige Werte sind NULL, true oder false. Bei true werden Abfrage- und Plantext angezeigt.

Rückgabetyp

Gibt für jeden nachverfolgten Plan eine Zeile zurück, die alle Spalten von `aurora_stat_statements` und die folgenden zusätzlichen Spalten enthält

- planid – Plan-ID
- explain_plan – Erklärt den Plantext
- plan_type:
  - `no plan` – Kein Plan wurde erfasst
  - `estimate` – Plan mit geschätzten Kosten erfasst
  - `actual` – Plan mit EXPLAIN ANALYZE erfasst
- plan_captured_time – Zeit der letzten Planerfassung

Nutzungshinweise

`aurora_compute_plan_id` muss aktiviert sein und `pg_stat_statements` muss sich in `shared_preload_libraries` befinden, damit die Pläne nachverfolgt werden können.

Die Anzahl der verfügbaren Pläne wird durch den im Parameter `pg_stat_statements.max` festgelegten Wert gesteuert. Wenn `aurora_compute_plan_id` aktiviert ist, können Sie die Pläne bis zu diesem angegebenen Wert in `aurora_stat_plans` nachverfolgen.

Diese Funktion ist in den Aurora-PostgreSQL-Versionen 14.10, 15.5 und für alle späteren Versionen verfügbar.

Beispiele

Im folgenden Beispiel werden die beiden Pläne zur Abfrage-ID -5471422286312252535 erfasst und die Statistik der Anweisungen wird über die planid nachverfolgt.

```
db1=# select calls, total_exec_time, planid, plan_captured_time, explain_plan
db1-# from aurora_stat_plans(true)
db1-# where queryid = '-5471422286312252535'

calls    |  total_exec_time   |   planid    |      plan_captured_time       |
                explain_plan
---------+--------------------+-------------+-------------------------------
+-----------------------------------------------------------------
 1532632 |  3209846.097107853 |  1602979607 | 2023-10-31 03:27:16.925497+00 | Update on
 pgbench_branches                                      +
         |                    |             |                               |   ->
 Bitmap Heap Scan on pgbench_branches                     +
         |                    |             |                               |
 Recheck Cond: (bid = 76)                                   +
         |                    |             |                               |         -
>  Bitmap Index Scan on pgbench_branches_pkey          +
         |                    |             |                               |
     Index Cond: (bid = 76)
   61365 | 124078.18012200127 | -2054628807 | 2023-10-31 03:20:09.85429+00  | Update on
 pgbench_branches                                      +
         |                    |             |                               |   ->
 Index Scan using pgbench_branches_pkey on pgbench_branches+
         |                    |             |                               |
 Index Cond: (bid = 17)
```

aurora_stat_reset_wal_cache

Setzt den Zähler für den logischen WAL-Cache zurück.

Syntax

So setzen Sie einen bestimmten Slot zurück

```
SELECT * FROM aurora_stat_reset_wal_cache('slot_name')
```

So setzen Sie alle Slots zurück

```
SELECT * FROM aurora_stat_reset_wal_cache(NULL)
```

Argumente

`NULL` oder `slot_name`

Rückgabetyp

Statusmeldung, Textzeichenfolge

- Setzen Sie den logischen WAL-Cache-Zähler zurück – Erfolgsmeldung. Dieser Text wird zurückgegeben, wenn die Funktion erfolgreich ist.
- Der Replikations-Slot wurde nicht gefunden. Bitte versuchen Sie es noch einmal. – Fehlernachricht Dieser Text wird zurückgegeben, wenn die Funktion nicht erfolgreich ist.

Nutzungshinweise

Diese Funktion steht für die folgenden Versionen zur Verfügung.

- Aurora PostgreSQL 14.5 und höher
- Aurora PostgreSQL 13.8 und höhere Versionen
- Aurora PostgreSQL 12.12 und höhere Versionen
- Aurora PostgreSQL 11.17 und höhere Versionen

Beispiele

Im folgenden Beispiel wird die `aurora_stat_reset_wal_cache`-Funktion verwendet, um einen Slot namens `test_results` zurückzusetzen. Anschließend wird versucht, einen Slot zurückzusetzen, der nicht existiert.

```
=> SELECT *
     FROM aurora_stat_reset_wal_cache('test_slot');
aurora_stat_reset_wal_cache
------------------------------------
 Reset the logical wal cache counter.
(1 row)
=> SELECT *
     FROM aurora_stat_reset_wal_cache('slot-not-exist');
```

```
aurora_stat_reset_wal_cache
--------------------------------------------
 Replication slot not found. Please try again.
(1 row)
```

aurora_stat_resource_usage

Meldet die Ressourcennutzung in Echtzeit, bestehend aus Backend-Ressourcenmetriken und CPU-Auslastung für alle Backend-Prozesse von Aurora PostgreSQL

Syntax

```
aurora_stat_resource_usage()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit Spalten:

- pid – Prozess-ID
- allocated_memory – Vom Prozess zugewiesener Gesamtspeicher in Byte
- used_memory – Tatsächlich vom Prozess genutzter Speicher in Byte
- cpu_usage_percent – Prozentsatz der CPU-Auslastung des Prozesses

Nutzungshinweise

Diese Funktion zeigt die Backend-Ressourcennutzung für jeden Backend-Prozess von Aurora PostgreSQL an.

Diese Funktion steht ab den folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- Aurora PostgreSQL 17.5 und höhere Versionen von 17
- Aurora PostgreSQL 16.9 und höhere Versionen von 16
- Aurora PostgreSQL 15.13 und höhere Versionen von 15
- Aurora PostgreSQL 14.18 und höhere Versionen von 14
- Aurora PostgreSQL 13.21 und höhere Versionen von 13

Beispiele

Im folgenden Beispiel wird die Ausgabe der Funktion `aurora_stat_resource_usage` gezeigt.

```
=> select * from aurora_stat_resource_usage();
 pid  | allocated_memory | used_memory |   cpu_usage_percent
------+------------------+-------------+-----------------------
  666 |          1074032 |      333544 |   0.00729274882897963
  667 |           787312 |      287360 | 0.0029263928146372746
  668 |          3076776 |     1563488 |  0.006013116835953961
  684 |           803744 |      307480 |  0.002226855426881142
 2401 |          1232992 |      943144 |                     0
  647 |             8000 |         944 |   0.48853387812429855
  659 |           319344 |      243000 | 0.0004135602076683591
  663 |           262000 |      185736 |  0.008181301476644002
  664 |             9024 |        1216 |   0.10992313082653653
(9 rows)
```

aurora_stat_statements

Zeigt alle Spalten von `pg_stat_statements` an und fügt am Ende weitere Spalten hinzu.

Syntax

```
aurora_stat_statements(showtext boolean)
```

Argumente

showtext boolean

Rückgabetyp

SETOF-Datensatz mit allen Spalten von `pg_stat_statements` und den folgenden zusätzlichen Spalten. Weitere Informationen zu den Spalten von `pg_stat_statements` finden Sie unter `pg_stat_statements`.

Sie können die Statistiken für diese Funktion mit `pg_stat_statements_reset()` zurücksetzen.

- `storage_blks_read`: Gesamtzahl der gemeinsam genutzten Blöcke, die von dieser Anweisung aus dem Aurora-Speicher gelesen wurden.

- `orcache_blks_hit`: Gesamtzahl der mit dieser Anweisung erzielten Treffer im Cache für optimierte Lesevorgänge.
- `storage_blk_read_time` – Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit in Millisekunden aufgezeichnet, die die Anweisung für das Lesen von gemeinsam genutzten Blöcken aus dem Aurora-Speicher benötigt hat. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.
- `local_blk_read_time` – Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit in Millisekunden aufgezeichnet, die die Anweisung für das Lesen von lokalen Blöcken benötigt hat. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.
- `orcache_blk_read_time` – Wenn `track_io_timing` aktiviert ist, wird die Gesamtzeit in Millisekunden aufgezeichnet, die die Anweisung für das Lesen von gemeinsam genutzten Blöcken aus dem Cache für optimierte Lesevorgänge benötigt hat. Andernfalls beträgt der Wert Null. Weitere Informationen finden Sie unter track_io_timing.
- `total_plan_peakmem` – Gesamtsumme der Werte zur maximalen Speichernutzung während der Planungsphase für alle Aufrufe dieser Anweisung. Um die durchschnittliche maximale Speichernutzung während der Planung für die Anweisung zu ermitteln, teilen Sie diesen Wert durch die Anzahl der Aufrufe.
- `min_plan_peakmem` – Kleinster Wert der maximalen Speichernutzung während der Planung über alle Aufrufe dieser Anweisung hinweg
- `max_plan_peakmem` – Größter Wert der maximalen Speichernutzung während der Planung über alle Aufrufe dieser Anweisung hinweg
- `total_exec_peakmem` – Gesamtsumme der Werte zur maximalen Speichernutzung während der Ausführungsphase für alle Aufrufe dieser Anweisung. Um die durchschnittliche maximale Speichernutzung während der Ausführung für die Anweisung zu ermitteln, teilen Sie diesen Wert durch die Anzahl der Aufrufe.
- `min_exec_peakmem` – Kleinster Wert der maximalen Speichernutzung in Byte während der Ausführung über alle Aufrufe dieser Anweisung hinweg
- `max_exec_peakmem` – Größter Wert der maximalen Speichernutzung in Byte während der Ausführung über alle Aufrufe dieser Anweisung hinweg

> **Note**
>
> `total_plan_peakmen`, `min_plan_peakmem` und `max_plan_peakmem` werden nur überwacht, wenn die Einstellung `pg_stat_statements.track_planning` aktiviert ist.

Nutzungshinweise

Zur Verwendung der Funktion aurora_stat_statements() müssen Sie die Erweiterung `pg_stat_statements` in den Parameter `shared_preload_libraries` aufnehmen.

Diese Funktion steht in folgenden Aurora-PostgreSQL-Versionen zur Verfügung:

- 15.4 und höhere 15-Versionen
- 14.9 und höhere 14-Versionen

Die Spalten zur maximalen Speichernutzung sind in den folgenden Versionen verfügbar:

- 16.3 und höhere Versionen
- 15.7 und höhere Versionen
- 14.12 und höhere Versionen

Beispiele

Das folgende Beispiel zeigt, wie alle Spalten von pg_stat_statements enthalten sind und am Ende 11 neue Spalten angefügt werden:

```
=> select * from aurora_stat_statements(true) where query like 'with window_max%';
-[ RECORD 1 ]----------
+-----------------------------------------------------------------------------------------------
userid                 | 16409
dbid                   | 5
toplevel               | t
queryid                | -8347523682669847482
query                  | with window_max as (select custid, max(scratch) over (order by
 scratch rows between $1 preceding
and $2 following) wmax from ts) select sum(wmax), max(custid) from window_max
plans                  | 0
total_plan_time        | 0
min_plan_time          | 0
max_plan_time          | 0
mean_plan_time         | 0
stddev_plan_time       | 0
calls                  | 4
total_exec_time        | 254.105121
min_exec_time          | 57.503164000000005
max_exec_time          | 68.687418
```

```
mean_exec_time          | 63.52628025
stddev_exec_time        | 5.150765359979643
rows                    | 4
shared_blks_hit         | 200192
shared_blks_read        | 0
shared_blks_dirtied     | 0
shared_blks_written     | 0
local_blks_hit          | 0
local_blks_read         | 0
local_blks_dirtied      | 0
local_blks_written      | 0
temp_blks_read          | 0
temp_blks_written       | 0
blk_read_time           | 0
blk_write_time          | 0
temp_blk_read_time      | 0
temp_blk_write_time     | 0
wal_records             | 0
wal_fpi                 | 0
wal_bytes               | 0
jit_functions           | 0
jit_generation_time     | 0
jit_inlining_count      | 0
jit_inlining_time       | 0
jit_optimization_count  | 0
jit_optimization_time   | 0
jit_emission_count      | 0
jit_emission_time       | 0
storage_blks_read       | 0
orcache_blks_hit        | 0
storage_blk_read_time   | 0
local_blk_read_time     | 0
orcache_blk_read_time   | 0
total_plan_peakmem      | 0
min_plan_peakmem        | 0
max_plan_peakmem        | 0
total_exec_peakmem      | 6356224
min_exec_peakmem        | 1589056
max_exec_peakmem        | 1589056
```

aurora_stat_system_waits

Meldet Warteereignisinformationen für die Aurora-PostgreSQL-DB-Instance.

Syntax

```
aurora_stat_system_waits()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz

Nutzungshinweise

Diese Funktion gibt die kumulative Anzahl der Wartezeiten und die kumulative Wartezeit für jedes Warteereignis zurück, das von der DB-Instance generiert wird, mit der Sie derzeit verbunden sind.

Der zurückgegebene Datensatz enthält die folgenden Felder:

- `type_id` – die ID des Warteereignistyps.
- `event_id` – die ID des Warteereignisses.
- `waits` – die Häufigkeit, mit der das Warteereignis aufgetreten ist.
- `wait_time` – die Gesamtzeit in Mikrosekunden, die auf dieses Ereignis gewartet wurde.

Die von dieser Funktion zurückgegebenen Statistiken werden zurückgesetzt, wenn eine DB-Instance neu gestartet wird.

Beispiele

Im folgenden Beispiel sehen Sie Ergebnisse vom Aufruf der Funktion `aurora_stat_system_waits`.

```
=> SELECT *
    FROM aurora_stat_system_waits();
 type_id | event_id  |   waits   |  wait_time
---------+-----------+-----------+--------------
       1 |  16777219 |        11 |        12864
       1 |  16777220 |       501 |       174473
```

```
       1 |  16777270 |     53171 |     23641847
       1 |  16777271 |        23 |       319668
       1 |  16777274 |        60 |        12759
.
.
.
      10 | 167772231 |    204596 |    790945212
      10 | 167772232 |         2 |        47729
      10 | 167772234 |         1 |          888
      10 | 167772235 |         2 |           64
```

Im folgenden Beispiel wird gezeigt, wie Sie diese Funktion mit `aurora_stat_wait_event` und `aurora_stat_wait_type` ausführen können, um besser lesbare Ergebnisse zu erzeugen.

```
=> SELECT type_name,
             event_name,
             waits,
             wait_time
        FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();

 type_name |       event_name       |  waits   |  wait_time
-----------+------------------------+----------+--------------
 LWLock    | XidGenLock             |       11 |        12864
 LWLock    | ProcArrayLock          |      501 |       174473
 LWLock    | buffer_content         |    53171 |     23641847
 LWLock    | rdsutils               |        2 |        12764
 Lock      | tuple                  |    75686 |   2033956052
 Lock      | transactionid          |  1765147 |  47267583409
 Activity  | AutoVacuumMain         |   136868 |  56305604538
 Activity  | BgWriterHibernate      |     7486 |  55266949471
 Activity  | BgWriterMain           |     7487 |   1508909964
.
.
.
 IO        | SLRURead               |        3 |        11756
 IO        | WALWrite               | 52544463 |    388850428
 IO        | XactSync               |   187073 |    597041642
 IO        | ClogRead               |        2 |        47729
 IO        | OutboundCtrlRead       |        1 |          888
 IO        | OutboundCtrlWrite      |        2 |           64
```

aurora_stat_wait_event

Listet alle unterstützten Warteereignisse für Aurora PostgreSQL auf. Informationen zu Aurora PostgreSQL-Warteereignissen finden Sie unter Amazon-Aurora-PostgreSQL-Warteereignisse.

Syntax

```
aurora_stat_wait_event()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- type_id – die ID des Warteereignistyps.
- event_id – die ID des Warteereignisses.
- type_name – Name des Wartetyps
- event_name – Name des Warteereignisses

Nutzungshinweise

Kombinieren Sie diese Funktion mit anderen Funktionen wie `aurora_stat_wait_type` und `aurora_stat_system_waits`, um Ereignisnamen mit Ereignistypen anstelle von IDs anzuzeigen. Namen von Warteereignissen, die von dieser Funktion zurückgegeben werden, sind die gleichen Namen, die von der Funktion `aurora_wait_report` zurückgegeben werden.

Beispiele

Im folgenden Beispiel sehen Sie Ergebnisse vom Aufruf der Funktion `aurora_stat_wait_event`.

```
=>  SELECT *
    FROM aurora_stat_wait_event();

 type_id | event_id  |               event_name
---------+-----------+-----------------------------------------
       1 |  16777216 | <unassigned:0>
```

```
       1 |  16777217 | ShmemIndexLock
       1 |  16777218 | OidGenLock
       1 |  16777219 | XidGenLock
.
.
.
       9 | 150994945 | PgSleep
       9 | 150994946 | RecoveryApplyDelay
      10 | 167772160 | BufFileRead
      10 | 167772161 | BufFileWrite
      10 | 167772162 | ControlFileRead
.
.
.
      10 | 167772226 | WALInitWrite
      10 | 167772227 | WALRead
      10 | 167772228 | WALSync
      10 | 167772229 | WALSyncMethodAssign
      10 | 167772230 | WALWrite
      10 | 167772231 | XactSync
.
.
.
      11 | 184549377 | LsnAllocate
```

Im folgenden Beispiel werden `aurora_stat_wait_type` und `aurora_stat_wait_event` zusammengeführt, um aus Gründen der Lesbarkeit Typnamen und Ereignisnamen zurückzugeben.

```
=> SELECT *
    FROM aurora_stat_wait_type() t
    JOIN aurora_stat_wait_event() e
      ON t.type_id = e.type_id;

 type_id | type_name | type_id | event_id  |                event_name
---------+-----------+---------+-----------+-------------------------------------------
       1 | LWLock    |       1 |  16777216 | <unassigned:0>
       1 | LWLock    |       1 |  16777217 | ShmemIndexLock
       1 | LWLock    |       1 |  16777218 | OidGenLock
       1 | LWLock    |       1 |  16777219 | XidGenLock
       1 | LWLock    |       1 |  16777220 | ProcArrayLock
.
.
.
```

```
       3 | Lock      |       3 |  50331648 | relation
       3 | Lock      |       3 |  50331649 | extend
       3 | Lock      |       3 |  50331650 | page
       3 | Lock      |       3 |  50331651 | tuple
.
.
.
      10 | IO        |      10 | 167772214 | TimelineHistorySync
      10 | IO        |      10 | 167772215 | TimelineHistoryWrite
      10 | IO        |      10 | 167772216 | TwophaseFileRead
      10 | IO        |      10 | 167772217 | TwophaseFileSync
.
.
.
      11 | LSN       |      11 | 184549376 | LsnDurable
```

aurora_stat_wait_type

Listet alle unterstützten Wartetypen für Aurora PostgreSQL auf.

Syntax

```
aurora_stat_wait_type()
```

Argumente

Keine

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- type_id – die ID des Warteereignistyps.
- type_name – name des Wartetyps.

Nutzungshinweise

Kombinieren Sie diese Funktion mit anderen Funktionen wie `aurora_stat_wait_event` und `aurora_stat_system_waits`, um Warteereignisnamen mit Warteereignistypen anstelle von IDs anzuzeigen. Namen von Wartetypen, die von dieser Funktion zurückgegeben werden, sind die gleichen Namen, die von der Funktion `aurora_wait_report` zurückgegeben werden.

Beispiele

Im folgenden Beispiel sehen Sie Ergebnisse vom Aufruf der Funktion `aurora_stat_wait_type`.

```
=> SELECT *
     FROM aurora_stat_wait_type();
 type_id | type_name
---------+-----------
       1 | LWLock
       3 | Lock
       4 | BufferPin
       5 | Activity
       6 | Client
       7 | Extension
       8 | IPC
       9 | Timeout
      10 | IO
      11 | LSN
```

aurora_version

Gibt den Zeichenfolgewert der Versionsnummer der PostgreSQL-kompatiblen Edition von Amazon Aurora zurück.

Syntax

```
aurora_version()
```

Argumente

Keine

Rückgabetyp

CHAR- oder VARCHAR-Zeichenfolge

Nutzungshinweise

Diese Funktion zeigt die Version der Datenbank-Engine der PostgreSQL-kompatiblen Edition von Amazon Aurora an. Die Versionsnummer wird als Zeichenfolge zurückgegeben im Format *major*.*minor*.*patch*. Weitere Informationen zu den Aurora-PostgreSQL-Versionsnummern erhalten Sie unter Aurora-Versionsnummer.

Sie können auswählen, wann Upgrades von Nebenversionen angewendet werden sollen, indem Sie das Wartungsfenster für Ihren Aurora-PostgreSQL-DB-Cluster festlegen. Um zu erfahren wie, siehe Warten eines Amazon Aurora-DB-Clusters.

Ab der Veröffentlichung der PostgreSQL-Versionen 13.3, 12.8, 11.13 und 10.18 und für alle späteren Versionen sind die Aurora-Versionsnummern genauer auf die PostgreSQL-Versionsnummern abgestimmt. Weitere Informationen zu allen Aurora-PostgreSQL-Versionen finden Sie unter Updates für Amazon Aurora PostgreSQL in den Versionshinweisen für Aurora PostgreSQL.

Beispiele

Das folgende Beispiel zeigt die Ergebnisse des Aufrufs der `aurora_version`-Funktion auf einem Aurora-PostgreSQL-DB-Cluster, der PostgreSQL 12.7, Aurora-PostgreSQL- Version 4.2 ausführt und dann die gleiche Funktion auf einem Cluster mit Aurora-PostgreSQL-Version 13.3 ausführt.

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
```

Dieses Beispiel zeigt, wie Sie die Funktion mit verschiedenen Optionen verwenden, um mehr über die Aurora-PostgreSQL-Version zu erfahren. Dieses Beispiel verwendet eine Aurora-Versionsnummer, die sich von der PostgreSQL-Versionsnummer unterscheidet.

```
=> SHOW SERVER_VERSION;
 server_version
----------------
 12.7
(1 row)

=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
(1 row)
```

```
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
-------------------------------------------------------------------------------------------------
 PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-
gcc (GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------
 Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)
```

In diesem nächsten Beispiel wird die Funktion mit den gleichen Optionen wie im vorherigen Beispiel verwendet. Dieses Beispiel hat keine Aurora-Versionsnummer, die sich von der PostgreSQL-Versionsnummer unterscheidet.

```
=> SHOW SERVER_VERSION;
server_version
---------------
 13.3

=> SELECT * FROM aurora_version();
aurora_version
---------------
 13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 13.3

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
```

```
-----------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------------
 Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

aurora_volume_logical_start_lsn

Gibt die Log-Sequenznummer (LSN) zurück, die zur Identifizierung des Anfangs eines Datensatzes im Logical Write-Ahead Log (WAL)-Stream des Aurora-Cluster-Volumes verwendet wird.

Syntax

```
aurora_volume_logical_start_lsn()
```

Argumente

Keine

Rückgabetyp

`pg_lsn`

Nutzungshinweise

Diese Funktion identifiziert den Anfang des Datensatzes im logischen WAL-Stream für ein bestimmtes Aurora-Cluster-Volume. Sie können diese Funktion verwenden, während Sie ein Hauptversions-Upgrade durchführen, indem Sie logische Replikation und schnelles Klonen von Aurora verwenden, um zu ermitteln, an welcher LSN ein Snapshot oder ein Datenbankklon erstellt wird. Anschließend können Sie die logische Replikation verwenden, um die neueren Daten, die nach dem LSN aufgezeichnet wurden, kontinuierlich zu streamen und die Änderungen vom Publisher zum Subscriber zu synchronisieren.

Weitere Informationen zur Verwendung der logischen Replikation für ein Hauptversions-Upgrade finden Sie unter Verwenden der logischen Replikation, um ein Hauptversions-Upgrade für Aurora PostgreSQL durchzuführen.

Diese Funktion steht für die folgenden Versionen von Aurora PostgreSQL zur Verfügung:

- 15.2 und höhere 15-Versionen
- 14.3 und höhere 14-Versionen
- 13.6 und höhere 13-Versionen
- 12.10 und höhere 12-Versionen
- 11.15 und höhere 11-Versionen
- 10.20 und höhere 10-Versionen

Beispiele

Sie können die Log-Sequenznummer (LSN) mithilfe der folgenden Abfrage abrufen:

```
postgres=> SELECT aurora_volume_logical_start_lsn();

aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

aurora_wait_report

Diese Funktion zeigt die Aktivität „Warteereignis“ über einen bestimmten Zeitraum an.

Syntax

```
aurora_wait_report([time])
```

Argumente

Zeit (optional)

Die Zeit in Sekunden. Der Standardwert beträgt 10 Sekunden.

Rückgabetyp

SETOF-Datensatz mit den folgenden Spalten:

- type_name – Name des Wartetyps
- event_name – Name des Warteereignisses
- wait – Anzahl der Warteereignisse
- wait_time – Wartezeit in Millisekunden
- ms_per_wait – durchschnittliche Millisekunden pro Anzahl eines Warteereignisses
- waits_per_xact – durchschnittliche Warteereignisse pro Anzahl einer Transaktion
- ms_per_xact – durchschnittliche Millisekunden nach Anzahl der Transaktionen

Nutzungshinweise

Diese Funktion ist ab Aurora-PostgreSQL-Release 1.1 verfügbar, die mit PostgreSQL 9.6.6 und höheren Versionen kompatibel ist.

Wenn Sie diese Funktion nutzen möchten, müssen Sie zuerst die Aurora-PostgreSQL-Erweiterung `aurora_stat_utils` wie folgt erstellen:

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

Weitere Informationen zu verfügbaren Aurora-PostgreSQL-Erweiterungsversionen finden Sie unter Erweiterungsversionen für Amazon Aurora PostgreSQL in den Versionshinweisen für Aurora PostgreSQL.

Diese Funktion berechnet die Warteereignisse auf Instance-Ebene, indem zwei Snapshots von Statistikdaten aus der Funktion aurora_stat_system_waits() und den Postgre-SQL-Statistikansichten pg_stat_database verglichen werden.

Weitere Informationen über `aurora_stat_system_waits()` und `pg_stat_database` finden Sie unter Die Statistikerfassung in der PostgreSQL-Dokumentation.

Wenn diese Funktion ausgeführt wird, erstellt sie einen ersten Snapshot, wartet die angegebene Anzahl von Sekunden und erstellt dann einen zweiten Snapshot. Die Funktion vergleicht die beiden Snapshots und gibt den Unterschied zurück. Dieser Unterschied stellt die Aktivität der Instance für dieses Zeitintervall dar.

In der Writer-Instance zeigt die Funktion auch die Anzahl der Transaktionen, für die ein Commit ausgeführt wurde, und TPS (Transaktionen pro Sekunde) an. Diese Funktion gibt Informationen auf Instance-Ebene zurück und enthält alle Datenbanken in der Instance.

Beispiele

Dieses Beispiel zeigt, wie die Erweiterung aurora_stat_utils erstellt wird, um die Funktion aurora_wait_report verwenden zu können.

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

Dieses Beispiel zeigt, wie der Wartebericht 10 Sekunden lang überprüft wird.

```
=> SELECT *
     FROM aurora_wait_report();
NOTICE:  committed 34 transactions in 10 seconds (tps 3)
 type_name |    event_name     | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----------+-------------------+-------+-----------+-------------+----------------
+-------------
 Client    | ClientRead        |    26 |  30003.00 |    1153.961 |           0.76 |
 882.441
 Activity  | WalWriterMain     |    50 |  10051.32 |     201.026 |           1.47 |
 295.627
 Timeout   | PgSleep           |     1 |  10049.52 |   10049.516 |           0.03 |
 295.574
 Activity  | BgWriterHibernate |     1 |  10048.15 |   10048.153 |           0.03 |
 295.534
 Activity  | AutoVacuumMain    |    18 |   9941.66 |     552.314 |           0.53 |
 292.402
 Activity  | BgWriterMain      |     1 |    201.09 |     201.085 |           0.03 |
   5.914
 IO        | XactSync          |    15 |     25.34 |       1.690 |           0.44 |
   0.745
 IO        | RelationMapRead   |    12 |      0.54 |       0.045 |           0.35 |
   0.016
 IO        | WALWrite          |    84 |      0.21 |       0.002 |           2.47 |
   0.006
 IO        | DataFileExtend    |     1 |      0.02 |       0.018 |           0.03 |
   0.001
```

Dieses Beispiel zeigt, wie der Wartebericht 60 Sekunden lang überprüft wird.

```
=> SELECT *
     FROM aurora_wait_report(60);
NOTICE:  committed 1544 transactions in 60 seconds (tps 25)
```

```
 type_name |       event_name       |  waits  | wait_time | ms_per_wait |
 waits_per_xact | ms_per_xact
-----------+------------------------+---------+-----------+-------------
+---------------+-------------
 Lock      | transactionid          |    6422 | 477000.53 |      74.276 |
 4.16 |     308.938
 Client    | ClientRead             |    8265 | 270752.99 |      32.759 |
 5.35 |     175.358
 Activity  | CheckpointerMain       |       1 |  60100.25 |   60100.246 |
 0.00 |      38.925
 Timeout   | PgSleep                |       1 |  60098.49 |   60098.493 |
 0.00 |      38.924
 Activity  | WalWriterMain          |     296 |  60010.99 |     202.740 |
 0.19 |      38.867
 Activity  | AutoVacuumMain         |     107 |  59827.84 |     559.139 |
 0.07 |      38.749
 Activity  | BgWriterMain           |     290 |  58821.83 |     202.834 |
 0.19 |      38.097
 IO        | XactSync               |    1295 |  55220.13 |      42.641 |
 0.84 |      35.764
 IO        | WALWrite               | 6602259 |  47810.94 |       0.007 |
 4276.07 |      30.966
 Lock      | tuple                  |     473 |  29880.67 |      63.173 |
 0.31 |      19.353
 LWLock    | buffer_mapping         |     142 |   3540.13 |      24.930 |
 0.09 |       2.293
 Activity  | BgWriterHibernate      |     290 |   1124.15 |       3.876 |
 0.19 |       0.728
 IO        | BufFileRead            |    7615 |    618.45 |       0.081 |
 4.93 |       0.401
 LWLock    | buffer_content         |      73 |    345.93 |       4.739 |
 0.05 |       0.224
 LWLock    | lock_manager           |      62 |    191.44 |       3.088 |
 0.04 |       0.124
 IO        | RelationMapRead        |      72 |      5.16 |       0.072 |
 0.05 |       0.003
 LWLock    | ProcArrayLock          |       1 |      2.01 |       2.008 |
 0.00 |       0.001
 IO        | ControlFileWriteUpdate |       2 |      0.03 |       0.013 |
 0.00 |       0.000
 IO        | DataFileExtend         |       1 |      0.02 |       0.018 |
 0.00 |       0.000
```

```
IO        | ControlFileSyncUpdate  |       1 |      0.00 |       0.000 |
0.00 |       0.000
```

Amazon-Aurora-PostgreSQL-Parameter

Sie können Ihr Amazon-Aurora-DB-Cluster auf dieselbe Art und Weise verwalten, wie Sie Ihre anderen Amazon-RDS-DB-Instances verwalten, und zwar indem Sie die Parameter in einer DB-Parametergruppe verwenden. Amazon Aurora unterscheidet sich jedoch von Amazon RDS dadurch, dass ein Aurora-DB-Cluster mehrere DB-Instances hat. Einige der Parameter, mit denen Sie Ihren Amazon-Aurora-DB-Cluster verwalten, werden auf den gesamten Cluster angewendet, andere hingegen werden wie folgt nur auf eine bestimmte DB-Instance im DB-Cluster angewendet:

- DB-Cluster-Parametergruppe – Eine DB-Cluster-Parametergruppe enthält den Satz von Engine-Konfigurationsparametern, die für den gesamten Aurora-DB-Cluster gelten. Beispielsweise ist die Cluster-Cache-Verwaltung ein Feature eines Aurora-DB-Clusters, der von dem Parameter `apg_ccm_enabled` kontrolliert wird, der Teil der DB-Cluster-Parametergruppe ist. Die DB-Cluster-Parametergruppe enthält auch Standardeinstellungen für die DB-Parametergruppe für die DB-Instances, aus denen der Cluster besteht.
- DB-Parametergruppe – Eine DB-Parametergruppe ist der Satz von Engine-Konfigurationswerten, die für eine bestimmte DB-Instance dieses Engine-Typs gelten. Die DB-Parametergruppen für die PostgreSQL-DB-Engine werden von einer RDS-for-PostgreSQL-DB-Instance und einem Aurora-PostgreSQL-DB-Cluster verwendet. Diese Konfigurationseinstellungen gelten für Eigenschaften, die je nach DB-Instances in einem Aurora-Cluster unterschiedlich sein können, z. B. Größe der Speicherpuffer.

Sie verwalten Parameter auf Cluster-Ebene in DB-Clusterparametergruppen. Sie verwalten Parameter auf Instance-Ebene in DB-Parametergruppen. Sie können die Parameter mithilfe der Amazon-RDS-Konsole, AWS CLI oder Amazon RDS API verwalten. Es gibt separate Befehle für das Verwalten von Parametern auf der Cluster- und Instance-Ebene.

- Verwenden Sie den AWS CLI-Befehl modify-db-cluster-parameter-group, um Parameter auf Cluster-Ebene in einer DB-Cluster-Parametergruppe zu verwalten.
- Verwenden Sie den AWS CLI-Befehl modify-db-parameter-group, um Parameter auf Instance-Ebene in einer DB-Parametergruppe für eine DB-Instance in einem DB-Cluster zu verwalten.

Weitere Informationen zur AWS CLI finden Sie unter Verwenden der AWS CLI im AWS Command Line Interface-Benutzerhandbuch.

Weitere Informationen zu Parametergruppen finden Sie unter Parametergruppen für Amazon Aurora.

Anzeigen von Aurora-PostgreSQL-DB-Cluster- und DB-Parametern

Sie können alle verfügbaren Standardparametergruppen für RDS-for-PostgreSQL-DB-Instances und für Aurora-PostgreSQL-DB-Cluster in der AWS-Managementkonsole anzeigen. Die Standardparametergruppen für alle DB-Engines und DB-Cluster-Typen und -Versionen werden für jede AWS-Region aufgeführt. Alle benutzerdefinierten Parametergruppen werden ebenfalls aufgeführt.

Anstatt sie in der AWS-Managementkonsole zu überprüfen, können Sie in den DB-Cluster-Parametergruppen und DB-Parametergruppen enthaltene Parameter auch auflisten, indem Sie die AWS CLI oder Amazon RDS API verwenden. Beispielsweise können Sie Parameter in einer DB-Cluster-Parametergruppe aufführen, indem Sie den AWS CLI-Befehl describe-db-cluster-parameters wie folgt verwenden:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12
```

Der Befehl gibt detaillierte JSON-Beschreibungen für jeden Parameter zurück. Sie können die Menge der zurückgegebenen Informationen reduzieren, indem Sie die Option `--query` verwenden. Beispielsweise können Sie den Parameternamen, seine Beschreibung und die zulässigen Werte für die standardmäßige Aurora-PostgreSQL-12-DB-Cluster-Parametergruppe wie folgt abrufen:

Für Linux, macOS oder Unix:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Für Windows:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Eine Aurora-DB-Cluster-Parametergruppe umfasst die DB-Instance-Parametergruppe und Standardwerte für eine bestimmte Aurora-DB-Engine. Sie können die Liste der DB-Parameter aus

derselben standardmäßigen Aurora-PostgreSQL-Standardparametergruppe abrufen, indem Sie den AWS CLI-Befehl describe-db-parameters wie folgt verwenden.

Für Linux, macOS oder Unix:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Für Windows:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Die vorherigen Befehle geben Listen von Parametern aus dem DB-Cluster oder der DB-Parametergruppe mit Beschreibungen und anderen Details zurück, die in der Abfrage angegeben sind. Nachfolgend finden Sie eine Beispielantwort.

```
[
    [
        {
            "ParameterName": "apg_enable_batch_mode_function_execution",
            "ApplyType": "dynamic",
            "Description": "Enables batch-mode functions to process sets of rows at a
 time.",
            "AllowedValues": "0,1"
        }
    ],
    [
        {
            "ParameterName": "apg_enable_correlated_any_transform",
            "ApplyType": "dynamic",
            "Description": "Enables the planner to transform correlated ANY Sublink
 (IN/NOT IN subquery) to JOIN when possible.",
            "AllowedValues": "0,1"
        }
    ],...
```

Im Folgenden finden Sie Tabellen, die Werte für den standardmäßigen DB-Cluster-Parameter und den DB-Parameter für Aurora-PostgreSQL-Version 14 enthalten.

Aurora-PostgreSQL-Parameter auf Cluster-Ebene

Sie können die Parameter auf Cluster-Ebene für eine bestimmte Aurora-PostgreSQL-Version mithilfe der AWS-Managementkonsole, der AWS CLI oder Amazon-RDS-API anzeigen. Weitere Informationen zum Anzeigen von Parametern in einer Parametergruppe des DB-Clusters von Aurora PostgreSQL finden Sie unter Anzeigen von Parameterwerten für eine DB-Cluster Parametergruppe in Amazon Aurora.

Einige Parameter auf Cluster-Ebene sind nicht in allen Versionen verfügbar und manche sind veraltet. Hinweise zum Anzeigen der Parameter einer bestimmten Aurora-PostgreSQL-Version finden Sie unter Anzeigen von Aurora-PostgreSQL-DB-Cluster- und DB-Parametern.

Die folgende Tabelle enthält beispielsweise die Parameter, die in der standardmäßigen DB-Cluster-Parametergruppe für Aurora PostgreSQL Version 14 verfügbar sind. Wenn Sie einen Aurora-PostgreSQL-DB-Cluster erstellen, ohne Ihre eigene benutzerdefinierte DB-Parametergruppe anzugeben, wird Ihr DB-Cluster mit der standardmäßigen Aurora-DB-Cluster-Parametergruppe für die gewählte Version erstellt, z. B. `default.aurora-postgresql14`, `default.aurora-postgresql13` usw.

Eine Auflistung der DB-Instance-Parameter für dieselbe Standard-DB-Cluster-Parametergruppe finden Sie unter Aurora-PostgreSQL-Parameter auf Instance-Ebene.

Parametername	Beschreibung	Standard
ansi_constraint_trigger_ordering	Ändern Sie die Ausführungsreihenfolge von Einschränkungsauslösern für Kompatibilität mit dem ANSI-SQL-Standard.	–
ansi_force_foreign_key_checks	Stellen Sie sicher, dass referenzielle Aktionen wie kaskadiertes Löschen oder kaskadiertes Aktualisieren immer ablaufen, unabhängig von den diversen Auslöserkontexten der Aktion.	–
ansi_qualified_update_set_target	Unterstützt Tabellen- und Schemaqualifizierer in UPDATE … SET-Anweisungen.	–
apg_ccm_enabled	Aktivieren oder deaktivieren Sie die Cluster-Cache-Verwaltung für den Cluster.	–

Parametername	Beschreibung	Standard
apg_enable_batch_mode_function_execution	Aktiviert Batch-Modus-Funktionen, um mehrere Zeilen gleichzeitig zu verarbeiten.	–
apg_enable_correlated_any_transform	Ermöglicht es dem Planer, korrelierte ANY-Sublinks (IN/NOT IN Unterabfrage) nach Möglichkeit in JOIN umzuwandeln.	–
apg_enable_function_migration	Ermöglicht dem Planer, berechtigte Skalarfunktionen in die FROM-Klausel zu migrieren.	–
apg_enable_not_in_transform	Ermöglicht es dem Planer, NOT IN-Unterabfragen nach Möglichkeit in ANTI JOIN umzuwandeln.	–
apg_enable_remove_redundant_inner_joins	Ermöglicht dem Planer, redundante Inner Joins zu entfernen.	–
apg_enable_semijoin_push_down	Ermöglicht die Verwendung von Semijoin-Filtern für Hash-Joins.	–
apg_plan_mgmt.capture_plan_baselines	Baseline-Modus für Planerfassung. Manuell – Planerfassung für jede SQL-Anweisung aktivieren. Aus – Planerfassung deaktivieren. Automatisch – Planerfassung für Anweisungen in pg_stat_statement aktivieren, die die Berechtigungskriterien erfüllen.	aus
apg_plan_mgmt.max_databases	Legt die maximale Anzahl von Datenbanken fest, die Abfragen mit apg_plan_mgmt verwalten können.	10
apg_plan_mgmt.max_plans	Legt die maximale Anzahl von Plänen fest, die von apg_plan_mgmt zwischengespeichert werden können.	10000

Parametername	Beschreibung	Standard
apg_plan_mgmt.plan_retention_period	Maximale Anzahl von Tagen, seit ein Plan zuletzt verwendet wurde, bevor ein Plan automatisch gelöscht wird.	32
apg_plan_mgmt.unapproved_plan_execution_threshold	Geschätzte Gesamtplankosten, unter denen ein nicht genehmigter Plan ausgeführt wird.	0
apg_plan_mgmt.use_plan_baselines	Verwenden Sie nur genehmigte oder feste Pläne für verwaltete Anweisungen.	false
application_name	Legt den Namen der Anwendung fest, der in Statistiken und Protokollen verwendet werden soll.	–
array_nulls	Ermöglicht die Eingabe von NULL-Elementen in Arrays.	–
aurora_compute_plan_id	Überwacht Ausführungspläne von Abfragen, um die Ausführungspläne zu ermitteln, die zur aktuellen Datenbanklast beitragen, und um Leistungsstatistiken der Ausführungspläne im Laufe der Zeit zu verfolgen Weitere Informationen finden Sie unter Überwachen von Abfrageausführungsplänen für Aurora PostgreSQL.	on
authentication_timeout	(s) Legt die Zeit fest, die maximal zulässig ist, um die Client-Authentifizierung durchzuführen.	–
auto_explain.log_analyze	Verwenden Sie EXPLAIN ANALYZE zur Planprotokollierung.	–
auto_explain.log_buffers	Protokollieren der Puffernutzung.	–

Parametername	Beschreibung	Standard
auto_explain.log_format	EXPLAIN-Format, das für die Planprotokollierung verwendet werden soll.	–
auto_explain.log_min_duration	Legt die Mindestausführungszeit fest, ab der Pläne protokolliert werden.	–
auto_explain.log_nested_statements	Protokollieren verschachtelter Anweisungen.	–
auto_explain.log_timing	Erfasst Timing-Daten, nicht nur Zeilenzahlen.	–
auto_explain.log_triggers	Fügt Auslöserstatistiken in Pläne ein.	–
auto_explain.log_verbose	Verwenden Sie EXPLAIN VERBOSE zur Planprotokollierung.	–
auto_explain.sample_rate	Bruchteil der zu verarbeitenden Abfragen.	–
autovacuum	Startet den Untervorgang der Selbstbereinigung.	–
autovacuum_analyze_scale_factor	Anzahl von Tupel-Einfügungen, -Aktualisierungen oder -Löschungen vor der Analyse als Bruchteil von Reltupeln.	0.05
autovacuum_analyze_threshold	Mindestanzahl von Tupel-Einfügungen, -Aktualisierungen oder -Löschungen vor der Analyse.	–
autovacuum_freeze_max_age	Alter, bei dem eine Selbstbereinigung für eine Tabelle ausgeführt werden soll, um einen Transaktions-ID-Wraparound zu verhindern.	–

Parametername	Beschreibung	Standard
autovacuum_max_workers	Legt die maximale Anzahl gleichzeitig ausgeführter Worker-Vorgänge für die Selbstbereinigung fest.	GREATEST(DBInstanceClassMemory/64371566592,3)
autovacuum_multixact_freeze_max_age	Multixact-Alter, bei dem eine Tabelle sich selbst bereinigt, um Multixact-Wraparound zu verhindern.	–
autovacuum_naptime	(s) Inaktivitätszeit zwischen Selbstbereinigungen.	5
autovacuum_vacuum_cost_delay	(ms) Bereinigungskostenverzögerung in Millisekunden für die Selbstbereinigung.	5
autovacuum_vacuum_cost_limit	Bereinigungskostenbetrag für die Selbstbereinigung, der vor der Inaktivität verfügbar ist.	GREATEST(log(DBInstanceClassMemory/21474836480)*600,200)
autovacuum_vacuum_insert_scale_factor	Anzahl von Tupel-Einfügungen vor der Bereinigung als Bruchteil von Reltupeln.	–
autovacuum_vacuum_insert_threshold	Mindestanzahl von Tupel-Einfügungen vor der Bereinigung oder -1 zum Deaktivieren von Bereinigungen.	–
autovacuum_vacuum_scale_factor	Anzahl von Tupel-Aktualisierungen oder -Löschungen vor der Bereinigung als Bruchteil von Reltupeln.	0.1
autovacuum_vacuum_threshold	Mindestanzahl von Tupel-Aktualisierungen oder -Löschungen vor der Bereinigung.	–
autovacuum_work_mem	(kB) Legt den maximalen Arbeitsspeicher fest, der von jedem Selbstbereinigungs-Worker-Prozess verwendet werden soll.	GREATEST(DBInstanceClassMemory/32768,131072)

Parametername	Beschreibung	Standard
babelfishpg_tds.default_server_name	Der Standardname des Babelfish-Servers	Microsoft SQL Server
babelfishpg_tds.listen_adressen	Legt den Host-Namen oder die IP-Adresse(n) fest, die auf TDS überwacht werden sollen.	*
babelfishpg_tds.port	Legt den TDS-TCP-Port fest, den der Server überwacht.	1433
babelfishpg_tds.tds_debug_log_level	Legt die Protokollierungsstufe in TDS fest, 0 deaktiviert die Protokollierung	1
babelfishpg_tds.tds_default_numeric_precision	Legt die Standardgenauigkeit des numerischen Typs fest, der in den Metadaten der TDS-Spalte gesendet werden soll, wenn die Engine keine angibt.	38
babelfishpg_tds.tds_default_numeric_scale	Legt den Standardmaßstab des numerischen Typs fest, der in den Metadaten der TDS-Spalte gesendet werden soll, wenn die Engine keine angibt.	8
babelfishpg_tds.tds_default_packet_size	Legt die Standardpaketgröße für alle SQL.Server-Clients fest, die verbunden werden	4096
babelfishpg_tds.tds_default_protocol_version	Legt eine Standardversion des TDS-Protokolls für alle Clients fest, die verbunden werden	DEFAULT
babelfishpg_tds.tds_ssl_encrypt	Legt die SSL-Verschlüsselungsoption fest	0
babelfishpg_tds.tds_ssl_max_protocol_version	Legt die Höchstversion des SSL/TLS-Protokolls fest, die für die tds-Sitzung verwendet werden soll.	TLSv1.2

Parametername	Beschreibung	Standard
babelfishpg_tds.tds_ssl_min_protocol_version	Legt die Mindestversion des SSL/TLS-Protokolls fest, die für die tds-Sitzung verwendet werden soll.	TLSv1.2 ab Aurora-PostgreSQL-Version 16, TLSv1 für Versionen älter als Aurora-PostgreSQL-Version 16
babelfishpg_tsql.default_locale	Standardgebietsschema, das für Sortierungen verwendet wird, die von CREATE COLLATION erstellt wurden.	en-US
babelfishpg_tsql.migration_mode	Legt fest, ob mehrere Benutzerdatenbanken unterstützt werden	multi-db ab Aurora-PostgreSQL-Version 16, single-db für Versionen älter als Aurora-PostgreSQL-Version 16
babelfishpg_tsql.server_colation_name	Name der Standardsortierung für den Server	sql_latin1_general_cp1_ci_as
babelfishpg_tsql.version	Legt die Ausgabe der Variablen @@VERSION fest	default
backend_flush_after	(8 Kb) Anzahl der Seiten, nach denen zuvor ausgeführte Schreibvorgänge auf die Festplatte geschrieben werden.	–
backslash_quote	Legt fest, ob \\ in Zeichenfolgeliteralen zulässig ist.	–
backtrace_functions	Protokollrückverfolgung für Fehler in diesen Funktionen.	–
bytea_output	Legt das Ausgabeformat für bytea fest.	–
check_function_bodies	Überprüft die Funktionstexte während CREATE FUNCTION.	–

Parametername	Beschreibung	Standard
client_connection_check_interval	Legt das Zeitintervall zwischen Prüfungen auf Verbindungsabbrüche während der Ausführung von Abfragen fest.	–
client_encoding	Legt die Zeichensatzkodierung des Clients fest.	UTF8
client_min_messages	Legt die Nachrichtenebenen fest, die an den Client gesendet werden.	–
compute_query_id	Abfrage-IDs berechnen.	auto
config_file	Legt die Hauptkonfigurationsdatei des Servers fest.	/rdsdbdata/config/postgresql.conf
constraint_exclusion	Ermöglicht dem Planer die Verwendung von Einschränkungen, um Abfragen zu optimieren.	–
cpu_index_tuple_cost	Legt die Schätzung des Planers für die Kosten der Verarbeitung der einzelnen Indexeinträge während einer Indexprüfung fest.	–
cpu_operator_cost	Legt die Schätzung des Planers für die Kosten der Verarbeitung der einzelnen Operator- oder Funktionsaufrufe fest.	–
cpu_tuple_cost	Legt die Schätzung des Planers für die Kosten der Verarbeitung der einzelnen Tupeln (Zeilen) fest.	–
cron.database_name	Legt die Datenbank fest, um pg_cron-Metadatentabellen zu speichern	postgres
cron.log_run	Protokolliert alle Aufträge in der Tabelle job_run_details	on
cron.log_statement	Protokolliert alle Cron-Anweisungen vor der Ausführung.	aus

Parametername	Beschreibung	Standard
cron.max_running_jobs	Die maximale Anzahl von Aufträgen, die gleichzeitig ausgeführt werden können.	5
cron.use_background_workers	Aktiviert Hintergrund-Workers für pg_cron	on
cursor_tuple_fraction	Legt die Schätzung des Planers für den Bruchteil der Zeilen eines Cursors fest, die abgerufen werden.	–
data_directory	Legt das Datenverzeichnis des Servers fest.	/rdsdbdata/db
datestyle	Legt das Anzeigeformat für Datum- und Uhrzeitwerte fest.	–
db_user_namespace	Aktiviert Benutzernamen pro Datenbank.	–
deadlock_timeout	(ms) Legt die Zeit fest, die während einer Sperre gewartet wird, bevor auf einen Deadlock geprüft wird.	–
debug_pretty_print	Erstellt Einschübe für Analyse- und Planstruk turanzeigen.	–
debug_print_parse	Protokolliert die Analysestruktur der einzelnen Abfragen.	–
debug_print_plan	Protokolliert den Ausführungsplan der einzelnen Abfragen.	–
debug_print_rewritten	Protokolliert die neu geschriebene Analysest ruktur der einzelnen Abfragen.	–
default_statistics_target	Legt das Standardstatistikziel fest.	–
default_tablespace	Legt den Standardtabellenraum fest, in dem Tabellen und Indexe erstellt werden.	–

Parametername	Beschreibung	Standard
default_toast_compression	Legt die Standardkomprimierungsmethode für komprimierbare Werte fest.	–
default_transaction_deferrable	Legt den Standardaufschiebbarkeitsstatus neuer Transaktionen fest.	–
default_transaction_isolation	Legt die Transaktionsisolierungsstufe jeder neuen Transaktion fest.	–
default_transaction_read_only	Legt den Standardschreibschutzstatus neuer Transaktionen fest.	–
effective_cache_size	(8 kB) Legt die Annahme des Planers hinsichtlich der Größe des Datenträger-Caches fest.	SUM(DBInstanceClassMemory/12038,-50003)
effective_io_concurrency	Die Anzahl der gleichzeitigen Anfragen, die durch das Datenträgersubsystem effizient bearbeitet werden können.	–
enable_async_append	Ermöglicht den Planern die Verwendung paralleler Append-Pläne.	–
enable_bitmapscan	Ermöglicht die Verwendung von Bitmap-Prüfungsplänen durch den Planer.	–
enable_gathermerge	Ermöglicht die Verwendung von Gather-Merge-Plänen durch den Planer.	–
enable_hashagg	Ermöglicht die Verwendung von Hash-Aggregationsplänen durch den Planer.	–
enable_hashjoin	Ermöglicht die Verwendung von Hash-Join-Plänen durch den Planer.	–
enable_incremental_sort	Ermöglicht den Planern die Verwendung inkrementeller Sortierschritte.	–

Parametername	Beschreibung	Standard
enable_indexonlyscan	Ermöglicht die Verwendung von reinen Indexprüfungsplänen durch den Planer.	–
enable_indexscan	Ermöglicht die Verwendung von Indexprüfungsplänen durch den Planer.	–
enable_material	Ermöglicht die Verwendung von Materialisierung durch den Planer.	–
enable_memoize	Ermöglicht den Planern die Verwendung der Speicherung	–
enable_mergejoin	Ermöglicht die Verwendung von Merge-Join-Plänen durch den Planer.	–
enable_nestloop	Ermöglicht die Verwendung von Join-Plänen mit verschachtelten Schleifen durch den Planer.	–
enable_parallel_append	Ermöglicht den Planern die Verwendung paralleler Append-Pläne.	–
enable_parallel_hash	Ermöglicht den Planern die Verwendung paralleler Hash-Pläne.	–
enable_partition_pruning	Aktivieren Sie das Bereinigen von Planzeit- und Laufzeitpartitionen.	–
enable_partitionwise_aggregate	Ermöglicht die partitionsweise Aggregation und Gruppierung.	–
enable_partitionwise_join	Aktiviert partitionsweise Joins.	–
enable_seqscan	Ermöglicht die Verwendung von sequenziellen Prüfungsplänen durch den Planer.	–
enable_sort	Ermöglicht die Verwendung von expliziten Sortierschritten durch den Planer.	–

Parametername	Beschreibung	Standard
enable_tidscan	Ermöglicht die Verwendung von TID-Prüfungsplänen durch den Planer.	–
escape_string_warning	Warnt vor Escape-Notierungen mit Backslash in gewöhnlichen Zeichenfolgeliteralen.	–
exit_on_error	Beendet die Sitzung bei einem Fehler.	–
extra_float_digits	Legt die Anzahl der Stellen fest, die für Gleitkommawerte angezeigt werden.	–
force_parallel_mode	Erzwingt die Verwendung paralleler Abfrageeinrichtungen.	–
from_collapse_limit	Legt die Größe der FROM-Liste fest, jenseits der Unterabfragen nicht ausgeblendet werden.	–
geqo	Ermöglicht die genetische Abfrageoptimierung.	–
geqo_effort	GEQO: Der Aufwand, der verwendet wird, um den Standard für andere GEQO-Parameter festzulegen.	–
geqo_generations	GEQO: Die Zahl der Iterationen des Algorithmus.	–
geqo_pool_size	GEQO: Die Anzahl der Personen in der Population.	–
geqo_seed	GEQO: Der Seed für die zufällige Pfadauswahl.	–
geqo_selection_bias	GEQO: Selektiver Druck innerhalb der Population.	–
geqo_threshold	Legt den Schwellenwert für FROM-Elemente fest, jenseits derer GEQO verwendet wird.	–

Parametername	Beschreibung	Standard
gin_fuzzy_search_limit	Legt das maximal zulässige Ergebnis für die exakte Suche durch GIN fest.	–
gin_pending_list_limit	(kB) Legt die maximale Größe der ausstehen den Liste für den GIN-Index fest.	–
hash_mem_multiplier	Vielfaches von work_mem für Hash-Tabellen.	–
hba_file	Legt die hba-Konfigurationsdatei des Servers fest.	/rdsdbdata/config/ pg_hba.conf
hot_standby_feedback	Ermöglicht Feedback von einem Hot Standby zum primären, um Abfragekonflikte zu vermeiden.	on
huge_pages	Reduziert den Overhead, wenn eine DB-Instan ce mit großen, zusammenhängenden Arbeitssp eicherblöcken arbeitet, wie sie von gemeinsam genutzten Puffern verwendet werden. Der Parameter ist standardmäßig für alle DB-Instan ce-Klassen aktiviert, außer t3.medium, db.t3.lar ge, db.t4g.medium, db.t4g.large.	on
ident_file	Legt die ident-Konfigurationsdatei des Servers fest.	/rdsdbdata/config/ pg_ident.conf
idle_in_transactio n_session_timeout	(ms) Legt die maximal zulässige Dauer von Leerlauftransaktionen fest.	86400000
idle_session_timeout	Beendet jede Sitzung, die länger als die angegebene Zeitspanne im Leerlauf ist (d. h. auf eine Client-Abfrage gewartet hat), sich jedoch nicht innerhalb einer offenen Transakti on befindet	–
intervalstyle	Legt das Anzeigeformat für Intervallwerte fest.	–

Parametername	Beschreibung	Standard
join_collapse_limit	Legt die Größe der FROM-Liste fest, jenseits der JOIN-Konstrukte nicht auf eine Ebene gebracht werden.	–
krb_caseins_users	Legt fest, ob Generic Security Service API (GSSAPI)-Benutzernamen ohne Berücksichtigung von Groß- und Kleinschreibung behandelt werden sollen (true) oder nicht. Standardmäßig ist dieser Parameter auf false festgelegt, sodass Kerberos erwartet, dass bei Benutzernamen zwischen Groß- und Kleinschreibung unterschieden wird. Weitere Informationen finden Sie unter GSSAPI-Authentifizierung in der PostgreSQL-Dokumentation.	false
lc_messages	Legt die Sprache fest, in der Nachrichten angezeigt werden.	–
lc_monetary	Legt das Gebietsschema für die Formatierung von monetären Beträgen fest.	–
lc_numeric	Legt das Gebietsschema für die Formatierung von Zahlen fest.	–
lc_time	Legt das Gebietsschema für die Formatierung von Datum- und Uhrzeitwerten fest.	–
listen_addresses	Legt den Host-Namen oder die IP-Adresse(n) fest, auf die gewartet werden soll.	*
lo_compat_privileges	Aktiviert den Abwärtskompatibilitätsmodus für Berechtigungsprüfungen für große Objekte.	0
log_autovacuum_min_duration	(ms) Legt die Mindestausführungszeit fest, jenseits der Aktionen für die Selbstbereinigung protokolliert werden.	10000

Parametername	Beschreibung	Standard
log_connections	Protokolliert jede erfolgreiche Verbindung.	–
log_destination	Legt das Ziel für die Serverprotokollausgabe fest.	stderr
log_directory	Legt das Zielverzeichnis für Protokolldateien fest.	/rdsdbdata/log/error
log_disconnections	Protokolliert das Ende einer Sitzung einschließlich der Dauer.	–
log_duration	Protokolliert die Dauer jeder abgeschlossenen SQL-Anweisung.	–
log_error_verbosity	Legt die Ausführlichkeit protokollierter Nachrichten fest.	–
log_executor_stats	Schreibt die Leistungsstatistik des Executors in das Serverprotokoll.	–
log_file_mode	Legt die Dateiberechtigungen für Protokolldateien fest.	0644
log_filename	Legt das Dateinamenmuster für Protokolldateien fest.	postgresql.log.%Y-%m-%d-%H%M
logging_collector	Startet einen Unterprozess, um die stderr-Ausgabe und/oder csvlogs in Protokolldateien zu erfassen.	1
log_hostname	Protokolliert den Hostnamen in den Verbindungsprotokollen.	0
logical_decoding_work_mem	(kB) So viel Speicher kann von jedem internen Neuordnungspuffer genutzt werden, bevor er auf die Festplatte geschrieben wird.	–

Parametername	Beschreibung	Standard
log_line_prefix	Steuert Informationen, die jeder Protokollzeile vorangestellt sind.	%t:%r:%u@%d:%p]:
log_lock_waits	Protokolliert lange Sperrenwartezeiten.	–
log_min_duration_sample	(ms) Legt die Mindestausführungszeit fest, jenseits der stichprobenartig Anweisungen protokolliert werden. Das Sampling wird durch log_statement_sample_rate bestimmt.	–
log_min_duration_statement	(ms) Legt die Mindestausführungszeit fest, jenseits der Anweisungen protokolliert werden.	–
log_min_error_statement	Veranlasst, dass alle Anweisungen, die einen Fehler auf oder jenseits dieser Stufe generieren, protokolliert werden.	–
log_min_messages	Legt die Nachrichtenebenen fest, die protokolliert werden.	–
log_parameter_max_length	(B) Beim Protokollieren von Anweisungen werden die protokollierten Parameterwerte auf die ersten N Byte beschränkt.	–
log_parameter_max_length_on_error	(B) Bei der Meldung eines Fehlers werden die protokollierten Parameterwerte auf die ersten N Byte beschränkt.	–
log_parser_stats	Schreibt die Leistungsstatistik des Parsers in das Serverprotokoll.	–
log_planner_stats	Schreibt die Leistungsstatistik des Planers in das Serverprotokoll.	–
log_replication_commands	Protokolliert jeden Replikationsbefehl.	–

Parametername	Beschreibung	Standard
log_rotation_age	(min) Die automatische Protokolldateirotation wird nach N Minuten ausgeführt.	60
log_rotation_size	(kB) Die automatische Protokolldateirotation wird nach N Kilobyte ausgeführt.	100000
log_statement	Legt den Typ der protokollierten Anweisungen fest.	–
log_statement_sample_rate	Anteil der Anweisungen, die log_min_duration_sample überschreiten, die protokolliert werden sollen.	–
log_statement_stats	Schreibt kumulative Leistungsstatistiken in das Serverprotokoll.	–
log_temp_files	(kB) Protokolliert die Verwendung temporärer Dateien, die größer als diese Kilobyte-Angabe sind.	–
log_timezone	Legt die Zeitzone fest, die in Protokoll meldungen verwendet werden soll.	UTC
log_transaction_sample_rate	Legt den Anteil der neuen Transaktionen fest, die protokolliert werden sollen.	–
log_truncate_on_rotation	Kürzt vorhandene Protokolldateien mit demselben Namen während der Protokoll rotation.	0
maintenance_io_concurrency	Eine Variante von effective_io_concurrency, die für Wartungsarbeiten verwendet wird.	1
maintenance_work_mem	(kB) Legt den maximalen Arbeitsspeicher fest, der für Wartungsoperationen verwendet werden darf.	GREATEST(DBInstanceClassMemory/63963136*1024, 65536)

Parametername	Beschreibung	Standard
max_connections	Legt die maximale Anzahl gleichzeitiger Verbindungen fest.	LEAST(DBInstanceClassMemory/9531392,5000)
max_files_per_process	Legt die maximale Anzahl gleichzeitig geöffneter Dateien für die einzelnen Serverprozesse fest.	–
max_locks_per_transaction	Legt die maximale Anzahl von Sperren pro Transaktion fest.	64
max_logical_replication_workers	Maximale Anzahl von Worker-Prozessen für logische Replikation.	–
max_parallel_maintenance_workers	Legt die maximale Anzahl von parallelen Vorgängen pro Wartungsvorgang fest.	–
max_parallel_workers	Legt die maximale Anzahl von parallelen Workers fest, die gleichzeitig aktiv sein können.	GREATEST($DBInstanceVCPU/2,8)
max_parallel_workers_per_gather	Legt die maximale Anzahl von parallelen Prozessen pro Executor-Knoten fest.	–
max_pred_locks_per_page	Legt die maximale Anzahl von prädikatsgesperrten Tupeln pro Seite fest.	–
max_pred_locks_per_relation	Legt die maximale Anzahl von prädikatsgesperrten Seiten und Tupeln pro Beziehung fest.	–
max_pred_locks_per_transaction	Legt die maximale Anzahl von Prädikatssperren pro Transaktion fest.	–
max_prepared_transactions	Legt die maximale Anzahl gleichzeitig vorbereiteter Transaktionen fest.	0

Parametername	Beschreibung	Standard
max_replication_slots	Legt die maximale Anzahl von Replikations-Slots fest, die der Server unterstützen kann.	20
max_slot_wal_keep_size	(MB) Replikations-Slots werden als fehlgeschlagen gekennzeichnet und Segmente werden zum Löschen oder Recycling freigegeben, wenn WAL auf der Festplatte so viel Speicherplatz belegt.	–
max_stack_depth	(kB) Legt die maximale Stack-Tiefe in Kilobytes fest.	6144
max_standby_streaming_delay	(ms) Legt die maximale Verzögerung fest, bevor Abfragen storniert werden, wenn ein Hot Standby-Server gestreamte WAL-Daten verarbeitet.	14000
max_sync_workers_per_subscription	Maximale Anzahl von Synchronisations-Workers pro Abonnement	2
max_wal_senders	Legt die maximale Anzahl gleichzeitig ausgeführter WAL-Senderprozesses fest.	10
max_worker_processes	Legt die maximale Anzahl gleichzeitiger Worker-Prozesse fest.	GREATEST($DBInstanceVCPU*2, 8)
min_dynamic_shared_memory	(MB) Menge des dynamischen gemeinsam genutzten Speichers, der beim Start reserviert wurde.	–
min_parallel_index_scan_size	(8 kB) Legt die Mindestmenge an Indexdaten für einen parallelen Scan fest.	–
min_parallel_table_scan_size	(8 kB) Legt die Mindestmenge an Tabellendaten für einen parallelen Scan fest.	–

Parametername	Beschreibung	Standard
old_snapshot_threshold	(min) Zeit, bis ein Snapshot zu alt ist, um Seiten zu lesen, die nach dem Erstellen des Snapshots geändert wurden.	–
orafce.nls_date_format	Emuliert das Datumsausgabeverhalten von Oracle.	–
orafce.timezone	Gibt die für die Funktion sysdate verwendete Zeitzone an.	–
parallel_leader_participation	Steuert, ob Gather und Gather Merge auch Unterpläne ausführen.	–
parallel_setup_cost	Legt die Kostenschätzung des Planers für das Starten von Worker-Prozessen für parallele Abfragen fest.	–
parallel_tuple_cost	Legt die Kostenschätzung des Planers für die Übergabe von Tupeln (Zeilen) von Worker auf Haupt-Backend fest.	–
password_encryption	Verschlüsselt Passwörter.	–
pgaudit.log	Gibt an, welche Klassen von Anweisungen durch die Sitzungsprüfungs-Protokollierung protokolliert werden.	–
pgaudit.log_catalog	Gibt an, dass die Sitzungsprotokollierung aktiviert werden soll, wenn sich alle Beziehungen in einer Anweisung in pg_catalog befinden.	–
pgaudit.log_level	Gibt die Protokollstufe an, die für Protokolleinträge verwendet wird.	–
pgaudit.log_parameter	Gibt an, dass die Audit-Protokollierung die Parameter enthalten sollte, die mit der Anweisung übergeben wurden.	–

Parametername	Beschreibung	Standard
pgaudit.log_relation	Gibt an, ob die Sitzungsprüfungs-Protokollierung für jede Beziehung (TABLE, VIEW usw.), auf die in einer SELECT- oder DML-Anweisung verwiesen wird, einen separaten Protokolleintrag erstellen soll.	–
pgaudit.log_statement_once	Gibt an, ob die Protokollierung den Anweisungstext und die Parameter mit dem ersten Protokolleintrag für eine Kombination aus Anweisung/Unteranweisung oder bei jedem Eintrag enthält.	–
pgaudit.role	Gibt die Hauptrolle an, die für die Objektüberwachungsprotokollierung verwendet werden soll.	–
pg_bigm.enable_recheck	Gibt an, ob Recheck durchgeführt werden soll, was ein interner Prozess der Volltextsuche ist.	on
pg_bigm.gin_key_limit	Gibt die maximale Anzahl von Bigrammen des Suchschlüsselworts an, das für die Volltextsuche verwendet werden soll.	0
pg_bigm.last_update	Meldet das letzte Aktualisierungsdatum des pg_bigm-Moduls.	2013.11.22
pg_bigm.similarity_limit	Es gibt den Mindestschwellenwert an, der von der Ähnlichkeitssuche verwendet wird.	0.3
pg_hint_plan.debug_print	Protokolliert Ergebnisse der Hinweisanalyse.	–
pg_hint_plan.enable_hint	Zwingt Planer zur Verwendung der Pläne, die im Hinweiskommentar vor der Abfrage angegeben sind.	–

Parametername	Beschreibung	Standard
pg_hint_plan.enable_hint_table	Zwingt Planer, Hinweise nicht über Tabellensuchen zu erhalten.	–
pg_hint_plan.message_level	Nachrichtenebene von Debug-Meldungen.	–
pg_hint_plan.parse_messages	Nachrichtenebene von Analysefehlern.	–
pglogical.batch_inserts	Batch-Inserts wenn möglich	–
pglogical.conflict_log_level	Legt die Protokollstufe gelöster Konflikte fest	–
pglogical.conflict_resolution	Legt die Methode zur Konfliktlösung für lösbare Konflikte fest.	–
pglogical.extra_connection_options	Verbindungsoptionen zum Hinzufügen zu allen Peer-Knotenverbindungen	–
pglogical.synchronous_commit	Spezifischer synchroner Commit-Wert für pglogical	–
pglogical.use_spi	Verwenden Sie SPI anstelle der Low-Level-API zum Anwenden von Änderungen	–
pgtle.clientauth_databases_to_skip	Liste der Datenbanken, die für die Clientauth-Funktion übersprungen werden sollen	–
pgtle.clientauth_db_name	Steuert, welche Datenbank für die Clientauth-Funktion verwendet wird	–
pgtle.clientauth_num_parallel_workers	Anzahl der Hintergrund-Worker, die für die Clientauth-Funktion verwendet werden	–
pgtle.clientauth_users_to_skip	Liste der Benutzer, die für die Clientauth-Funktion übersprungen werden sollen	–

Parametername	Beschreibung	Standard
pgtle.enable_clientauth	Aktiviert die Clientauth-Funktion	–
pgtle.passcheck_db_name	Legt fest, welche Datenbank für die Cluster-weite Passcheck-Funktion verwendet wird	–
pg_prewarm.autoprewarm	Startet den autoprewarm-Worker.	–
pg_prewarm.autoprewarm_interval	Legt das Intervall zwischen Sicherungen freigegebener Puffer fest	–
pg_similarity.block_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.block_threshold	Legt den Schwellenwert fest, der von der Block-Ähnlichkeitsfunktion verwendet wird.	–
pg_similarity.block_tokenizer	Legt den Tokenizer für die Block-Ähnlichkeitsfunktion fest.	–
pg_similarity.cosine_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.cosine_threshold	Legt den Schwellenwert fest, der von der Kosinus-Ähnlichkeitsfunktion verwendet wird.	–
pg_similarity.cosine_tokenizer	Legt den Tokenizer für die Kosinus-Ähnlichkeitsfunktion fest.	–
pg_similarity.dice_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.dice_threshold	Legt den Schwellenwert fest, der vom Dice-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.dice_tokenizer	Legt den Tokenizer für das Dice-Ähnlichkeitsmaß fest.	–

Parametername	Beschreibung	Standard
pg_similarity.euclidean_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.euclidean_threshold	Legt den Schwellenwert fest, der vom euklidischen Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.euclidean_tokenizer	Legt den Tokenizer für das euklidische Ähnlichkeitsmaß fest.	–
pg_similarity.hamming_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.hamming_threshold	Legt den Schwellenwert fest, der von der Block-Ähnlichkeitsmetrik verwendet wird.	–
pg_similarity.jaccard_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.jaccard_threshold	Legt den Schwellenwert fest, der vom Jaccard-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.jaccard_tokenizer	Legt den Tokenizer für das Jaccard-Ähnlichkeitsmaß fest.	–
pg_similarity.jaro_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.jaro_threshold	Legt den Schwellenwert fest, der vom Jaro-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.jarowinkler_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.jarowinkler_threshold	Legt den Schwellenwert fest, der vom Jaro-Winkler-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.levenshtein_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–

Parametername	Beschreibung	Standard
pg_similarity.levenshtein_threshold	Legt den Schwellenwert fest, der vom Levenshtein-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.matching_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.matching_threshold	Legt den Schwellenwert fest, der vom Matching-Koeffizient-Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.matching_tokenizer	Legt den Tokenizer für das Matching-Koeffizient-Ähnlichkeitsmaß fest.	–
pg_similarity.mongeelkan_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.mongeelkan_threshold	Legt den Schwellenwert fest, der vom Monge-Elkan-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.mongeelkan_tokenizer	Legt den Tokenizer für das Monge-Elkan-Ähnlichkeitsmaß fest.	–
pg_similarity.nw_gap_penalty	Legt den Lückenabzug fest, der vom Ähnlichkeitsmaß Needleman-Wunsch verwendet wird.	–
pg_similarity.nw_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.nw_threshold	Legt den Schwellenwert fest, der vom Ähnlichkeitsmaß Needleman-Wunsch verwendet wird.	–
pg_similarity.overlap_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–

Parametername	Beschreibung	Standard
pg_similarity.overlap_threshold	Legt den Schwellenwert fest, der vom Überschneidungskoeffizient-Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.overlap_tokenizer	Legt den Tokenizer für das Überschneidungskoeffizient-Ähnlichkeitsmaß fest.	–
pg_similarity.qgram_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.qgram_threshold	Legt den Schwellenwert fest, der vom Q-Gram-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.qgram_tokenizer	Legt den Tokenizer für die Q-Gram-Messung fest.	–
pg_similarity.swg_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.swg_threshold	Legt den Schwellenwert fest, der vom Smith-Waterman-Gotoh-Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.sw_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.sw_threshold	Legt den Schwellenwert fest, der vom Smith-Waterman-Ähnlichkeitsmaß verwendet wird	–
pg_stat_statements.max	Legt die maximale Anzahl von Anweisungen fest, die von pg_stat_statements verfolgt werden.	–
pg_stat_statements.save	Speichert pg_stat_statements-Statistiken über Serverausfälle hinweg.	–

Parametername	Beschreibung	Standard
pg_stat_statements.track	Wählt aus, welche Anweisungen von pg_stat_statements verfolgt werden.	–
pg_stat_statements.track_planning	Wählt aus, ob die Planungsdauer von pg_stat_statements verfolgt wird.	–
pg_stat_statements.track_utility	Wählt aus, ob Dienstprogrammbefehle von pg_stat_statements verfolgt werden.	–
plan_cache_mode	Steuert die Planerauswahl des benutzerdefinierten oder generischen Plans.	–
port	Legt den TCP-Port fest, den der Server überwacht.	Endpunkt-Port
postgis.gdal_enabled_drivers	Aktiviert oder deaktiviert GDAL-Treiber, die mit PostGIS in Postgres 9.3.5 und höher verwendet werden.	ENABLE_ALL
quote_all_identifiers	Fügt beim Generieren von SQL-Fragmenten allen Bezeichnern Anführungszeichen hinzu.	–
random_page_cost	Legt die Schätzung des Planers für die Kosten einer nicht sequenziell abgerufenen Datenträgerseite fest.	–
rdkit.dice_threshold	Unterer Schwellenwert der Dice-Ähnlichkeit. Moleküle mit einer Ähnlichkeit unter dem Schwellenwert sind durch die Operation # nicht ähnlich.	–
rdkit.do_chiral_sss	Ob Stereochemie beim Unterstrukturabgleich berücksichtigt wird. Bei false werden keine Stereochemie-Informationen für den Unterstrukturabgleich verwendet.	–

Parametername	Beschreibung	Standard
rdkit.tanimoto_threshold	Unterer Schwellenwert der Tanimoto-Ähnlichkeit. Moleküle mit einer Ähnlichkeit unter dem Schwellenwert sind durch die Operation % nicht ähnlich.	–
rds.accepted_password_auth_method	Erzwingt die Authentifizierung für Verbindungen mit lokal gespeichertem Passwort.	md5+scram
rds.adaptive_autovacuum	RDS-Parameter zum Aktivieren/Deaktivieren der adaptiven Selbstbereinigung.	1
rds.babelfish_status	RDS-Parameter zum Aktivieren/Deaktivieren von Babelfish-for-Aurora-PostgreSQL.	aus
rds.enable_plan_management	Aktivieren oder deaktivieren der apg_plan_mgmt-Erweiterung.	0

Parametername	Beschreibung	Standard
rds.extensions	Liste von Erweiterungen, die RDS zur Verfügung stellt	address_standardizer, address_standardizer_data_us, apg_plan_mgmt, aurora_stat_utils, amcheck, autoinc, aws_commons, aws_ml, aws_s3, aws_lambda, bool_plperl, bloom, btree_gin, btree_gist, citext, cube, dblink, dict_int, dict_xsyn, earthdistance, fuzzystrmatch, hll, hstore, hstore_plperl, insert_username, intagg, intarray, ip4r, isn, jsonb_plperl, lo, log_fdw, ltree, moddatetime, old_snapshot, oracle_fdw, orafce, pgaudit, pgcrypto, pglogical, pgrouting, pgrowlocks, pgstattuple, pgtap, pg_bigm, pg_buffercache, pg_cron, pg_freespacemap, pg_hint_plan, pg_partman, pg_prewarm, pg_proctab, pg_repack, pg_simila

Parametername	Beschreibung	Standard
		rity, pg_stat_s tatements, pg_trgm, pg_visibility, plcoffee, plls, plperl, plpgsql, plprofiler, pltcl, plv8, postgis, postgis_t iger_geocoder, postgis_raster, postgis_topology, postgres_fdw, prefix, rdkit, rds_tools, refint, sslinfo, tablefunc, tds_fdw, test_parser, tsm_system_rows, tsm_system_time, unaccent, uuid-ossp
rds.force_admin_lo gging_level	Siehe Protokollmeldungen für RDS-Admin-Benutzeraktionen in Kundendatenbanken.	–
rds.force_autovacu um_logging_level	Siehe Protokollmeldungen zu Selbstber einigungsvorgängen.	WARNING
rds.force_ssl	SSL-Verbindungen erzwingen.	0

Parametername	Beschreibung	Standard
rds.global_db_rpo	(s) Der Schwellenwert für das Recovery Point Objective in Sekunden, der bei Überschre itungen Commits von Benutzern blockiert. **Important** Dieser Parameter ist für Aurora-Po stgreSQL-basierte globale Datenbank en vorgesehen. Belassen Sie für eine nicht-globale Datenbank den Standardwert. Weitere Informationen zur Nutzung dieses Parameters finden Sie unter the section called “Verwalten von RPOs (Aurora PostgreSQL)”.	–
rds.logical_replication	Aktiviert die logische Dekodierung.	0
rds.logically_repl icate_unlogged_tables	Nicht protokollierte Tabellen werden logisch repliziert.	1
rds.log_retention_ period	Amazon RDS löscht PostgreSQL-Protokolle, die älter als N Minuten sind.	4320
rds.pg_stat_ramdis k_size	Größe der Ramdisk-Statistiken in MB. Ein Wert ungleich Null richtet die Ramdisk ein. Dieser Parameter ist nur für Aurora-PostgreSQL-14-Versionen und niedriger verfügbar.	0
rds.rds_superuser_ reserved_connections	Legt die Anzahl der Verbindungs-Slots fest, die für rds_superusers reserviert sind. Dieser Parameter ist nur für Version 15 und niedriger verfügbar. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation unter Reservierte Verbindungen.	2

Parametername	Beschreibung	Standard
rds.restrict_password_commands	Beschränkt passwortbezogene Befehle auf Mitglieder von rds_password	–
rds.superuser_variables	Liste der reinen Superuser-Variablen, für die wir die Änderungsanweisungen von rds_superuser erhöhen.	session_replication_role
recovery_init_sync_method	Legt die Methode zum Synchronisieren des Datenverzeichnisses vor der Wiederherstellung nach Abstürzen fest.	syncfs
remove_temp_files_after_crash	Entfernt temporäre Dateien nach dem Backend-Absturz.	0
restart_after_crash	Initialisiert den Server nach dem Backend-Absturz neu.	–
row_security	Aktiviert Zeilensicherheit.	–
search_path	Legt die Schemasuchreihenfolge für Namen fest, die nicht schemaqualifiziert sind.	–
seq_page_cost	Legt die Schätzung des Planers für die Kosten einer sequenziell abgerufenen Datenträgerseite fest.	–
session_replication_role	Legt das Sitzungsverhalten für Auslöser und Neuschreibungsregeln fest.	–
shared_buffers	(8 kB) Legt die maximale Anzahl freigegebener Arbeitsspeicherpuffer fest, die vom Server verwendet werden.	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	Listet freigegebene Bibliotheken auf, die in den Server vorgeladen werden.	pg_stat_statements
ssl	Ermöglicht SSL-Verbindungen.	1

Parametername	Beschreibung	Standard
ssl_ca_file	Speicherort der SSL-Server-Zertifizierungss tellendatei.	/rdsdbdata/rds-met adata/ca-cert.pem
ssl_cert_file	Speicherort der SSL-Serverzertifikatdatei.	/rdsdbdata/rds-met adata/server-cert.pem
ssl_ciphers	Legt die Liste der zulässigen TLS-Versc hlüsselungen fest, die für sichere Verbindungen verwendet werden sollen.	–
ssl_crl_dir	Speicherort des Verzeichnisses der SSL-Zerti fikatsperrliste.	/rdsdbdata/rds-met adata/ssl_crl_dir/
ssl_key_file	Speicherort der privaten Schlüsseldatei des SSL-Servers	/rdsdbdata/rds-met adata/server-key.pem
ssl_max_protocol_v ersion	Legt die maximal zulässige SSL-/TLS-Protokoll version fest	–
ssl_min_protocol_v ersion	Legt die zulässige Mindestversion des SSL-/ TLS-Protokolls fest	TLSv1.2
standard_conformin g_strings	Veranlasst Zeichenfolgen „...“, Backslashes als Zeichen zu behandeln.	–
statement_timeout	(ms) Legt die maximal zulässige Dauer von Anweisungen fest.	–
stats_temp_directory	Schreibt temporäre Statistikdateien in das angegebene Verzeichnis.	/rdsdbdata/db/pg_s tat_tmp
superuser_reserved _connections	Legt die Anzahl der Verbindungs-Slots fest, die für Superusers reserviert sind.	3
synchronize_seqsca ns	Ermöglicht synchronisierte sequenzielle Prüfungen.	–

Parametername	Beschreibung	Standard
synchronous_commit	Legt die Synchronisierungsstufe aktueller Transaktionen fest.	on
tcp_keepalives_count	Maximale Anzahl von TCP-Keepalive-Neuü bertragungen.	–
tcp_keepalives_idle	(s) Zeit zwischen der Ausgabe von TCP-Keepa lives.	–
tcp_keepalives_int erval	(s) Zeit zwischen der Ausgabe von TCP-Keepa live-Neuübertragungen.	–
temp_buffers	(8 kB) Legt die maximale Anzahl der temporäre n Puffer fest, die von den einzelnen Sitzungen verwendet werden.	–
temp_file_limit	Schränkt den Gesamtspeicherplatz in Kilobyte ein, den ein bestimmter PostgreSQL-Prozess für temporäre Dateien verwenden kann, ausschließlich von Speicherplatz, der für explizite temporäre Tabellen verwendet wird	-1
temp_tablespaces	Legt die Tabellenräume fest, die für temporäre Tabellen und Sortierdateien verwendet werden sollen.	–
Zeitzone	Legt die Zeitzone für die Anzeige und Interpret ation von Zeitstempeln fest.	UTC
track_activities	Sammelt Informationen zu Befehlen, die ausgeführt werden.	–
track_activity_que ry_size	Legt die für pg_stat_activity.current_query reservierte Größe in Bytes fest.	4096
track_commit_times tamp	Erfasst den Commit-Zeitpunkt der Transaktion.	–

Parametername	Beschreibung	Standard
track_counts	Sammelt Statistiken zur Datenbankaktivität.	–
track_functions	Sammelt Statistiken auf Funktionsebene zur Datenbankaktivität.	pl
track_io_timing	Erfasst Zeitpunktstatistiken zur Datenbank-E/A-Aktivität.	1
track_wal_io_timing	Erfasst Zeitpunktstatistiken zur WAL-E/A-A ktivität.	–
transform_null_equals	Behandelt expr=NULL als expr IS NULL.	–
update_process_title	Aktualisiert den Titel des Vorgangs, um den aktiven SQL-Befehl anzuzeigen.	–
vacuum_cost_delay	(ms) Bereinigungskostenverzögerung in Millisekunden.	–
vacuum_cost_limit	Bereinigungskostenbetrag, der vor der Inaktivit ät verfügbar ist.	–
vacuum_cost_page_d irty	Bereinigungskosten für eine Seite, die durch eine Bereinigung ungültig wurde.	–
vacuum_cost_page_h it	Bereinigungskosten für eine Seite, die im Puffer-Cache gefunden wurde.	–
vacuum_cost_page_m iss	Bereinigungskosten für eine Seite, die nicht im Puffer-Cache gefunden wurde.	0
vacuum_defer_clean up_age	Anzahl der Transaktionen, um die VACUUM und HOT Cleanup aufgeschoben werden sollen, wenn vorhanden.	–
vacuum_failsafe_age	Alter, bei dem der VACUUM-Vorgang die Ausfallsicherheit auslösen sollte, um einen Wraparound-Ausfall zu vermeiden.	1200000000

Parametername	Beschreibung	Standard
vacuum_freeze_min_age	Mindestalter, bei dem der VACUUM-Vorgang eine Tabellenzeile eingefroren werden sollte.	–
vacuum_freeze_table_age	Alter, bei dem der VACUUM-Vorgang eine Tabelle vollständig scannen sollte, um Tupel einzufrieren.	–
vacuum_multixact_failsafe_age	Multixact-Alter, bei dem der VACUUM-Vorgang die Ausfallsicherheit auslösen sollte, um einen Wraparound-Ausfall zu vermeiden.	1200000000
vacuum_multixact_freeze_min_age	Mindestalter, bei dem der VACUUM-Vorgang eine MultiXactId in einer Tabellenzeile einfrieren sollte.	–
vacuum_multixact_freeze_table_age	Multixact-Alter, bei dem der VACUUM-Vorgang eine Tabelle vollständig scannen sollte, um Tupel einzufrieren.	–
wal_buffers	(8 kB) Legt die Anzahl von Datenträger-Seiten puffern im freigegebenen Arbeitsspeicher für WAL fest.	–
wal_receiver_create_temp_slot	Legt fest, ob ein WAL-Receiver einen temporären Replikations-Slot erstellen soll, wenn kein permanenter Slot konfiguriert ist.	0
wal_receiver_status_interval	(s) Legt das maximale Intervall zwischen den Statusberichten des WAL-Receivers auf den primären Wert fest.	–
wal_receiver_timeout	(ms) Legt die maximale Wartezeit fest, um Daten von der Primär-Instance zu empfangen.	30000
wal_sender_timeout	(ms) Legt fest, wie lange höchstens auf die WAL-Replikation gewartet wird.	–

Parametername	Beschreibung	Standard
work_mem	(kB) Legt den maximalen Arbeitsspeicher fest, der für Abfrage-Workspaces verwendet werden darf.	–
xmlbinary	Legt die Kodierung von Binärwerten in XML fest.	–
xmloption	Legt fest, ob XML-Daten in impliziten Parsing- und Serialisierungsoperationen als Dokumente oder Inhaltsfragmente betrachtet werden sollen.	–

Aurora-PostgreSQL-Parameter auf Instance-Ebene

Sie können die Parameter auf Instance-Ebene für eine bestimmte Aurora-PostgreSQL-Version mithilfe der AWS-Managementkonsole, der AWS CLI oder Amazon-RDS-API anzeigen. Weitere Informationen zum Anzeigen von Parametern in einer DB-Parametergruppe von Aurora PostgreSQL finden Sie unter Anzeigen von Parameterwerten für eine DB-Parametergruppe in Amazon Aurora.

Einige Parameter auf Instance-Ebene sind nicht in allen Versionen verfügbar und manche sind veraltet. Hinweise zum Anzeigen der Parameter einer bestimmten Aurora-PostgreSQL-Version finden Sie unter Anzeigen von Aurora-PostgreSQL-DB-Cluster- und DB-Parametern.

In der folgenden Tabelle werden beispielsweise alle Parameter aufgeführt, die für eine bestimmte DB-Instance in einem DB-Cluster von Aurora PostgreSQL gelten. Diese Liste wurde erzeugt, indem der AWS CLI-Befehl describe-db-parameters mit `default.aurora-postgresql14` für den Wert `--db-parameter-group-name` ausgeführt wurde.

Eine Auflistung der DB-Cluster-Parameter für dieselbe Standard-DB-Parametergruppe finden Sie unter Aurora-PostgreSQL-Parameter auf Cluster-Ebene.

Parametername	Beschreibung	Standard
apg_enable_batch_mode_function_execution	Aktiviert Batch-Modus-Funktionen, um mehrere Zeilen gleichzeitig zu verarbeiten.	–
apg_enable_correlated_any_transform	Ermöglicht es dem Planer, korrelierte ANY-Sublinks (IN/NOT IN Unterabfrage) nach Möglichkeit in JOIN umzuwandeln.	–
apg_enable_function_migration	Ermöglicht dem Planer, berechtigte Skalarfunktionen in die FROM-Klausel zu migrieren.	–
apg_enable_not_in_transform	Ermöglicht es dem Planer, NOT IN-Unterabfragen nach Möglichkeit in ANTI JOIN umzuwandeln.	–
apg_enable_remove_redundant_inner_joins	Ermöglicht dem Planer, redundante Inner Joins zu entfernen.	–

Parametername	Beschreibung	Standard
apg_enable_semijoin_push_down	Ermöglicht die Verwendung von Semijoin-Filtern für Hash-Joins.	–
apg_plan_mgmt.capture_plan_baselines	Baseline-Modus für Planerfassung. Manuell – Planerfassung für jede SQL-Anweisung aktivieren. Aus – Planerfassung deaktivieren. Automatisch – Planerfassung für Anweisungen in pg_stat_statement aktivieren, die die Berechtigungskriterien erfüllen.	aus
apg_plan_mgmt.max_databases	Legt die maximale Anzahl von Datenbanken fest, die Abfragen mit apg_plan_mgmt verwalten können.	10
apg_plan_mgmt.max_plans	Legt die maximale Anzahl von Plänen fest, die von apg_plan_mgmt zwischengespeichert werden können.	10000
apg_plan_mgmt.plan_retention_period	Maximale Anzahl von Tagen, seit ein Plan zuletzt verwendet wurde, bevor ein Plan automatisch gelöscht wird.	32
apg_plan_mgmt.unapproved_plan_execution_threshold	Geschätzte Gesamtplankosten, unter denen ein nicht genehmigter Plan ausgeführt wird.	0
apg_plan_mgmt.use_plan_baselines	Verwenden Sie nur genehmigte oder feste Pläne für verwaltete Anweisungen.	false
application_name	Legt den Namen der Anwendung fest, der in Statistiken und Protokollen verwendet werden soll.	–

Parametername	Beschreibung	Standard
aurora_compute_plan_id	Überwacht Ausführungspläne von Abfragen, um die Ausführungspläne zu ermitteln, die zur aktuellen Datenbanklast beitragen, und um Leistungsstatistiken der Ausführungspläne im Laufe der Zeit zu verfolgen Weitere Informationen finden Sie unter Überwachen von Abfrageausführungsplänen für Aurora PostgreSQL.	on
aurora_temp_space_size	(MB) Legt die Größe des zugewiesenen Speicherplatzes für temporäre Objekte mit aktivierten optimierten Lesevorgängen auf E/A-optimierten Aurora-Clustern mit unterstützten Instance-Klassen fest	DBInstanceClassMemory/524288
authentication_timeout	(s) Legt die Zeit fest, die maximal zulässig ist, um die Client-Authentifizierung durchzuführen.	–
auto_explain.log_analyze	Verwenden Sie EXPLAIN ANALYZE zur Planprotokollierung.	–
auto_explain.log_buffers	Protokollieren der Puffernutzung.	–
auto_explain.log_format	EXPLAIN-Format, das für die Planprotokollierung verwendet werden soll.	–
auto_explain.log_min_duration	Legt die Mindestausführungszeit fest, ab der Pläne protokolliert werden.	–
auto_explain.log_nested_statements	Protokollieren verschachtelter Anweisungen.	–
auto_explain.log_timing	Erfasst Timing-Daten, nicht nur Zeilenzahlen.	–

Parametername	Beschreibung	Standard
auto_explain.log_triggers	Fügt Auslöserstatistiken in Pläne ein.	–
auto_explain.log_verbose	Verwenden Sie EXPLAIN VERBOSE zur Planprotokollierung.	–
auto_explain.sample_rate	Bruchteil der zu verarbeitenden Abfragen.	–
babelfishpg_tds.listen_adressen	Legt den Host-Namen oder die IP-Adresse(n) fest, die auf TDS überwacht werden sollen.	*
babelfishpg_tds.tds_debug_log_level	Legt die Protokollierungsstufe in TDS fest, 0 deaktiviert die Protokollierung	1
backend_flush_after	(8 Kb) Anzahl der Seiten, nach denen zuvor ausgeführte Schreibvorgänge auf die Festplatte geschrieben werden.	–
bytea_output	Legt das Ausgabeformat für bytea fest.	–
check_function_bodies	Überprüft die Funktionstexte während CREATE FUNCTION.	–
client_connection_check_interval	Legt das Zeitintervall zwischen Prüfungen auf Verbindungsabbrüche während der Ausführung von Abfragen fest.	–
client_min_messages	Legt die Nachrichtenebenen fest, die an den Client gesendet werden.	–
config_file	Legt die Hauptkonfigurationsdatei des Servers fest.	/rdsdbdata/config/postgresql.conf
constraint_exclusion	Ermöglicht dem Planer die Verwendung von Einschränkungen, um Abfragen zu optimieren.	–

Parametername	Beschreibung	Standard
cpu_index_tuple_cost	Legt die Schätzung des Planers für die Kosten der Verarbeitung der einzelnen Indexeinträge während einer Indexprüfung fest.	–
cpu_operator_cost	Legt die Schätzung des Planers für die Kosten der Verarbeitung der einzelnen Operator- oder Funktionsaufrufe fest.	–
cpu_tuple_cost	Legt die Schätzung des Planers für die Kosten der Verarbeitung der einzelnen Tupeln (Zeilen) fest.	–
cron.database_name	Legt die Datenbank fest, um pg_cron-M etadatentabellen zu speichern	postgres
cron.log_run	Protokolliert alle Aufträge in der Tabelle job_run_details	on
cron.log_statement	Protokolliert alle Cron-Anweisungen vor der Ausführung.	aus
cron.max_running_jobs	Die maximale Anzahl von Aufträgen, die gleichzeitig ausgeführt werden können.	5
cron.use_background_workers	Aktiviert Hintergrund-Workers für pg_cron	on
cursor_tuple_fraction	Legt die Schätzung des Planers für den Bruchteil der Zeilen eines Cursors fest, die abgerufen werden.	–
db_user_namespace	Aktiviert Benutzernamen pro Datenbank.	–
deadlock_timeout	(ms) Legt die Zeit fest, die während einer Sperre gewartet wird, bevor auf einen Deadlock geprüft wird.	–

Parametername	Beschreibung	Standard
debug_pretty_print	Erstellt Einschübe für Analyse- und Planstruk turanzeigen.	–
debug_print_parse	Protokolliert die Analysestruktur der einzelnen Abfragen.	–
debug_print_plan	Protokolliert den Ausführungsplan der einzelnen Abfragen.	–
debug_print_rewritten	Protokolliert die neu geschriebene Analysest ruktur der einzelnen Abfragen.	–
default_statistics _target	Legt das Standardstatistikziel fest.	–
default_transactio n_deferrable	Legt den Standardaufschiebbarkeitsstatus neuer Transaktionen fest.	–
default_transactio n_isolation	Legt die Transaktionsisolierungsstufe jeder neuen Transaktion fest.	–
default_transactio n_read_only	Legt den Standardschreibschutzstatus neuer Transaktionen fest.	–
effective_cache_size	(8 kB) Legt die Annahme des Planers hinsichtl ich der Größe des Datenträger-Caches fest.	SUM(DBInstanceClas sMemory/12038,-500 03
effective_io_concu rrency	Die Anzahl der gleichzeitigen Anfragen, die durch das Datenträgersubsystem effizient bearbeitet werden können.	–
enable_async_append	Ermöglicht den Planern die Verwendung paralleler Append-Pläne.	–
enable_bitmapscan	Ermöglicht die Verwendung von Bitmap-Pr üfungsplänen durch den Planer.	–

Parametername	Beschreibung	Standard
enable_gathermerge	Ermöglicht die Verwendung von Gather-Merge-Plänen durch den Planer.	–
enable_hashagg	Ermöglicht die Verwendung von Hash-Aggr egationsplänen durch den Planer.	–
enable_hashjoin	Ermöglicht die Verwendung von Hash-Join-Plänen durch den Planer.	–
enable_incremental _sort	Ermöglicht den Planern die Verwendung inkrementeller Sortierschritte.	–
enable_indexonlyscan	Ermöglicht die Verwendung von reinen Indexprüfungsplänen durch den Planer.	–
enable_indexscan	Ermöglicht die Verwendung von Indexprüf ungsplänen durch den Planer.	–
enable_material	Ermöglicht die Verwendung von Materiali sierung durch den Planer.	–
enable_memoize	Ermöglicht den Planern die Verwendung der Speicherung	–
enable_mergejoin	Ermöglicht die Verwendung von Merge-Join-Plänen durch den Planer.	–
enable_nestloop	Ermöglicht die Verwendung von Join-Plänen mit verschachtelten Schleifen durch den Planer.	–
enable_parallel_ap pend	Ermöglicht den Planern die Verwendung paralleler Append-Pläne.	–
enable_parallel_hash	Ermöglicht den Planern die Verwendung paralleler Hash-Pläne.	–
enable_partition_p runing	Aktivieren Sie das Bereinigen von Planzeit- und Laufzeitpartitionen.	–

Parametername	Beschreibung	Standard
enable_partitionwise_aggregate	Ermöglicht die partitionsweise Aggregation und Gruppierung.	–
enable_partitionwise_join	Aktiviert partitionsweise Joins.	–
enable_seqscan	Ermöglicht die Verwendung von sequenziellen Prüfungsplänen durch den Planer.	–
enable_sort	Ermöglicht die Verwendung von expliziten Sortierschritten durch den Planer.	–
enable_tidscan	Ermöglicht die Verwendung von TID-Prüfungsplänen durch den Planer.	–
escape_string_warning	Warnt vor Escape-Notierungen mit Backslash in gewöhnlichen Zeichenfolgeliteralen.	–
exit_on_error	Beendet die Sitzung bei einem Fehler.	–
force_parallel_mode	Erzwingt die Verwendung paralleler Abfrageeinrichtungen.	–
from_collapse_limit	Legt die Größe der FROM-Liste fest, jenseits der Unterabfragen nicht ausgeblendet werden.	–
geqo	Ermöglicht die genetische Abfrageoptimierung.	–
geqo_effort	GEQO: Der Aufwand, der verwendet wird, um den Standard für andere GEQO-Parameter festzulegen.	–
geqo_generations	GEQO: Die Zahl der Iterationen des Algorithmus.	–
geqo_pool_size	GEQO: Die Anzahl der Personen in der Population.	–

Parametername	Beschreibung	Standard
geqo_seed	GEQO: Der Seed für die zufällige Pfadauswahl.	–
geqo_selection_bias	GEQO: Selektiver Druck innerhalb der Population.	–
geqo_threshold	Legt den Schwellenwert für FROM-Elemente fest, jenseits derer GEQO verwendet wird.	–
gin_fuzzy_search_limit	Legt das maximal zulässige Ergebnis für die exakte Suche durch GIN fest.	–
gin_pending_list_limit	(kB) Legt die maximale Größe der ausstehen den Liste für den GIN-Index fest.	–
hash_mem_multiplier	Vielfaches von work_mem für Hash-Tabellen.	–
hba_file	Legt die hba-Konfigurationsdatei des Servers fest.	/rdsdbdata/config/ pg_hba.conf
hot_standby_feedback	Ermöglicht Feedback von einem Hot Standby zum primären, um Abfragekonflikte zu vermeiden.	on
ident_file	Legt die ident-Konfigurationsdatei des Servers fest.	/rdsdbdata/config/ pg_ident.conf
idle_in_transactio n_session_timeout	(ms) Legt die maximal zulässige Dauer von Leerlauftransaktionen fest.	86400000
idle_session_timeout	Beendet jede Sitzung, die länger als die angegebene Zeitspanne im Leerlauf ist (d. h. auf eine Client-Abfrage gewartet hat), sich jedoch nicht innerhalb einer offenen Transakti on befindet	–

Parametername	Beschreibung	Standard
join_collapse_limit	Legt die Größe der FROM-Liste fest, jenseits der JOIN-Konstrukte nicht auf eine Ebene gebracht werden.	–
lc_messages	Legt die Sprache fest, in der Nachrichten angezeigt werden.	–
listen_addresses	Legt den Host-Namen oder die IP-Adresse(n) fest, auf die gewartet werden soll.	*
lo_compat_privileges	Aktiviert den Abwärtskompatibilitätsmodus für Berechtigungsprüfungen für große Objekte.	0
log_connections	Protokolliert jede erfolgreiche Verbindung.	–
log_destination	Legt das Ziel für die Serverprotokollausgabe fest.	stderr
log_directory	Legt das Zielverzeichnis für Protokolldateien fest.	/rdsdbdata/log/error
log_disconnections	Protokolliert das Ende einer Sitzung einschlie ßlich der Dauer.	–
log_duration	Protokolliert die Dauer jeder abgeschlossenen SQL-Anweisung.	–
log_error_verbosity	Legt die Ausführlichkeit protokollierter Nachricht en fest.	–
log_executor_stats	Schreibt die Leistungsstatistik des Executors in das Serverprotokoll.	–
log_file_mode	Legt die Dateiberechtigungen für Protokoll dateien fest.	0644
log_filename	Legt das Dateinamenmuster für Protokoll dateien fest.	postgresql.log.%Y-%m-%d-%H%M

Parametername	Beschreibung	Standard
logging_collector	Startet einen Unterprozess, um die stderr-Ausgabe und/oder csvlogs in Protokolldateien zu erfassen.	1
log_hostname	Protokolliert den Hostnamen in den Verbindungsprotokollen.	0
logical_decoding_work_mem	(kB) So viel Speicher kann von jedem internen Neuordnungspuffer genutzt werden, bevor er auf die Festplatte geschrieben wird.	–
log_line_prefix	Steuert Informationen, die jeder Protokollzeile vorangestellt sind.	%t:%r:%u@%d:%p]:
log_lock_waits	Protokolliert lange Sperrenwartezeiten.	–
log_min_duration_sample	(ms) Legt die Mindestausführungszeit fest, jenseits der Anweisungen stichprobenartig protokolliert werden. Das Sampling wird durch log_statement_sample_rate bestimmt.	–
log_min_duration_statement	(ms) Legt die Mindestausführungszeit fest, jenseits der Anweisungen protokolliert werden.	–
log_min_error_statement	Veranlasst, dass alle Anweisungen, die einen Fehler auf oder jenseits dieser Stufe generieren, protokolliert werden.	–
log_min_messages	Legt die Nachrichtenebenen fest, die protokolliert werden.	–
log_parameter_max_length	(B) Beim Protokollieren von Anweisungen werden die protokollierten Parameterwerte auf die ersten N Byte beschränkt.	–

Parametername	Beschreibung	Standard
log_parameter_max_length_on_error	(B) Bei der Meldung eines Fehlers werden die protokollierten Parameterwerte auf die ersten N Byte beschränkt.	–
log_parser_stats	Schreibt die Leistungsstatistik des Parsers in das Serverprotokoll.	–
log_planner_stats	Schreibt die Leistungsstatistik des Planers in das Serverprotokoll.	–
log_replication_commands	Protokolliert jeden Replikationsbefehl.	–
log_rotation_age	(min) Die automatische Protokolldateirotation wird nach N Minuten ausgeführt.	60
log_rotation_size	(kB) Die automatische Protokolldateirotation wird nach N Kilobyte ausgeführt.	100000
log_statement	Legt den Typ der protokollierten Anweisungen fest.	–
log_statement_sample_rate	Anteil der Anweisungen, die log_min_duration_sample überschreiten, die protokolliert werden sollen.	–
log_statement_stats	Schreibt kumulative Leistungsstatistiken in das Serverprotokoll.	–
log_temp_files	(kB) Protokolliert die Verwendung temporärer Dateien, die größer als diese Kilobyte-Angabe sind.	–
log_timezone	Legt die Zeitzone fest, die in Protokoll meldungen verwendet werden soll.	UTC

Parametername	Beschreibung	Standard
log_truncate_on_rotation	Kürzt vorhandene Protokolldateien mit demselben Namen während der Protokoll rotation.	0
maintenance_io_concurrency	Eine Variante von effective_io_concurrency, die für Wartungsarbeiten verwendet wird.	1
maintenance_work_mem	(kB) Legt den maximalen Arbeitsspeicher fest, der für Wartungsoperationen verwendet werden darf.	GREATEST(DBInstanceClassMemory/63963136*1024, 65536
max_connections	Legt die maximale Anzahl gleichzeitiger Verbindungen fest.	LEAST(DBInstanceClassMemory/9531392, 5000
max_files_per_process	Legt die maximale Anzahl gleichzeitig geöffneter Dateien für die einzelnen Serverprozesse fest.	–
max_locks_per_transaction	Legt die maximale Anzahl von Sperren pro Transaktion fest.	64
max_parallel_maintenance_workers	Legt die maximale Anzahl von parallelen Vorgängen pro Wartungsvorgang fest.	–
max_parallel_workers	Legt die maximale Anzahl von parallelen Workers fest, die gleichzeitig aktiv sein können.	GREATEST($DBInstanceVCPU/2, 8
max_parallel_workers_per_gather	Legt die maximale Anzahl von parallelen Prozessen pro Executor-Knoten fest.	–
max_pred_locks_per_page	Legt die maximale Anzahl von prädikats gesperrten Tupeln pro Seite fest.	–

Parametername	Beschreibung	Standard
max_pred_locks_per_relation	Legt die maximale Anzahl von prädikats gesperrten Seiten und Tupeln pro Beziehung fest.	–
max_pred_locks_per_transaction	Legt die maximale Anzahl von Prädikatssperren pro Transaktion fest.	–
max_slot_wal_keep_size	(MB) Replikationsslots werden als fehlgesch lagen gekennzeichnet und Segmente zum Löschen oder Recycling freigegeben, wenn WAL auf der Festplatte so viel Speicherplatz belegt.	–
max_stack_depth	(kB) Legt die maximale Stack-Tiefe in Kilobyte fest.	6144
max_standby_streaming_delay	(ms) Legt die maximale Verzögerung fest, bevor Abfragen storniert werden, wenn ein Hot-Standby-Server gestreamte WAL-Daten verarbeitet.	14000
max_worker_processes	Legt die maximale Anzahl gleichzeitiger Worker-Prozesse fest.	GREATEST($DBInstanceVCPU*2, 8
min_dynamic_shared_memory	(MB) Menge des dynamischen gemeinsam genutzten Speichers, der beim Start reserviert wurde.	–
min_parallel_index_scan_size	(8 kB) Legt die Mindestmenge an Indexdaten für einen parallelen Scan fest.	–
min_parallel_table_scan_size	(8 kB) Legt die Mindestmenge an Tabellend aten für einen parallelen Scan fest.	–

Parametername	Beschreibung	Standard
old_snapshot_threshold	(min) Zeit, bis ein Snapshot zu alt ist, um Seiten zu lesen, die nach dem Erstellen des Snapshots geändert wurden.	–
parallel_leader_participation	Steuert, ob Gather und Gather Merge auch Unterpläne ausführen.	–
parallel_setup_cost	Legt die Kostenschätzung des Planers für das Starten von Worker-Prozessen für parallele Abfragen fest.	–
parallel_tuple_cost	Legt die Kostenschätzung des Planers für die Übergabe von Tupeln (Zeilen) von Worker auf Haupt-Backend fest.	–
pgaudit.log	Gibt an, welche Klassen von Anweisungen durch die Sitzungsprüfungs-Protokollierung protokolliert werden.	–
pgaudit.log_catalog	Gibt an, dass die Sitzungsprotokollierung aktiviert werden soll, wenn sich alle Beziehungen in einer Anweisung in pg_catalog befinden.	–
pgaudit.log_level	Gibt die Protokollstufe an, die für Protokolleinträge verwendet wird.	–
pgaudit.log_parameter	Gibt an, dass die Audit-Protokollierung die Parameter enthalten sollte, die mit der Anweisung übergeben wurden.	–
pgaudit.log_relation	Gibt an, ob die Sitzungsprüfungs-Protokollierung für jede Beziehung (TABLE, VIEW usw.), auf die in einer SELECT- oder DML-Anweisung verwiesen wird, einen separaten Protokolleintrag erstellen soll.	–

Parametername	Beschreibung	Standard
pgaudit.log_statement_once	Gibt an, ob die Protokollierung den Anweisungstext und die Parameter mit dem ersten Protokolleintrag für eine Kombination aus Anweisung/Unteranweisung oder bei jedem Eintrag enthält.	–
pgaudit.role	Gibt die Hauptrolle an, die für die Objektüberwachungsprotokollierung verwendet werden soll.	–
pg_bigm.enable_recheck	Gibt an, ob Recheck durchgeführt werden soll, was ein interner Prozess der Volltextsuche ist.	on
pg_bigm.gin_key_limit	Gibt die maximale Anzahl von Bigrammen des Suchschlüsselworts an, das für die Volltextsuche verwendet werden soll.	0
pg_bigm.last_update	Meldet das letzte Aktualisierungsdatum des pg_bigm-Moduls.	2013.11.22
pg_bigm.similarity_limit	Es gibt den Mindestschwellenwert an, der von der Ähnlichkeitssuche verwendet wird.	0.3
pg_hint_plan.debug_print	Protokolliert Ergebnisse der Hinweisanalyse.	–
pg_hint_plan.enable_hint	Zwingt Planer zur Verwendung der Pläne, die im Hinweiskommentar vor der Abfrage angegeben sind.	–
pg_hint_plan.enable_hint_table	Zwingt Planer, Hinweise nicht über Tabellensuchen zu erhalten.	–
pg_hint_plan.message_level	Nachrichtenebene von Debug-Meldungen.	–

Parametername	Beschreibung	Standard
pg_hint_plan.parse_messages	Nachrichtenebene von Analysefehlern.	–
pglogical.batch_inserts	Batch-Inserts wenn möglich	–
pglogical.conflict_log_level	Legt die Protokollstufe gelöster Konflikte fest	–
pglogical.conflict_resolution	Legt die Methode zur Konfliktlösung für lösbare Konflikte fest.	–
pglogical.extra_connection_options	Verbindungsoptionen zum Hinzufügen zu allen Peer-Knotenverbindungen	–
pglogical.synchronous_commit	Spezifischer synchroner Commit-Wert für pglogical	–
pglogical.use_spi	Verwenden Sie SPI anstelle der Low-Level-API zum Anwenden von Änderungen	–
pg_similarity.block_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.block_threshold	Legt den Schwellenwert fest, der von der Block-Ähnlichkeitsfunktion verwendet wird.	–
pg_similarity.block_tokenizer	Legt den Tokenizer für die Block-Ähnlichkeitsfunktion fest.	–
pg_similarity.cosine_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.cosine_threshold	Legt den Schwellenwert fest, der von der Kosinus-Ähnlichkeitsfunktion verwendet wird.	–
pg_similarity.cosine_tokenizer	Legt den Tokenizer für die Kosinus-Ähnlichkeitsfunktion fest.	–

Parametername	Beschreibung	Standard
pg_similarity.dice_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.dice_threshold	Legt den Schwellenwert fest, der vom Dice-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.dice_tokenizer	Legt den Tokenizer für das Dice-Ähnlichkeitsmaß fest.	–
pg_similarity.euclidean_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.euclidean_threshold	Legt den Schwellenwert fest, der vom euklidischen Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.euclidean_tokenizer	Legt den Tokenizer für das euklidische Ähnlichkeitsmaß fest.	–
pg_similarity.hamming_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.hamming_threshold	Legt den Schwellenwert fest, der von der Block-Ähnlichkeitsmetrik verwendet wird.	–
pg_similarity.jaccard_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.jaccard_threshold	Legt den Schwellenwert fest, der vom Jaccard-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.jaccard_tokenizer	Legt den Tokenizer für das Jaccard-Ähnlichkeitsmaß fest.	–
pg_similarity.jaro_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.jaro_threshold	Legt den Schwellenwert fest, der vom Jaro-Ähnlichkeitsmaß verwendet wird	–

Parametername	Beschreibung	Standard
pg_similarity.jaro winkler_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.jaro winkler_threshold	Legt den Schwellenwert fest, der vom Jaro-Winkler-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.leve nshtein_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.leve nshtein_threshold	Legt den Schwellenwert fest, der vom Levenshtein-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.matc hing_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.matc hing_threshold	Legt den Schwellenwert fest, der vom Matching-Koeffizient-Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.matc hing_tokenizer	Legt den Tokenizer für das Matching-Koeffizie nt-Ähnlichkeitsmaß fest.	–
pg_similarity.mong eelkan_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.mong eelkan_threshold	Legt den Schwellenwert fest, der vom Monge-Elkan-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.mong eelkan_tokenizer	Legt den Tokenizer für das Monge-Elkan-Ähnlic hkeitsmaß fest.	–
pg_similarity.nw_g ap_penalty	Legt den Lückenabzug fest, der vom Ähnlichke itsmaß Needleman-Wunsch verwendet wird.	–
pg_similarity.nw_i s_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–

Parametername	Beschreibung	Standard
pg_similarity.nw_threshold	Legt den Schwellenwert fest, der vom Ähnlichkeitsmaß Needleman-Wunsch verwendet wird.	–
pg_similarity.overlap_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.overlap_threshold	Legt den Schwellenwert fest, der vom Überschneidungskoeffizient-Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.overlap_tokenizer	Legt den Tokenizer für das Überschneidungskoeffizient-Ähnlichkeitsmaß fest.	–
pg_similarity.qgram_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.qgram_threshold	Legt den Schwellenwert fest, der vom Q-Gram-Ähnlichkeitsmaß verwendet wird	–
pg_similarity.qgram_tokenizer	Legt den Tokenizer für die Q-Gram-Messung fest.	–
pg_similarity.swg_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.swg_threshold	Legt den Schwellenwert fest, der vom Smith-Waterman-Gotoh-Ähnlichkeitsmaß verwendet wird.	–
pg_similarity.sw_is_normalized	Legt fest, ob der Ergebniswert normalisiert ist oder nicht.	–
pg_similarity.sw_threshold	Legt den Schwellenwert fest, der vom Smith-Waterman-Ähnlichkeitsmaß verwendet wird	–

Parametername	Beschreibung	Standard
pg_stat_statements.max	Legt die maximale Anzahl von Anweisungen fest, die von pg_stat_statements verfolgt werden.	–
pg_stat_statements.save	Speichert pg_stat_statements-Statistiken über Serverausfälle hinweg.	–
pg_stat_statements.track	Wählt aus, welche Anweisungen von pg_stat_statements verfolgt werden.	–
pg_stat_statements.track_planning	Wählt aus, ob die Planungsdauer von pg_stat_statements verfolgt wird.	–
pg_stat_statements.track_utility	Wählt aus, ob Dienstprogrammbefehle von pg_stat_statements verfolgt werden.	–
postgis.gdal_enabled_drivers	Aktiviert oder deaktiviert GDAL-Treiber, die mit PostGIS in Postgres 9.3.5 und höher verwendet werden.	ENABLE_ALL
quote_all_identifiers	Fügt beim Generieren von SQL-Fragmenten allen Bezeichnern Anführungszeichen hinzu.	–
random_page_cost	Legt die Schätzung des Planers für die Kosten einer nicht sequenziell abgerufenen Datenträgerseite fest.	–
rds.enable_memory_management	Verbessert die Funktionen zur Arbeitsspeicherverwaltung in Aurora-PostgreSQL-Versionen 12.17, 13.13, 14.10, 15.5 und höher, die Stabilitätsprobleme und Neustarts der Datenbank aufgrund von zu wenig freiem Arbeitsspeicher verhindern Weitere Informationen finden Sie unter Verbesserte Arbeitsspeicherverwaltung in Aurora PostgreSQL.	True

Parametername	Beschreibung	Standard
rds.force_admin_logging_level	Siehe Protokollmeldungen für RDS-Admin-Benutzeraktionen in Kundendatenbanken.	–
rds.log_retention_period	Amazon RDS löscht PostgreSQL-Protokolle, die älter als N Minuten sind.	4320
rds.memory_allocation_guard	Verbessert die Funktionen zur Arbeitsspeicherverwaltung in Aurora-PostgreSQL-Versionen 11.21, 12.16, 13.12, 14.9, 15.4 und älter, die Stabilitätsprobleme und Neustarts der Datenbank aufgrund von zu wenig freiem Arbeitsspeicher verhindern Weitere Informationen finden Sie unter Verbesserte Arbeitsspeicherverwaltung in Aurora PostgreSQL.	False
rds.pg_stat_ramdisk_size	Größe der Ramdisk-Statistiken in MB. Ein Wert ungleich Null richtet die Ramdisk ein.	0
rds.rds_superuser_reserved_connections	Legt die Anzahl der Verbindungs-Slots fest, die für rds_superusers reserviert sind. Dieser Parameter ist nur für Version 15 und niedriger verfügbar. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation unter Reservierte Verbindungen.	2
rds.superuser_variables	Liste der reinen Superuser-Variablen, für die wir die Änderungsanweisungen von rds_superuser erhöhen.	session_replication_role
remove_temp_files_after_crash	Entfernt temporäre Dateien nach dem Backend-Absturz.	0
restart_after_crash	Initialisiert den Server nach dem Backend-Absturz neu.	–
row_security	Aktiviert Zeilensicherheit.	–

Parametername	Beschreibung	Standard
search_path	Legt die Schemasuchreihenfolge für Namen fest, die nicht schemaqualifiziert sind.	–
seq_page_cost	Legt die Schätzung des Planers für die Kosten einer sequenziell abgerufenen Datenträgerseite fest.	–
session_replication_role	Legt das Sitzungsverhalten für Auslöser und Neuschreibungsregeln fest.	–
shared_buffers	(8 kB) Legt die maximale Anzahl freigegebener Arbeitsspeicherpuffer fest, die vom Server verwendet werden.	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	Listet freigegebene Bibliotheken auf, die in den Server vorgeladen werden.	pg_stat_statements
ssl_ca_file	Speicherort der SSL-Server-Zertifizierungsstellendatei.	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	Speicherort der SSL-Serverzertifikatdatei.	/rdsdbdata/rds-metadata/server-cert.pem
ssl_crl_dir	Speicherort des Verzeichnisses der SSL-Zertifikatsperrliste.	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	Speicherort der privaten Schlüsseldatei des SSL-Servers	/rdsdbdata/rds-metadata/server-key.pem
standard_conforming_strings	Veranlasst Zeichenfolgen „...“, Backslashes als Zeichen zu behandeln.	–
statement_timeout	(ms) Legt die maximal zulässige Dauer von Anweisungen fest.	–
stats_temp_directory	Schreibt temporäre Statistikdateien in das angegebene Verzeichnis.	/rdsdbdata/db/pg_stat_tmp

Parametername	Beschreibung	Standard
superuser_reserved_connections	Legt die Anzahl der Verbindungs-Slots fest, die für Superusers reserviert sind.	3
synchronize_seqscans	Ermöglicht synchronisierte sequenzielle Prüfungen.	–
tcp_keepalives_count	Maximale Anzahl von TCP-Keepalive-Neuübertragungen.	–
tcp_keepalives_idle	(s) Zeit zwischen der Ausgabe von TCP-Keepalives.	–
tcp_keepalives_interval	(s) Zeit zwischen der Ausgabe von TCP-Keepalive-Neuübertragungen.	–
temp_buffers	(8 kB) Legt die maximale Anzahl der temporären Puffer fest, die von den einzelnen Sitzungen verwendet werden.	–
temp_file_limit	Schränkt den Gesamtspeicherplatz in Kilobyte ein, den ein bestimmter PostgreSQL-Prozess für temporäre Dateien verwenden kann, ausschließlich von Speicherplatz, der für explizite temporäre Tabellen verwendet wird	-1
temp_tablespaces	Legt die Tabellenräume fest, die für temporäre Tabellen und Sortierdateien verwendet werden sollen.	–
track_activities	Sammelt Informationen zu Befehlen, die ausgeführt werden.	–
track_activity_query_size	Legt die für pg_stat_activity.current_query reservierte Größe in Bytes fest.	4096
track_counts	Sammelt Statistiken zur Datenbankaktivität.	–

Parametername	Beschreibung	Standard
track_functions	Sammelt Statistiken auf Funktionsebene zur Datenbankaktivität.	pl
track_io_timing	Erfasst Zeitpunktstatistiken zur Datenbank-E/A-Aktivität.	1
transform_–_equals	Behandelt expr=– als expr IS –.	–
update_process_title	Aktualisiert den Titel des Vorgangs, um den aktiven SQL-Befehl anzuzeigen.	–
wal_receiver_statu s_interval	(s) Legt das maximale Intervall zwischen den Statusberichten des WAL-Receivers auf den primären Wert fest.	–
work_mem	(kB) Legt den maximalen Arbeitsspeicher fest, der für Abfrage-Workspaces verwendet werden darf.	–
xmlbinary	Legt die Kodierung von Binärwerten in XML fest.	–
xmloption	Legt fest, ob XML-Daten in impliziten Parsing- und Serialisierungsoperationen als Dokumente oder Inhaltsfragmente betrachtet werden sollen.	–

Amazon-Aurora-PostgreSQL-Warteereignisse

Im Folgenden sind gängige Warteereignisse für Aurora PostgreSQL aufgeführt. Weitere Informationen über Warteereignisse und die Optimierung Ihres Aurora-PostgreSQL-DB-Clusters finden Sie unter Optimierung mit Warteereignissen für Aurora PostgreSQL.

Aktivität: ArchiverMain

Der Archivierungsprozess wartet auf Aktivität.

Aktivität: AutoVacuumMain

Der Autovacuum-Launcher-Prozess wartet auf Aktivität.

Aktivität: BgWriterHibernate

Der Hintergrundschreibprozess befindet sich im Ruhezustand, während er auf Aktivität wartet.

Aktivität: BgWriterMain

Der Hintergrundschreibprozess wartet auf Aktivität.

Aktivität: CheckpointerMain

Der Checkpointer-Prozess wartet auf Aktivität.

Aktivität: LogicalApplyMain

Der Prozess zum Anwenden der logischen Replikation wartet auf Aktivität.

Aktivität: LogicalLauncherMain

Der Startprozess für die logische Replikation wartet auf Aktivität.

Aktivität: PgStatMain

Der Statistikkollektorprozess wartet auf Aktivität.

Aktivität: RecoveryWalAll

Ein Prozess wartet bei der Wiederherstellung auf das Write-Ahead-Log (WAL) von einem Stream.

Aktivität: RecoveryWalStream

Der Startup-Vorgang wartet auf das Eintreffen des Write-Ahead-Log (WAL) während der Streaming-Wiederherstellung.

Aktivität: SysLoggerMain

Der Syslogger-Prozess wartet auf Aktivität.

Aktivität: WalReceiverMain

Der WAL-Receiver-Prozess wartet auf Aktivität.

Aktivität: WalSenderMain

Der WAL-Senderprozess (Write-Ahead-Log) wartet auf Aktivität.

Aktivität: WalWriterMain

Der Write-Ahead-Log (WAL)-Writerprozess wartet auf Aktivität.

BufferPin:BufferPin

Ein Prozess wartet darauf, einen exklusiven Pin an einem Puffer zu erhalten.

Kunde: GSSOpen Server

Ein Prozess wartet darauf, Daten vom Client zu lesen, während er eine GSSAPI-Sitzung (Generic Security Service Application Program Interface) aufbaut.

Kunde: ClientRead

Ein Backend-Prozess wartet darauf, Daten von einem PostgreSQL-Client zu empfangen. Weitere Informationen finden Sie unter Client:ClientRead.

Kunde: ClientWrite

Ein Backend-Prozess wartet darauf, weitere Daten an einen PostgreSQL-Client zu senden. Weitere Informationen finden Sie unter Client:ClientWrite.

Kunde: Lib PQWal ReceiverConnect

Ein Prozess wartet im WAL-Receiver darauf, eine Verbindung zum Remote-Server herzustellen.

Kunde: Lib PQWal ReceiverReceive

Ein Prozess wartet im Write-Ahead-Log (WAL)-Receiver darauf, Daten vom Remote-Server zu empfangen.

Kunde: Server SSLOpen

Ein Prozess wartet auf Secure Sockets Layer (SSL), während er versucht, eine Verbindung herzustellen.

Kunde: WalReceiverWaitStart

Ein Prozess wartet darauf, dass der Startup-Prozess die Anfangsdaten für die Streaming-Replikation sendet.

Kunde: WalSenderWaitFor WAL

Ein Prozess wartet darauf, dass das Write-Ahead-Log (WAL) im WAL-Senderprozess geleert wird.

Kunde: WalSenderWriteData

Ein Prozess wartet auf eine Aktivität, wenn er Antworten vom Write-Ahead-Log (WAL)-Receiver im WAL-Senderprozess verarbeitet.

CPU

Ein Backend-Prozess ist in der CPU aktiv oder wartet auf diese. Weitere Informationen finden Sie unter CPU.

Extension:extension

Ein Backend-Prozess wartet auf eine Bedingung, die von einer Erweiterung oder einem Modul definiert wird.

IO: AuroraEnhancedLogical WALRead

Ein Backend-Prozess ruft Protokolldatensätze vom Change Data Capture (CDC)-Volume ab.

IO: AuroraOptimizedReadsCacheRead

Ein Prozess wartet auf einen Lesevorgang aus dem mehrstufigen Cache für optimierte Lesevorgänge, da die Seite nicht im gemeinsam genutzten Speicher verfügbar ist.

IO: AuroraOptimizedReads CacheSegmentKürzen

Ein Prozess wartet darauf, dass eine Segmentdatei des gestuften Caches für optimierte Lesevorgänge gekürzt wird.

IO: AuroraOptimizedReadsCacheWrite

Der Hintergrund-Writer-Prozess wartet darauf, in den mehrstufigen Cache für optimierte Lesevorgänge zu schreiben.

IO: AuroraStorageLogAllocate

Eine Sitzung weist Metadaten zu und bereitet einen Schreibvorgang für das Transaktionsprotokoll vor.

IO: BufFileRead

Wenn Vorgänge mehr Speicher benötigen, als durch die Parameter des Arbeitsspeichers definiert ist, erstellt die Engine temporäre Dateien auf der Festplatte. Dieses Warteereignis tritt auf, wenn Vorgänge aus den temporären Dateien gelesen werden. Weitere Informationen finden Sie unter IO: BufFileRead und IO: BufFileWrite.

IO: BufFileWrite

Wenn Vorgänge mehr Speicher benötigen, als durch die Parameter des Arbeitsspeichers definiert ist, erstellt die Engine temporäre Dateien auf der Festplatte. Dieses Warteereignis tritt auf, wenn Vorgänge in die temporären Dateien schreiben. Weitere Informationen finden Sie unter IO: BufFileRead und IO: BufFileWrite.

IO: ControlFileRead

Ein Prozess wartet auf das Lesen der `pg_control`-Datei.

IO: ControlFileSync

Ein Prozess wartet darauf, dass die `pg_control`-Datei den dauerhaften Speicher erreicht.

IO: ControlFileSyncUpdate

Ein Prozess wartet auf eine Aktualisierung der `pg_control`-Datei, um einen dauerhaften Speicher zu erreichen.

IO: ControlFileWrite

Ein Prozess wartet auf einen Schreibvorgang in die `pg_control`-Datei.

IO: ControlFileWriteUpdate

Ein Prozess wartet auf einen Schreibvorgang, um die `pg_control`-Datei zu aktualisieren.

IO: CopyFileRead

Ein Prozess wartet auf das Lesen während eines Dateikopiervorgangs.

IO: CopyFileWrite

Ein Prozess wartet während einer Dateikopieroperation auf einen Schreibvorgang.

IO: DataFileExtend

Ein Prozess wartet darauf, dass eine Relationsdatendatei erweitert wird.

IO: DataFileFlush

Ein Prozess wartet darauf, dass eine Relationsdatendatei einen dauerhaften Speicher erreicht.

IO: DataFileImmediateSync

Ein Prozess wartet auf eine sofortige Synchronisierung einer Relationsdatendatei mit einem dauerhaften Speicher.

IO: DataFilePrefetch

Ein Prozess wartet auf einen asynchronen Vorabruf aus einer Relationsdatendatei.

IO: DataFileSync

Ein Prozess wartet darauf, dass Änderungen an einer Relationsdatendatei einen dauerhaften Speicher erreichen.

IO: DataFileRead

Ein Backend-Prozess hat versucht, eine Seite in den freigegebenen Puffern zu finden, hat sie jedoch nicht gefunden und sie daher aus dem Speicher gelesen. Weitere Informationen finden Sie unter IO:DataFileRead.

IO: DataFileTruncate

Ein Prozess wartet darauf, dass eine Relationsdatendatei abgeschnitten wird.

IO: DataFileWrite

Ein Prozess wartet auf einen Schreibvorgang in eine Relationsdatendatei.

IO: DSMFill ZeroWrite

Ein Prozess wartet darauf, null Bytes in eine dynamische Shared Memory-Backing-Datei zu schreiben.

IO: LockFileAddToDataDirRead

Ein Prozess wartet auf das Lesen, während er der Datenverzeichnis-Sperrdatei eine Zeile hinzufügt.

IO: LockFileAddToDataDirSync

Ein Prozess wartet darauf, dass Daten einen dauerhaften Speicher erreichen, während er der Datenverzeichnis-Sperrdatei eine Zeile hinzufügt.

IO: LockFileAddToDataDirWrite

Ein Prozess wartet auf einen Schreibvorgang, während er der Datenverzeichnis-Sperrdatei eine Zeile hinzufügt.

IO: LockFileCreateRead

Ein Prozess wartet auf das Lesen, während er die Datenverzeichnis-Sperrdatei erstellt.

IO: LockFileCreateSync

Ein Prozess wartet darauf, dass Daten einen dauerhaften Speicher erreichen, während er die Datenverzeichnis-Sperrdatei erstellt.

IO: LockFileCreateWrite

Ein Prozess wartet auf einen Schreibvorgang, während er die Datenverzeichnis-Sperrdatei erstellt.

IO: LockFileReCheckDataDirRead

Ein Prozess wartet während der erneuten Überprüfung der Datenverzeichnis-Sperrdatei auf das Lesen.

IO: LogicalRewriteCheckpointSync

Ein Prozess wartet darauf, dass logische Rewrite-Mappings während eines Checkpoints dauerhaften Speicher erreichen.

IO: LogicalRewriteMappingSync

Ein Prozess wartet darauf, dass Mapping-Daten während eines logischen Neuschreibens einen dauerhaften Speicher erreichen.

IO: LogicalRewriteMappingWrite

Ein Prozess wartet während eines logischen Rewrite auf ein Schreiben von Mapping-Daten.

IO: LogicalRewriteSync

Ein Prozess wartet auf logische Rewrite-Zuordnungen, um dauerhaften Speicher zu erreichen.

IO: LogicalRewriteTruncate

Ein Prozess wartet auf das Abschneiden von Mapping-Daten während eines logischen Neuschreibens.

IO: LogicalRewriteWrite

Ein Prozess wartet auf einen Schreibvorgang von logischen Rewrite-Mappings.

IO: RelationMapRead

Ein Prozess wartet auf das Lesen der Relation Map-Datei.

IO: RelationMapSync

Ein Prozess wartet darauf, dass die Relationszuordnungsdatei einen dauerhaften Speicher erreicht.

IO: RelationMapWrite

Ein Prozess wartet auf einen Schreibvorgang in die Relation Map-Datei.

IO: ReorderBufferRead

Ein Prozess wartet während der Neuordnungs-Pufferverwaltung auf das Lesen.

IO: ReorderBufferWrite

Ein Prozess wartet während der Neuordnungs-Pufferverwaltung auf einen Schreibvorgang.

IO: ReorderLogicalMappingRead

Ein Prozess wartet während der Neuordnungspufferverwaltung auf das Lesen einer logischen Abbildung.

IO: ReplicationSlotRead

Ein Prozess wartet auf das Lesen aus einer Replikations-Slot-Steuerungsdatei.

IO: ReplicationSlotRestoreSync

Ein Prozess wartet darauf, dass eine Replikations-Slot-Steuerungsdatei einen dauerhaften Speicher erreicht, während sie im Arbeitsspeicher wiederhergestellt wird.

IO: ReplicationSlotSync

Ein Prozess wartet darauf, dass eine Replikations-Slot-Steuerdatei einen dauerhaften Speicher erreicht.

IO: ReplicationSlotWrite

Ein Prozess wartet auf einen Schreibvorgang in eine Replikations-Slot-Steuerdatei.

IO: SLRUFlush Synchronisieren

Ein Prozess wartet darauf, dass einfache, selten verwendete Daten (SLRU-Daten) während eines Prüfpunkts oder beim Herunterfahren der Datenbank in einem dauerhaften Speicher abgelegt werden.

IO: SLRURead

Ein Prozess wartet auf das Lesen einer einfachen, selten verwendeten Seite (SLRU-Seite).

IO: SLRUSync

Ein Prozess wartet darauf, dass einfache, selten verwendete Daten (SLRU-Daten) nach einem Seitenschreibvorgang in einem dauerhaften Speicher abgelegt werden.

IO: SLRUWrite

Ein Prozess wartet auf den Schreibvorgang einer einfachen, selten verwendeten Seite (SLRU-Seite).

IO: SnapbuildRead

Ein Prozess wartet auf das Lesen eines serialisierten historischen Katalog-Snapshots.

IO: SnapbuildSync

Ein Prozess wartet darauf, dass ein serialisierter historischer Katalog-Snapshot einen dauerhaften Speicher erreicht.

IO: SnapbuildWrite

Ein Prozess wartet auf das Schreiben eines serialisierten historischen Katalog-Snapshots.

IO: TimelineHistoryFileSync

Ein Prozess wartet darauf, dass eine Zeitachsenverlaufsdatei, die über die Streaming-Replikation empfangen wurde, im dauerhaften Speicher abgelegt wird.

IO: TimelineHistoryFileWrite

Ein Prozess wartet auf das Schreiben einer Zeitachsenverlaufsdatei, die über die Streaming-Replikation empfangen wurde.

IO: TimelineHistoryRead

Ein Prozess wartet auf das Lesen einer Zeitachsenverlaufsdatei.

IO: TimelineHistorySync

Ein Prozess wartet darauf, dass eine neu erstellte Zeitachsenverlaufsdatei einen dauerhaften Speicher erreicht.

IO: TimelineHistoryWrite

Ein Prozess wartet auf das Schreiben einer neu erstellten Zeitachsenverlaufsdatei.

IO: TwophaseFileRead

Ein Prozess wartet auf das Lesen einer Zweiphasen-Statusdatei.

IO: TwophaseFileSync

Ein Prozess wartet darauf, dass eine zweiphasige Zustandsdatei einen dauerhaften Speicher erreicht.

IO: TwophaseFileWrite

Ein Prozess wartet auf das Schreiben einer Zweiphasen-Zustandsdatei.

IO: WALBootstrap Synchronisieren

Ein Prozess wartet darauf, dass das Write-Ahead-Log (WAL) während des Bootstrapping im dauerhaften Speicher abgelegt wird.

IO: WALBootstrap Schreiben

Ein Prozess wartet während des Bootstrapping auf das Schreiben einer WAL-Seite.

IO: WALCopy Lesen

Ein Prozess wartet auf das Lesen, wenn ein neues Write-Ahead-Log (WAL)-Segment erstellt wird, indem ein vorhandenes kopiert wird.

IO: WALCopy Synchronisieren

Ein Prozess wartet auf ein neues WAL-Segment (Write-Ahead-Log), das durch Kopieren eines vorhandenen Segments erstellt wurde, um dauerhaften Speicher zu erreichen.

IO: WALCopy Schreiben

Ein Prozess wartet auf einen Schreibvorgang, wenn er ein neues WAL-Segment (Write-Ahead-Log) erstellt, indem er ein vorhandenes kopiert.

IO: WALInit Synchronisieren

Ein Prozess wartet darauf, dass eine neu initialisierte Write-Ahead-Log-Datei (WAL) im dauerhaften Speicher abgelegt wird.

IO: WALInit Schreiben

Ein Prozess wartet auf einen Schreibvorgang, während er eine neue Write-Ahead-Protokolldatei (WAL) initialisiert.

IO: WALRead

Ein Prozess wartet auf das Lesen aus einer Write-Ahead-Protokolldatei (WAL).

IO: WALSender TimelineHistoryRead

Ein Prozess wartet während eines WAL-Sender-Zeitachsenbefehls auf das Lesen aus einer Zeitachsenverlaufsdatei.

IO: WALSync

Ein Prozess wartet darauf, dass eine Write-Ahead-Protokolldatei (WAL) einen dauerhaften Speicher erreicht.

IO: WALSync MethodAssign

Ein Prozess wartet darauf, dass Daten einen dauerhaften Speicher erreichen, während er eine neue WAL-Synchronisierungsmethode (Write-Ahead-Log) zuweist.

IO: WALWrite

Ein Prozess wartet auf einen Schreibvorgang in eine Write-Ahead-Protokolldatei (WAL).

IO: XactSync

Ein Backend-Prozess wartet darauf, dass das Aurora-Speichersubsystem den Commit einer regulären Transaktion oder den Commit oder das Rollback einer vorbereiteten Transaktion bestätigt. Weitere Informationen finden Sie unter IO:XactSync.

IPC: AuroraLogicalSchemaUpdate

Zwei Backend-Prozesse versuchen, denselben Eintrag in den Schema-Cache einzufügen. Ein Prozess wird fortgesetzt, während der andere auf seinen Abschluss wartet.

IPC: AuroraOptimizedReadsCacheWriteStop

Ein Prozess wartet darauf, dass der Hintergrundschreiber das Schreiben in den gestuften Cache für optimierte Lesevorgänge beendet.

IPC: BackupWaitWalArchive

Ein Prozess wartet auf Write-Ahead-Log-Dateien (WAL), die für eine erfolgreiche Archivierung einer Sicherung erforderlich sind.

IPC: BgWorkerShutdown

Ein Prozess wartet darauf, dass ein Arbeitsprozess im Hintergrund heruntergefahren wird.

IPC: BgWorkerStartup

Ein Prozess wartet darauf, dass ein Hintergrund-Worker gestartet wird.

IPC: BtreePage

Ein Prozess wartet darauf, dass die Seitennummer, die zum Fortsetzen einer parallelen B-Baum-Abtastung benötigt wird, verfügbar wird.

IPC: CheckpointDone

Ein Prozess wartet darauf, dass ein Checkpoint abgeschlossen wird.

IPC: CheckpointStart

Ein Prozess wartet auf den Start eines Checkpoints.

IPC: ClogGroupUpdate

Ein Prozess wartet darauf, dass der Gruppenleiter den Transaktionsstatus am Ende einer Transaktion aktualisiert.

IPC: DamRecordTxAck

Ein Backend-Prozess hat ein Ereignis für Datenbank-Aktivitätsströme generiert und wartet darauf, dass das Ereignis dauerhaft wird. Weitere Informationen finden Sie unter IPC:DamRecordTxAck.

IPC: ExecuteGather

Ein Prozess wartet auf Aktivitäten von einem untergeordneten Prozess, während er einen Gather-Plan-Knoten ausführt.

IPC: Hash/Batch/Allocating

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer eine Hash-Tabelle zuweist.

IPC: Hash/Batch/Electing

Ein Prozess wählt einen parallelen Hash-Teilnehmer aus, um eine Hash-Tabelle zuzuweisen.

IPC: Hash/Batch/Loading

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer das Laden einer Hash-Tabelle abgeschlossen haben.

IPC: Hash/Build/Allocating

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer die anfängliche Hash-Tabelle zuweist.

IPC: Hash/Build/Electing

Ein Prozess wählt einen parallelen Hash-Teilnehmer aus, um die anfängliche Hash-Tabelle zuzuweisen.

IPC: Hash/Build/HashingInner

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer das Hashing der inneren Beziehung abgeschlossen haben.

IPC: Hash/Build/HashingOuter

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer die Partitionierung der äußeren Beziehung abschließen.

IPC: Hash/GrowBatches/Allocating

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer weitere Batches zuweist.

IPC: Hash/GrowBatches/Deciding

Ein Prozess wählt einen parallelen Hash-Teilnehmer aus, um über zukünftiges Batch-Wachstum zu entscheiden.

IPC: Hash/GrowBatches/Electing

Ein Prozess wählt einen parallelen Hash-Teilnehmer aus, um mehr Batches zuzuweisen.

IPC: Hash/GrowBatches/Finishing

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer über zukünftiges Batch-Wachstum entscheidet.

IPC: Hash/GrowBatches/Repartitioning

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer die Neupartitionierung abgeschlossen haben.

IPC: Hash/GrowBuckets/Allocating

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer die Zuweisung weiterer Buckets abgeschlossen hat.

IPC: Hash/GrowBuckets/Electing

Ein Prozess wählt einen parallelen Hash-Teilnehmer aus, um mehr Buckets zuzuweisen.

IPC: Hash/GrowBuckets/Reinserting

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer das Einfügen von Tupeln in neue Buckets abgeschlossen haben.

IPC: HashBatchAllocate

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer eine Hash-Tabelle zuweist.

IPC: HashBatchElect

Ein Prozess wartet darauf, einen parallelen Hash-Teilnehmer auszuwählen, um eine Hash-Tabelle zuzuweisen.

IPC: HashBatchLoad

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer das Laden einer Hash-Tabelle abgeschlossen haben.

IPC: HashBuildAllocate

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer die anfängliche Hash-Tabelle zuweist.

IPC: HashBuildElect

Ein Prozess wartet darauf, einen parallelen Hash-Teilnehmer auszuwählen, um die anfängliche Hash-Tabelle zuzuweisen.

IPC: HashBuildHashInner

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer das Hashing der inneren Beziehung abgeschlossen haben.

IPC: 'HashBuildHashOuter

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer die Partitionierung der äußeren Beziehung abschließen.

IPC: HashGrowBatchesAllocate

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer weitere Batches zuweist.

IPC: 'HashGrowBatchesDecide

Ein Prozess wartet darauf, einen parallelen Hash-Teilnehmer auszuwählen, um über das zukünftige Batch-Wachstum zu entscheiden.

IPC: HashGrowBatchesElect

Ein Prozess wartet darauf, einen parallelen Hash-Teilnehmer auszuwählen, um weitere Batches zuzuweisen.

IPC: HashGrowBatchesFinish

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer über zukünftiges Batch-Wachstum entscheidet.

IPC: HashGrowBatchesRepartition

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer die Neupartitionierung beenden.

IPC: HashGrowBucketsAllocate

Ein Prozess wartet darauf, dass ein gewählter paralleler Hash-Teilnehmer die Zuweisung weiterer Buckets abgeschlossen hat.

IPC: HashGrowBucketsElect

Ein Prozess wartet darauf, einen parallelen Hash-Teilnehmer auszuwählen, um weitere Buckets zuzuweisen.

IPC: HashGrowBucketsReinsert

Ein Prozess wartet darauf, dass andere parallele Hash-Teilnehmer das Einfügen von Tupeln in neue Buckets abgeschlossen haben.

IPC: LogicalSyncData

Ein Prozess wartet darauf, dass ein Remote-Server für die logische Replikation Daten für die anfängliche Tabellensynchronisierung sendet.

IPC: LogicalSyncStateChange

Ein Prozess wartet darauf, dass ein logischer Replikations-Remote-Server seinen Status ändert.

IPC: MessageQueueInternal

Ein Prozess wartet darauf, dass ein anderer Prozess an eine gemeinsam genutzte Nachrichtenwarteschlange angehängt wird.

IPC: MessageQueuePutMessage

Ein Prozess wartet darauf, eine Protokollnachricht in eine gemeinsam genutzte Nachrichtenwarteschlange zu schreiben.

IPC: MessageQueueReceive

Ein Prozess wartet darauf, Bytes aus einer gemeinsam genutzten Nachrichtenwarteschlange zu empfangen.

IPC: MessageQueueSend

Ein Prozess wartet darauf, Bytes an eine gemeinsam genutzte Nachrichtenwarteschlange zu senden.

IPC: ParallelBitmapScan

Ein Prozess wartet darauf, dass ein paralleler Bitmap-Scan initialisiert wird.

IPC: ParallelCreateIndexScan

Ein Prozess wartet darauf, dass parallele CREATE INDEX-Arbeitsprozesse einen Heap-Scan abschließen.

IPC: ParallelFinish

Ein Prozess wartet darauf, dass parallele Arbeitsprozesse die Berechnung abgeschlossen haben.

IPC: ProcArrayGroupUpdate

Ein Prozess wartet darauf, dass der Gruppenleiter die Transaktions-ID am Transaktionsende löscht.

IPC: ProcSignalBarrier

Ein Prozess wartet darauf, dass ein Barriere-Ereignis von allen Backends verarbeitet wird.

IPC:Promote

Ein Prozess wartet auf die Standby-Heraufstufung.

IPC: RecoveryConflictSnapshot

Ein Prozess wartet auf die Lösung des Wiederherstellungskonflikts für eine Vakuumbereinigung.

IPC: RecoveryConflictTablespace

Ein Prozess wartet auf die Lösung eines Wiederherstellungskonflikts, um einen Tablespace zu löschen.

IPC: RecoveryPause

Ein Prozess wartet darauf, dass die Wiederherstellung fortgesetzt wird.

IPC: ReplicationOriginDrop

Ein Prozess wartet darauf, dass ein Replikationsursprung inaktiv wird, damit er gelöscht werden kann.

IPC: ReplicationSlotDrop

Ein Prozess wartet darauf, dass ein Replikations-Slot inaktiv wird, damit er gelöscht werden kann.

IPC: SafeSnapshot

Ein Prozess wartet darauf, einen gültigen Snapshot für eine READ ONLY DEFERRABLE-Transaktion abzurufen.

IPC: SyncRep

Ein Prozess wartet während der synchronen Replikation auf die Bestätigung eines Remote-Servers.

IPC: XactGroupUpdate

Ein Prozess wartet darauf, dass der Gruppenleiter den Transaktionsstatus am Transaktionsende aktualisiert.

Lock:advisory

Ein Backend-Prozess hat eine Advisory-Sperre angefordert und wartet darauf. Weitere Informationen finden Sie unter Lock:advisory.

Lock:extend

Ein Backend-Prozess wartet darauf, dass eine Sperre freigegeben wird, damit er eine Relation erweitern kann. Diese Sperre wird benötigt, da jeweils nur ein Backend-Prozess eine Relation erweitern kann. Weitere Informationen finden Sie unter Lock:extend.

Lock:frozenid

Ein Prozess wartet darauf, `pg_database.datfrozenxid` und `pg_database.datminmxid` zu aktualisieren.

Lock:object

Ein Prozess wartet darauf, eine Sperre für ein Objekt einer nichtrelationalen Datenbank zu erhalten.

Lock:page

Ein Prozess wartet darauf, eine Sperre für eine Seite einer Relation zu erhalten.

Lock:Relation

Ein Backend-Prozess wartet darauf, eine Sperre für eine Relation zu erlangen, die durch eine andere Transaktion gesperrt ist. Weitere Informationen finden Sie unter Lock:Relation.

Lock:spectoken

Ein Prozess wartet darauf, eine spekulative Einfügesperre zu erhalten.

Lock:speculative token

Ein Prozess wartet darauf, eine spekulative Einfügesperre zu erlangen.

Lock:transactionid

Eine Transaktion wartet auf eine Sperre auf Zeilenebene. Weitere Informationen finden Sie unter Lock:transactionid.

Lock:tuple

Ein Backend-Prozess wartet darauf, eine Sperre für ein Tupel zu erlangen, während ein anderer Backend-Prozess eine widersprüchliche Sperre für dasselbe Tupel hält. Weitere Informationen finden Sie unter Lock:tuple.

Lock:userlock

Ein Prozess wartet darauf, eine Benutzersperre zu erhalten.

Lock:virtualxid

Ein Prozess wartet darauf, eine virtuelle Transaktions-ID-Sperre zu erhalten.

LWLock:AddinShmemInit

Ein Prozess wartet darauf, die Speicherplatzzuweisung einer Erweiterung im gemeinsam genutzten Speicher zu verwalten.

LWLock:AddinShmemInitLock

Ein Prozess wartet darauf, die Speicherplatzzuweisung im gemeinsam genutzten Speicher zu verwalten.

LWLockIPC: ----sep----:asynchron

Ein Prozess wartet I/O auf einen asynchronen Puffer (Notify).

LWLock:AsyncCtlLock

Ein Prozess wartet darauf, einen freigegebenen Benachrichtigungsstatus zu lesen oder zu aktualisieren.

LWLock:AsyncQueueLock

Ein Prozess wartet darauf, Benachrichtigungen zu lesen oder zu aktualisieren.

LWLock:AuroraOptimizedReadsCacheMapping

Ein Prozess wartet darauf, einen Datenblock mit einer Seite im mehrstufigen Cache für optimierte Lesevorgänge zu verknüpfen.

LWLock:AutoFile

Ein Prozess wartet darauf, die Datei `postgresql.auto.conf` zu aktualisieren.

LWLock:AutoFileLock

Ein Prozess wartet darauf, die Datei `postgresql.auto.conf` zu aktualisieren.

LWLockEin Prozess wartet auf einen asynchronen Puffer (Notify). ----sep----:AutoVacuum

Ein Prozess wartet darauf, den aktuellen Status der Autovacuum-Arbeitsprozesse zu lesen oder zu aktualisieren.

LWLock:AutovacuumLock

Ein Autovacuum-Worker oder ein Startprogramm wartet darauf, den aktuellen Status der Autovacuum-Worker zu aktualisieren oder zu lesen.

LWLock:AutovacuumSchedule

Ein Prozess wartet, um sicherzustellen, dass eine für den Autovaccuum-Prozess ausgewählte Tabelle noch bereinigt werden muss.

LWLock:AutovacuumScheduleLock

Ein Prozess wartet, um sicherzustellen, dass eine für den Autovakcuum-Prozess ausgewählte Tabelle noch bereinigt werden muss.

LWLock:BackendRandomLock

Ein Prozess wartet darauf, eine Zufallszahl zu generieren.

LWLock:BackgroundWorker

Ein Prozess wartet darauf, den Hintergrund-Worker-Status zu lesen oder zu aktualisieren.

LWLock:BackgroundWorkerLock

Ein Prozess wartet darauf, den Hintergrund-Worker-Status zu lesen oder zu aktualisieren.

LWLock:BtreeVacuum

Ein Prozess wartet darauf, vacuumbezogene Informationen für einen B-Baum-Index zu lesen oder zu aktualisieren.

LWLock:BtreeVacuumLock

Ein Prozess wartet darauf, vacuumbezogene Informationen für einen B-Baum-Index zu lesen oder zu aktualisieren.

LWLock:Autovacuum----sep----:buffer_content

Ein Backend-Prozess wartet darauf, eine einfache Sperre für den Inhalt eines gemeinsam genutzten Speicherpuffers zu erlangen. Weitere Informationen finden Sie unter LWLock:buffer_content (BufferContent).

LWLock:buffer_content ----sep----:buffer_mapping

Ein Backend-Prozess wartet darauf, einen Datenblock mit einem Puffer im gemeinsam genutzten Pufferpool zu verknüpfen. Weitere Informationen finden Sie unter LWLock:buffer_mapping.

LWLock:buffer_mapping ----sep----:BufferIO

Ein Backend-Prozess möchte eine Seite in den gemeinsam genutzten Speicher einlesen. Der Prozess wartet darauf, dass andere Prozesse ihre Arbeit I/O für die Seite beenden. Weitere Informationen finden Sie unter LWLock:BufferIO (IPC:BufferIO).

LWLockDer Prozess wartet darauf, dass andere Prozesse ihre Arbeit für die Seite beenden. ----sep----:Checkpoint

Ein Prozess wartet darauf, einen Checkpoint zu starten.

LWLock:CheckpointLock

Ein Prozess wartet darauf, einen Checkpoint auszuführen.

LWLock:CheckpointerComm

Ein Prozess wartet darauf, `fsync` Anfragen zu verwalten.

LWLock:CheckpointerCommLock

Ein Prozess wartet darauf, `fsync` Anfragen zu verwalten.

LWLock:Checkpoint ----sep----:Holzschuh

Ein Prozess wartet I/O auf einen Clog-Puffer (Transaktionsstatus).

LWLock:CLogControlLock

Ein Prozess wartet darauf, den Transaktionsstatus zu lesen oder zu aktualisieren.

LWLock:CLogTruncationLock

Ein Prozess wartet darauf, `txid_status` auszuführen oder die älteste verfügbare Transaktions-ID zu aktualisieren.

LWLockEin Prozess wartet auf einen Clog-Puffer (Transaktionsstatus). ----sep----:commit_timestamp

Ein Prozess wartet auf einen I/O Commit-Zeitstempelpuffer.

LWLock:CommitTs

Ein Prozess wartet darauf, den letzten für einen Transaktions-Commit-Zeitstempel festgelegten Wert zu lesen oder zu aktualisieren.

LWLock:CommitTsBuffer

Ein Prozess wartet I/O auf einen einfachen Puffer, der zuletzt zuletzt verwendet wurde (SLRU), auf einen Commit-Zeitstempel.

LWLock:CommitTsControlLock

Ein Prozess wartet darauf, Transaktions-Commit-Zeitstempel zu lesen oder zu aktualisieren.

LWLock:CommitTsLock

Ein Prozess wartet darauf, den letzten für den Transaktionszeitstempel festgelegten Wert zu lesen oder zu aktualisieren.

LWLock: SLRU CommitTs

Ein Prozess wartet auf den Zugriff auf den einfachen, selten verwendeten Cache (SLRU-Cache) für einen Commit-Zeitstempel.

LWLock:ControlFile

Ein Prozess wartet darauf, die Datei `pg_control` zu lesen oder zu aktualisieren oder eine neue Write-Ahead-Log-Datei (WAL) zu erstellen.

LWLock:ControlFileLock

Ein Prozess wartet darauf, die Steuerdatei zu lesen oder zu aktualisieren oder eine neue Write-Ahead-Protokolldatei (WAL) zu erstellen.

LWLock:DynamicSharedMemoryControl

Ein Prozess wartet darauf, Informationen zur Zuordnung des dynamischen gemeinsamen Speichers zu lesen oder zu aktualisieren.

LWLock:DynamicSharedMemoryControlLock

Ein Prozess wartet darauf, den Status des dynamischen gemeinsam genutzten Speichers zu lesen oder zu aktualisieren.

LWLock: SLRU ----sep----:lock_manager

Ein Backend-Prozess wartet darauf, Sperren für Backend-Prozesse hinzuzufügen oder zu untersuchen. Oder er wartet darauf, einer Sperrgruppe beizutreten oder diese zu verlassen, die von der parallelen Abfrage verwendet wird. Weitere Informationen finden Sie unter LWLock:lock_manager.

LWLock:LockFastPath

Ein Prozess wartet darauf, die Informationen zur Fast-Path-Sperre eines Prozesses zu lesen oder zu aktualisieren.

LWLock:LogicalRepWorker

Ein Prozess wartet darauf, den Status der logischen Replikations-Arbeitsprozesse zu lesen oder zu aktualisieren.

LWLock:LogicalRepWorkerLock

Ein Prozess wartet darauf, dass eine Aktion auf einem logischen Replikations-Worker beendet wird.

LWLock:LogicalSchemaCache

Ein Prozess hat den Schema-Cache geändert.

LWLock:lock_manager ----sep----:multixact_member

Ein Prozess wartet auf einen Multixact_Member-Puffer I/O .

LWLockEin Prozess wartet auf einen Multixact_Member-Puffer. ----sep----:multixact_offset

Ein Prozess wartet auf einen I/O Multixact-Offset-Puffer.

LWLock:MultiXactGen

Ein Prozess wartet darauf, den freigegebenen multixact-Status zu lesen oder zu aktualisieren.

LWLock:MultiXactGenLock

Ein Prozess wartet darauf, einen gemeinsam genutzten multixact-Zustand zu lesen oder zu aktualisieren.

LWLock:MultiXactMemberBuffer

Ein Prozess wartet I/O auf einen einfachen SLRU-Puffer (Least-Recently Used) für ein Multixact-Mitglied. Weitere Informationen finden Sie unter LWLock:MultiXact.

LWLock:MultiXactMemberControlLock

Ein Prozess wartet darauf, Multixact-Mitgliederzuordnungen zu lesen oder zu aktualisieren.

LWLock: SLRU MultiXactMember

– Ein Prozess wartet darauf, auf den einfachen, selten verwendeten Cache (SLRU-Cache) für ein Multixact-Mitglied zuzugreifen. Weitere Informationen finden Sie unter LWLock:MultiXact.

LWLock:MultiXactOffsetBuffer

Ein Prozess wartet I/O auf einen einfachen Puffer, der zuletzt zuletzt benutzt wurde (SLRU), auf einen Multixact-Offset. Weitere Informationen finden Sie unter LWLock:MultiXact.

LWLock:MultiXactOffsetControlLock

Ein Prozess wartet darauf, multixact Offset-Mappings zu lesen oder zu aktualisieren.

LWLock: SLRU MultiXactOffset

– Ein Prozess wartet darauf, auf den einfachen, selten verwendeten Cache (SLRU-Cache) für ein Multixact-Offset zuzugreifen. Weitere Informationen finden Sie unter LWLock:MultiXact.

LWLock:MultiXactTruncation

Ein Prozess wartet darauf, multixact-Informationen zu lesen oder abzuschneiden.

LWLock:MultiXactTruncationLock

Ein Prozess wartet darauf, multixact-Informationen zu lesen oder abzuschneiden.

LWLock:NotifyBuffer

Ein Prozess wartet I/O im SLRU-Puffer (Simple Least-Recently Used) auf eine NOTIFY-Nachricht.

LWLock:NotifyQueue

Ein Prozess wartet darauf, NOTIFY-Nachrichten zu lesen oder zu aktualisieren.

LWLock:NotifyQueueTail

Ein Prozess wartet darauf, einen Grenzwert für den NOTIFY-Nachrichtenspeicher zu aktualisieren.

LWLock:NotifyQueueTailLock

Ein Prozess wartet darauf, das Limit für den Speicher von Benachrichtigungsnachrichten zu aktualisieren.

LWLockEin Prozess wartet auf dem einfachen, zuletzt verwendeten (SLRU) -Puffer auf eine NOTIFY-Nachricht. ----sep----:notifySLRU

Ein Prozess wartet darauf, auf den einfachen, selten verwendeten Cache (SLRU-Cache) für eine NOTIFY-Nachricht zuzugreifen.

LWLock:OidGen

Ein Prozess wartet darauf, eine neue Objekt-ID (OID) zuzuweisen.

LWLock:OidGenLock

Ein Prozess wartet darauf, eine Objekt-ID (OID) zuzuweisen oder zuzuweisen.

LWLock:notifyslru ----sep----:oldserxid

Ein Prozess wartet auf einen I/O oldserxid-Puffer.

LWLock:OldSerXidLock

Ein Prozess wartet darauf, in Konflikt stehende serialisierbare Transaktionen zu lesen oder aufzuzeichnen.

LWLock:OldSnapshotTimeMap

Ein Prozess wartet darauf, alte Snapshot-Steuerungsinformationen zu lesen oder zu aktualisieren.

LWLock:OldSnapshotTimeMapLock

Ein Prozess wartet darauf, alte Snapshot-Steuerungsinformationen zu lesen oder zu aktualisieren.

LWLockEin Prozess wartet auf einen oldserxid-Puffer. ----sep----:parallel_append

Ein Prozess wartet darauf, während der parallelen Ausführung des Anhänge-Plans den nächsten Unterplan auszuwählen.

LWLock:parallel_append ----sep----:parallel_hash_join

Ein Prozess wartet darauf, während einer parallelen Hash-Plan-Ausführung einen Speicherblock zuzuweisen oder auszutauschen oder Zähler zu aktualisieren.

LWLock:parallel_hash_join ----sep----:parallel_query_dsa

Ein Prozess wartet auf eine Sperre für die dynamische Shared Memory-Zuweisung für eine parallele Abfrage.

LWLock:ParallelAppend

Ein Prozess wartet darauf, während der parallelen Ausführung des Anhänge-Plans den nächsten Unterplan auszuwählen.

LWLock:ParallelHashJoin

Ein Prozess wartet darauf, während der Planausführung Worker für einen parallelen Hash-Join zu synchronisieren.

Sperre: DSA ParallelQuery

Ein Prozess wartet auf eine dynamische Shared Memory-Zuweisung für eine parallele Abfrage.

Lwlock: DSA PerSession

Ein Prozess wartet auf eine dynamische Shared Memory-Zuweisung für eine parallele Abfrage.

Lwlock: PerSessionRecordType

Ein Prozess wartet darauf, auf die Informationen einer parallelen Abfrage zu zusammengesetzten Typen zuzugreifen.

Lwlock: PerSessionRecordTypmod

Ein Prozess wartet darauf, auf die Informationen einer parallelen Abfrage über Typmodifizierer zuzugreifen, die anonyme Datensatztypen identifizieren.

Lwlock: PerXactPredicateList

Ein Prozess wartet darauf, während einer parallelen Abfrage auf die Liste der Prädikatssperren zuzugreifen, die von der aktuellen serialisierbaren Transaktion gehalten werden.

Lwlock:predicate_lock_manager

Ein Prozess wartet darauf, Informationen zur Prädikatsperre hinzuzufügen oder zu untersuchen.

Lwlock: PredicateLockManager

Ein Prozess wartet darauf, auf Prädikats-Sperrinformationen zuzugreifen, die von serialisierbaren Transaktionen verwendet werden.

Lwlock:proc

Ein Prozess wartet darauf, die Fast-Path-Sperrinformationen zu lesen oder zu aktualisieren.

LWLock:ProcArray

Ein Prozess wartet darauf, auf die freigegebenen prozessbezogenen Datenstrukturen zuzugreifen (normalerweise um einen Snapshot zu erhalten oder die Transaktions-ID einer Sitzung zu melden).

LWLock:ProcArrayLock

Ein Prozess wartet darauf, einen Snapshot zu erhalten oder eine Transaktions-ID am Ende einer Transaktion zu löschen.

LWLock:RelationMapping

Ein Prozess wartet darauf, eine `pg_filenode.map`-Datei zu lesen oder zu aktualisieren (wird verwendet, um die Dateiknotenzuordnungen bestimmter Systemkataloge zu verfolgen).

LWLock:RelationMappingLock

Ein Prozess wartet darauf, die Relation-Map-Datei zu aktualisieren, die zum Speichern der catalog-to-file-node Zuordnung verwendet wird.

LWLock:RelCacheInit

Ein Prozess wartet darauf, eine `pg_internal.init`-Datei zu lesen oder zu aktualisieren (eine Initialisierungsdatei für den Beziehungscache).

LWLock:RelCacheInitLock

Ein Prozess wartet darauf, eine Initialisierungsdatei für den Beziehungscache zu lesen oder zu schreiben.

LWLockEin Prozess wartet darauf, die Relationszuordnungsdatei zu aktualisieren, die zum Speichern der Zuordnung verwendet wird. ----sep----:replication_origin

Ein Prozess wartet darauf, den Replikationsfortschritt zu lesen oder zu aktualisieren.

LWLock:replication_origin ----sep----:replication_slot_io

Auf einen Replikationsslot wartet ein Prozess. I/O

LWLock:ReplicationOrigin

Ein Prozess wartet darauf, einen Replikationsursprung zu erstellen, zu löschen oder zu verwenden.

LWLock:ReplicationOriginLock

Ein Prozess wartet darauf, einen Replikationsursprung einzurichten, zu löschen oder zu verwenden.

LWLock:ReplicationOriginState

Ein Prozess wartet darauf, den Fortschritt eines Replikationsursprungs zu lesen oder zu aktualisieren.

LWLock:ReplicationSlotAllocation

Ein Prozess wartet darauf, einen Replikations-Slot zuzuweisen oder freizugeben.

LWLock:ReplicationSlotAllocationLock

Ein Prozess wartet darauf, einen Replikations-Slot zuzuweisen oder freizugeben.

LWLock:ReplicationSlotControl

Ein Prozess wartet darauf, einen Replikations-Slot-Status zu lesen oder zu aktualisieren.

LWLock:ReplicationSlotControlLock

Ein Prozess wartet darauf, den Status des Replikations-Slots zu lesen oder zu aktualisieren.

LWLock: ReplicationSlot IO

I/O Auf einen Replikationssteckplatz wartet ein Prozess.

LWLock:SerialBuffer

Ein Prozess wartet I/O auf einen SLRU-Puffer (Simple Least-Recently Used) für einen serialisierbaren Transaktionskonflikt.

LWLock:SerializableFinishedList

Ein Prozess wartet darauf, auf die Liste der abgeschlossenen serialisierbaren Transaktionen zuzugreifen.

LWLock:SerializableFinishedListLock

Ein Prozess wartet darauf, auf die Liste der abgeschlossenen serialisierbaren Transaktionen zuzugreifen.

LWLock:SerializablePredicateList

Ein Prozess wartet darauf, auf die Liste der Prädikatssperren zuzugreifen, die von serialisierbaren Transaktionen gehalten werden.

LWLock:SerializablePredicateLockListLock

Ein Prozess wartet darauf, eine Operation für eine Liste von Sperren auszuführen, die von serialisierbaren Transaktionen gehalten werden.

LWLock:SerializableXactHash

Ein Prozess wartet darauf, Informationen zu serialisierbaren Transaktionen zu lesen oder zu aktualisieren.

LWLock:SerializableXactHashLock

Ein Prozess wartet darauf, Informationen zu serialisierbaren Transaktionen abzurufen oder zu speichern.

LWLockEin Prozess wartet auf einen einfachen, zuletzt verwendeten (SLRU) -Puffer für einen serialisierbaren Transaktionskonflikt. ----sep----:serialSLRU

Ein Prozess wartet auf den Zugriff auf den einfachen, selten verwendeten Cache (SLRU-Cache) für einen serialisierbaren Transaktionskonflikt.

LWLock:SharedTidBitmap

Ein Prozess wartet darauf, während einer parallelen Bitmap-Indexabtastung auf eine Bitmap mit gemeinsam genutzten Tupelidentifizierern (TID) zuzugreifen.

LWLock:SharedTupleStore

Ein Prozess wartet während einer parallelen Abfrage darauf, auf einen gemeinsam genutzten Tupelspeicher zuzugreifen.

LWLock:ShmemIndex

Ein Prozess wartet darauf, Speicherplatz im gemeinsam genutzten Speicher zu finden oder zuzuweisen.

LWLock:ShmemIndexLock

Ein Prozess wartet darauf, Speicherplatz im gemeinsam genutzten Speicher zu finden oder zuzuweisen.

LWLock: Lesen SInval

Ein Prozess wartet darauf, Nachrichten aus der freigegebenen Katalog-Invalidierungswarteschlange abzurufen.

LWLock:SInvalReadLock

Ein Prozess wartet darauf, Nachrichten aus einer gemeinsam genutzten Invalidierungswarteschlange abzurufen oder zu entfernen.

LWLock: SInval Schreiben

Ein Prozess wartet darauf, der gemeinsam genutzten Katalog-Invalidierungswarteschlange eine Nachricht hinzuzufügen.

LWLock:SInvalWriteLock

Ein Prozess wartet darauf, eine Nachricht in eine gemeinsam genutzte Invalidierungswarteschlange hinzuzufügen.

LWLock:SyncRep

Ein Prozess wartet darauf, Informationen zum Status der synchronen Replikation zu lesen oder zu aktualisieren.

LWLock:SyncRepLock

Ein Prozess wartet darauf, Informationen zu synchronen Replikaten zu lesen oder zu aktualisieren.

LWLock:SyncScan

Ein Prozess wartet darauf, den Startort eines synchronisierten Tabellenscans auszuwählen.

LWLock:SyncScanLock

Ein Prozess wartet darauf, die Startposition eines Scans in einer Tabelle für synchronisierte Scans abzurufen.

LWLock:TablespaceCreate

Ein Prozess wartet darauf, einen Tablespace zu erstellen oder zu löschen.

LWLock:TablespaceCreateLock

Ein Prozess wartet darauf, den Tablespace zu erstellen oder zu löschen.

LWLock: Schreiben Sie ----sep----:tbm

Ein Prozess wartet auf eine gemeinsame Iteratorsperre für eine Baum-Bitmap (TBM).

LWLock:TwoPhaseState

Ein Prozess wartet darauf, den Status vorbereiteter Transaktionen zu lesen oder zu aktualisieren.

LWLock:TwoPhaseStateLock

Ein Prozess wartet darauf, den Status vorbereiteter Transaktionen zu lesen oder zu aktualisieren.

LWLock:tbm ----sep----:wal_insert

Ein Prozess wartet darauf, das Write-Ahead-Log (WAL) in einen Speicherpuffer einzufügen.

LWLock: Kartierung WALBuf

Ein Prozess wartet darauf, eine Seite in Write-Ahead-Log(WAL)-Puffern zu ersetzen.

LWLock:WALBufMappingLock

Ein Prozess wartet darauf, eine Seite in Write-Ahead-Log(WAL)-Puffern zu ersetzen.

LWLock:WALInsert

Ein Prozess wartet darauf, Write-Ahead-Protokolldaten (WAL) in einen Speicherpuffer einzufügen.

LWLock:WALWrite

Ein Prozess wartet darauf, dass Write-Ahead-Protokoll-Puffer (WAL) auf die Platte geschrieben werden.

LWLock: WALWrite Sperren

Ein Prozess wartet darauf, dass Write-Ahead-Protokoll-Puffer (WAL) auf die Platte geschrieben werden.

LWLock:WrapLimitsVacuum

Ein Prozess wartet darauf, die Limits für Transaktions-ID und Multixact-Verbrauch zu aktualisieren.

LWLock:WrapLimitsVacuumLock

Ein Prozess wartet darauf, die Limits für Transaktions-ID und Multixact-Verbrauch zu aktualisieren.

LWLock:XactBuffer

Ein Prozess wartet in einem einfachen Puffer, I/O der zuletzt zuletzt verwendet wurde (SLRU), auf einen Transaktionsstatus.

LWLockEin Prozess wartet auf einen einfachen Puffer, der zuletzt zuletzt verwendet wurde (SLRU), auf einen Transaktionsstatus. ----sep----:xactSLRU

Ein Prozess wartet darauf, für einen Transaktionsstatus auf den einfachen, selten verwendeten Cache (SLRU-Cache) zuzugreifen.

LWLock:XactTruncation

Ein Prozess wartet darauf, pg_xact_status auszuführen oder die älteste verfügbare Transaktions-ID zu aktualisieren.

LWLock:XidGen

Ein Prozess wartet darauf, eine neue Transaktions-ID zuzuweisen.

LWLock:XidGenLock

Ein Prozess wartet darauf, eine Transaktions-ID zu vergeben oder zuzuweisen.

Auszeit: BaseBackupThrottle

Ein Prozess wartet während der Basissicherung, wenn die Aktivität gedrosselt wird.

Auszeit: PgSleep

Ein Backend-Prozess hat die Funktion pg_sleep aufgerufen und wartet auf das Ablaufen des Sleep-Timeouts. Weitere Informationen finden Sie unter Timeout:PgSleep.

Auszeit: RecoveryApplyDelay

Ein Prozess wartet aufgrund einer Verzögerungseinstellung darauf, während der Wiederherstellung ein Write-Ahead-Protokoll (WAL) anzuwenden.

Auszeit: RecoveryRetrieveRetryInterval

Ein Prozess wartet während der Wiederherstellung, wenn Write-Ahead-Protokoll-Daten (WAL) von keiner Quelle (pg_wal, Archiv oder Stream) verfügbar sind.

Auszeit: VacuumDelay

Ein Prozess wartet in einem preis-basierten Vakuum-Verzögerungspunkt.

Eine vollständige Liste der PostgreSQL-Warteereignisse finden Sie unter Der Statistikkollektor > Warteereignistabellen in der PostgreSQL-Dokumentation.

Aktualisierungen der Datenbank-Engine für Amazon Aurora PostgreSQL

Nachstehend finden Sie Informationen zu Versionen und Aktualisierungen der Amazon-Aurora-PostgreSQL-Engine. Sie erfahren auch, wie Sie Ihre Aurora-PostgreSQL-Engine aktualisieren. Weitere Informationen zu Aurora-Versionen im Allgemeinen finden Sie unter Amazon-Aurora-Versionen.

> **Tip**
>
> Sie können die für ein DB-Cluster-Upgrade erforderlichen Ausfallzeiten minimieren, indem Sie eine Blau/Grün-Bereitstellung verwenden. Weitere Informationen finden Sie unter Verwendung von Blue/Green Bereitstellungen für Datenbankaktualisierungen.

Themen

- Identifizieren der Versionen von Amazon Aurora PostgreSQL
- Amazon Aurora PostgreSQL Releases und Engine Versionen
- Erweiterungsversionen für Amazon Aurora PostgreSQL
- Upgrade von DB-Clustern von Amazon Aurora PostgreSQL
- Verwenden einer Aurora PostgreSQL Long-Term Support (LTS, Langzeit-Support)-Version

Identifizieren der Versionen von Amazon Aurora PostgreSQL

Amazon Aurora enthält bestimmte Funktionen, die für Aurora allgemein sind und für alle Aurora-DB-Cluster verfügbar sind. Amazon Aurora umfasst andere Funktionen, die spezifisch für eine bestimmte Datenbank-Engine sind, die Aurora unterstützt. Diese Funktionen stehen nur denjenigen Aurora-DB-Clustern zur Verfügung, die diese Datenbank-Engine verwenden, wie z. B. Aurora PostgreSQL.

Eine Aurora-Datenbank-Version hat normalerweise zwei Versionsnummern, die Aurora-Versionsnummer und die Versionsnummer der Datenbank-Engine. Wenn eine Aurora-PostgreSQL-Version eine Aurora-Versionsnummer hat, befindet sie sich in der Amazon Aurora PostgreSQL Releases und Engine Versionen-Auflistung hinter der Engine-Versionsnummer.

Themen

- Aurora-Versionsnummer

- Versionsnummern der PostgreSQL-Engine

Aurora-Versionsnummer

Aurora-Versionsnummern verwenden das Namensschema *major.minor.patch* (Haupt.Neben.Patch). Eine Aurora-Patch-Version enthält wichtige Bugfixes, die einer Nebenversion nach ihrer Veröffentlichung hinzugefügt werden. Weitere Informationen zu Haupt-, Neben- und Patch-Versionen von Amazon Aurora finden Sie unter Amazon-Aurora-Hauptversionen, Amazon-Aurora-Nebenversionen und Amazon-Aurora-Patchversionen.

Sie können die Aurora-Versionsnummer Ihrer Aurora-PostgreSQL-DB-Instance mit der folgenden SQL-Abfrage herausfinden:

```
postgres=> SELECT aurora_version();
```

Ab der Veröffentlichung der PostgreSQL-Versionen 13.3, 12.8, 11.13 und 10.18 und für alle späteren Versionen sind die Aurora-Versionsnummern genauer auf die PostgreSQL-Engine-Version abgestimmt. Wenn Sie beispielsweise einen Aurora-PostgreSQL-13.3-DB-Cluster abfragen, wird Folgendes zurückgegeben:

```
aurora_version
----------------
 13.3.1
(1 row)
```

Frühere Versionen, wie Aurora-PostgreSQL-10.14-DB-Cluster, liefern Versionsnummern wie folgende:

```
aurora_version
----------------
 2.7.3
(1 row)
```

Versionsnummern der PostgreSQL-Engine

Ab PostgreSQL 10 verwenden PostgreSQL-Datenbank-Engine-Versionen das Nummernschema *Haupt.Neben* für alle Releases. Beispiele: PostgreSQL 10.18, PostgreSQL 12.7 und PostgreSQL 13.3.

Releases vor PostgreSQL 10 verwendeten das Nummernschema *Haupt*.*Haupt*.*Neben*, in dem die ersten beiden Ziffern die Nummer der Hauptversion und die dritte Ziffer die Nebenversion angeben. PostgreSQL 9.6 war beispielsweise eine Hauptversion, wobei die Nebenversionen 9.6.21 oder 9.6.22 durch die dritte Ziffer gekennzeichnet waren.

> **Note**
>
> Die PostgreSQL-Engine-Version 9.6 wird nicht mehr unterstützt. Informationen zum Upgrade finden Sie unter Upgrade von DB-Clustern von Amazon Aurora PostgreSQL. Versionsrichtlinien und Release-Zeitpläne finden Sie unter Wie lange bleiben Amazon-Aurora-Hauptversionen verfügbar?.

Sie können die PostgreSQL-Datenbank-Engine-Versionsnummer mit der folgenden SQL-Abfrage herausfinden:

```
postgres=> SELECT version();
```

Für einen Aurora-PostgreSQL-13.3-DB-Cluster sind die Ergebnisse wie folgt:

```
version
------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
(1 row)
```

Amazon Aurora PostgreSQL Releases und Engine Versionen

Versionen der Amazon-Aurora-PostgreSQL-kompatiblen Edition werden regelmäßig aktualisiert. Updates werden auf Aurora-PostgreSQL-DB-Cluster während des Wartungszeitraums angewendet. Wann Aktualisierungen angewendet werden, hängt von der Art der Aktualisierung, der AWS-Region, der Einstellung für das Wartungsfenster und der Einstellung für den DB-Cluster ab. Viele der aufgelisteten Versionen enthalten sowohl eine PostgreSQL-Versionsnummer als auch eine Amazon-Aurora-Versionsnummer. Ab der Veröffentlichung der PostgreSQL-Versionen 13.3, 12.8, 11.13, 10.18 und für alle anderen späteren Versionen werden jedoch keine Aurora-Versionsnummern mehr verwendet. Informationen zum Ermitteln der Versionsnummern Ihrer Aurora-PostgreSQL-Datenbank finden Sie unter Identifizieren der Versionen von Amazon Aurora PostgreSQL.

Hinweise zu Erweiterungen und Modulen finden Sie unter Erweiterungsversionen für Amazon Aurora PostgreSQL.

> **Note**
>
> Informationen zu den Versionsrichtlinien und Veröffentlichungszeitplänen von Amazon Aurora finden Sie unter Wie lange bleiben Amazon-Aurora-Hauptversionen verfügbar?.

Informationen zu Support für Amazon Aurora finden Sie unter Häufig gestellte Fragen zu Amazon RDS.

Um festzustellen, welche Aurora-PostgreSQL-DB-Engine-Versionen in einer AWS-Region verfügbar sind, verwenden Sie den AWS CLI-Befehl describe-db-engine-versions. Beispiel:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].
[EngineVersion]' --output text --region aws-region
```

Eine Liste der AWS-Regionen finden Sie unter Aurora PostgreSQL-Verfügbarkeit in Regionen.

Weitere Informationen zu den PostgreSQL-Versionen, die in Aurora PostgreSQL verfügbar sind, finden Sie in den Versionshinweisen für Aurora PostgreSQL.

Erweiterungsversionen für Amazon Aurora PostgreSQL

Sie können verschiedene PostgreSQL-Erweiterungen installieren und konfigurieren, um sie mit Aurora-PostgreSQL-DB-Clustern zu verwenden. Sie können beispielsweise die PostgreSQL-Erweiterung `pg_partman` verwenden, um die Erstellung und Pflege von Tabellenpartitionen zu automatisieren. Weitere Informationen über diese und andere für Aurora PostgreSQL verfügbare Erweiterungen finden Sie unter Arbeiten mit Erweiterungen und Fremddaten-Wrappern.

Weitere Informationen zu den PostgreSQL-Erweiterungen, die von Aurora PostgreSQL unterstützt werden, finden Sie unter Versionen der Erweiterungen für Amazon Aurora PostgreSQL in den Versionshinweisen für Aurora PostgreSQL.

Upgrade von DB-Clustern von Amazon Aurora PostgreSQL

Amazon Aurora stellt neue Versionen des PostgreSQL-Datenbankmoduls AWS-Regionen erst nach umfangreichen Tests zur Verfügung. Sie können Ihre Aurora-PostgreSQL-DB-Cluster auf die neue Version aktualisieren, wenn diese in Ihrer Region verfügbar ist.

Abhängig von der Version von Aurora PostgreSQL, die Ihr DB-Cluster derzeit ausführt, ist ein Upgrade auf die neue Version entweder ein Nebenversions- oder ein Hauptversions-Upgrade. Zum Beispiel ist das Upgrade eines DB-Clusters von Aurora PostgreSQL 11.15 auf Aurora PostgreSQL 13.6 ein Hauptversions-Upgrade. Das Upgrade eines DB-Clusters von Aurora PostgreSQL 13.3 auf Aurora PostgreSQL 13.7 hingegen ist ein Nebenversions-Upgrade. In den folgenden Themen finden Sie Informationen darüber, wie beide Arten von Upgrades durchgeführt werden.

Inhalt

- Übersicht über die Upgrade-Prozesse für Aurora-PostgreSQL
- Abrufen einer Liste der verfügbaren Versionen in Ihrem AWS-Region
- Durchführen eines Hauptversions-Upgrades
  - Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion
  - Empfehlungen für die Zeit nach dem Upgrade
  - Upgrade der Aurora-PostgreSQL-Engine auf eine neue Hauptversion
    - Hauptversions-Upgrades für globale Datenbanken
- Durchführen eines Nebenversions-Upgrades
  - Vor dem Durchführen eines Nebenversions-Upgrades
  - So führen Sie Upgrades von Nebenversionen durch und wenden Patches an
  - Nebenversions-Upgrades und Zero-Downtime-Patching
  - Einschränkungen des Patchens ohne Ausfallzeiten
  - Durchführen eines Upgrades der Aurora-PostgreSQL-Engine auf eine neue Nebenversion
- Aktualisieren von PostgreSQL-Erweiterungen
- Alternatives Blau/Grün-Upgradeverfahren

Übersicht über die Upgrade-Prozesse für Aurora-PostgreSQL

Zwischen Haupt- und Nebenversions-Upgrades gibt es folgende Unterschiede:

Nebenversions-Upgrades und -Patches

> Nebenversions-Upgrades und -Patches enthalten nur Änderungen, die mit bestehenden Anwendungen abwärtskompatibel sind. Nebenversions-Upgrades und -Patches stehen Ihnen erst zur Verfügung, nachdem Aurora PostgreSQL sie getestet und genehmigt hat.

Upgrades von Nebenversionen können von Aurora automatisch angewendet werden. Wenn Sie einen neuen Aurora-PostgreSQL-DB-Cluster erstellen, ist die Option Nebenversions-Upgrade aktivieren standardmäßig aktiviert. Sofern Sie diese Option nicht manuell deaktivieren, führt Aurora während Ihres geplanten Wartungsfensters regelmäßig automatische Upgrades von Nebenversionen durch. Weitere Informationen zur Option für das automatische Nebenversions-Upgrade (Automatic Minor Version Upgrade, AmVU) und darüber, wie Sie Ihren Aurora-DB-Cluster ändern, um diese verwenden zu können, finden Sie unter Automatische Nebenversions-Upgrades für Aurora-DB-Cluster.

Wenn das automatische Upgrade der Nebenversion für Ihren Aurora-PostgreSQL-DB-Cluster nicht aktiviert ist, wird Aurora PostgreSQL nicht automatisch auf die neue Nebenversion aktualisiert. Stattdessen werden Sie von Aurora zum Upgrade aufgefordert, wenn eine neue Nebenversion in Ihrer AWS-Region veröffentlicht wird und Ihr DB-Cluster von Aurora PostgreSQL eine ältere Nebenversion ausführt. Dazu wird den Wartungsaufgaben für Ihren Cluster eine Empfehlung hinzugefügt.

Patches gelten nicht als Upgrade und werden nicht automatisch angewendet. Aurora PostgreSQL fordert Sie auf, alle Patches anzuwenden, indem es den Wartungsaufgaben für Ihren Aurora-PostgreSQL-DB-Cluster eine Empfehlung hinzufügt. Weitere Informationen finden Sie unter So führen Sie Upgrades von Nebenversionen durch und wenden Patches an.

> **Note**
>
> Patches, die Sicherheits- oder andere kritische Probleme lösen, werden ebenfalls als Wartungsaufgaben hinzugefügt. Diese Patches sind jedoch erforderlich. Stellen Sie sicher, dass Sie Sicherheitspatches auf Ihren Aurora-PostgreSQL-DB-Cluster anwenden, wenn diese in Ihren ausstehenden Wartungsaufgaben verfügbar sind. Automatische Nebenversions-Upgrades werden auf die standardmäßige Nebenversion durchgeführt.

Beim Upgrade-Prozess kann es zu kurzen Ausfällen kommen, während jede Instance im Cluster auf die neue Version aktualisiert wird. Nach den Aurora-PostgreSQL-Versionen 14.3.3, 13.7.3, 12.11.3, 11.16.3, 10.21.3 und anderen höheren Versionen dieser Nebenversionen verwendet der Upgrade-Prozess jedoch die Zero Downtime Patching (ZDP)-Funktion. Diese Funktion minimiert Ausfälle und eliminiert sie in den meisten Fällen vollständig. Weitere Informationen finden Sie unter Nebenversions-Upgrades und Zero-Downtime-Patching. Weitere Informationen zu den

unterstützten Features und Einschränkungen von ZDP finden Sie unter Einschränkungen des Patchens ohne Ausfallzeiten.

Hauptversions-Upgrades

Im Gegensatz zu Upgrades und Patches für Nebenversionen verfügt Aurora PostgreSQL über keine automatische Upgrade-Option für Hauptversionen. Hauptversions-Upgrades können Datenbankänderungen enthalten, die mit vorhandenen Anwendungen nicht abwärtskompatibel sind. Neue Funktionalität kann dazu führen, dass Ihre vorhandenen Anwendungen nicht mehr ordnungsgemäß funktionieren.

Zur Vermeidung von Problemen empfehlen wir dringend, dem in Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion beschriebenen Verfahren zu folgen, bevor Sie die DB-Instances in Ihren Aurora-PostgreSQL-DB-Clustern upgraden. Stellen Sie zunächst sicher, dass Ihre Anwendungen mit der neuen Version ausgeführt werden können, indem Sie diesem Verfahren folgen. Dann können Sie Ihren Aurora-PostgreSQL-DB-Cluster manuell auf die neue Version aktualisieren.

Beim Upgrade-Prozess kann es zu kurzen Ausfällen kommen, während alle Instances im Cluster auf die neue Version aktualisiert werden. Der vorläufige Planungsprozess braucht ebenfalls Zeit. Wir empfehlen Ihnen, Upgrade-Aufgaben immer während des Wartungsfensters Ihres Clusters oder bei minimalem Betrieb auszuführen. Weitere Informationen finden Sie unter Durchführen eines Hauptversions-Upgrades.

> **Note**
>
> Sowohl Upgrades von Neben- als auch von Hauptversionen können kurze Ausfälle verursachen. Aus diesem Grund empfehlen wir dringend, dass Sie Upgrades während Ihres Wartungsfensters oder in anderen Zeiträumen geringer Auslastung durchführen oder planen.

DB-Cluster von Aurora PostgreSQL erfordern gelegentlich Betriebssystemupdates. Diese Updates können eine neuere Version der Glibc-Bibliothek umfassen. Bei solchen Updates empfehlen wir Ihnen, die unter In Aurora PostgreSQL unterstützte Sortierungen beschriebenen Richtlinien zu befolgen.

Abrufen einer Liste der verfügbaren Versionen in Ihrem AWS-Region

Sie können eine Liste aller Engine-Versionen abrufen, die als Upgrade-Ziele für Ihren Aurora PostgreSQL-DB-Cluster verfügbar sind, indem Sie Sie AWS-Region mit dem describe-db-engine-versions AWS CLI folgenden Befehl abfragen.

Für Linux, macOS oder Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Für Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

Um beispielsweise die gültigen Upgrade-Ziele für einen Aurora PostgreSQL-DB-Cluster der Version 12.10 zu identifizieren, führen Sie den folgenden Befehl aus: AWS CLI

Für Linux, macOS oder Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version 12.10 \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Für Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version 12.10 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

In der folgenden Tabelle finden Sie die Ziele der Haupt- und Nebenversions-Upgrades, die für verschiedene Aurora-PostgreSQL-DB-Versionen verfügbar sind. Um die Kompatibilität aufrechtzuerhalten, werden nicht alle Versionen als Upgrade-Ziele angeboten. Aurora PostgreSQL führt mit jeder vierteljährlichen Veröffentlichung von Nebenversionen neue Features und Bugfixes ein. Weitere Informationen zu Nebenversionen von Aurora-PostgreSQL finden Sie unter Versionshinweise für Aurora PostgreSQL.

Aktuelle Quellversion	Upgrade-Ziele
17.6	Keine
17,5	17,6
17,4	17.6, 17.5, 17.4
16,10	Keine
16,9	17.6, 17.5 16,10
16,8	17.6, 17.5, 17.4 16.10, 16.9
16,6	17.6, 17.5, 17.4 16.10, 16.9, 16.8
16,4	17.6, 17.5, 17.4 16.10, 16.9, 16.8, 16.6
16,3	17.6, 17.5, 17.4 16.10, 16.9, 16.8, 16.6, 16.4
16,2	17.6, 17.5, 17.4 16,10 , 16,9, 16,816,6, 16,4, 16,3
16,1	17.6, 17.5, 17.4

Aktuelle Quellversion	Upgrade-Ziele
	16,10 , 16,9, 16,8, 16,6, 16,4, 16,3, 16,2
15,13	17,6 16,10 15,14
15,12	17,6 16,10 15,14
15,1	17,6 16,10 15,14
15,8	17,6 16,10 15,14
15,7	17,6 16,10 15,14
15,6	17,6 16,10 15,14

Aktuelle Quellversion	Upgrade-Ziele
15,5	17,6 16,10 15,14
15,4	17,6 16,10 15,14
15,3	17,6 16,10 15,14
15,2	17,6 16,10 15,14
14,19	Keine
14,18	17,6 16,10 15,14 14,19
14,17	17,6 16,10 15,14 14,19

Aktuelle Quellversion	Upgrade-Ziele
14,15	17,6 16,10 15,14 14,19
14,13	17,6 16,10 15,14 14,19
14,12	17,6 16,10 15,14 14,19
14,11	17,6 16,10 15,14 14,19
14,10	17,6 16,10 15,14 14,19

Aktuelle Quellversion	Upgrade-Ziele
14,9	17,6 16,10 15,14 14,19
14,8	17,6 16,10 15,14 14,19
14,7	17,6 16,10 15,14 14,19
14,6	17,6 16,10 15,14 14,19
14,5	17,6 16,10 15,14 14,19

Aktuelle Quellversion	Upgrade-Ziele
14,4	17,6 16,10 15,14 14,19
14,3	17,6 16,10 15,14 14,19
13,22	Keine
13,21	17,6 16,10 15,14 14,19 13,22
13,20	17,6 16,10 15,14 14,19 13,22

Aktuelle Quellversion	Upgrade-Ziele
13,18	17,6 16,10 15,14 14,19 13,22
13,16	17,6 16,10 15,14 14,19 13,22
13,15	17,6 16,10 15,14 14,19 13,22
13,14	17,6 16,10 15,14 14,19 13,22

Aktuelle Quellversion	Upgrade-Ziele
13,13	17,6 16,10 15,14 14,19 13,22
13,12	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13
13,11	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12
13,10	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11

Aktuelle Quellversion	Upgrade-Ziele
13,9	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13,13.12, 13.11, 13.10
13,8	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9
13,7	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5, 14.4, 14.3 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8

Aktuelle Quellversion	Upgrade-Ziele
12,22	17.5, 17.4 16.9, 16.8, 16.6 15.13, 15.12, 15.10 14.18, 14.17, 14.15 13.21, 13.20, 13.18
12,20	17.5, 17.4 16.9, 16.8, 16.6, 16.4 15.13, 15.12, 15.10, 15.8 14.15, 14.13 13.21, 13.20, 13.18, 13.16 12,22
12,19	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3 15.13, 15.12, 15.10, 15.8, 15.7 14.18, 14.17, 14.15, 14.13, 14.12 13.21, 13.20, 13.18, 13.16, 13.15 12.22, 12.20

Aktuelle Quellversion	Upgrade-Ziele
12,18	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2 15.13, 15.12, 15.10, 15.8, 15.7, 15.6 14.18, 14.17, 14.15, 14.13, 14.12, 14.11 13.21, 13.20, 13.18, 13.16, 13.15, 13.14 12.22, 12.20, 12.19
12,17	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13 12.22, 12.20, 12.19, 12.18
12,16	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12 12.22, 12.20, 12.19, 12.18, 12.17

Aktuelle Quellversion	Upgrade-Ziele
12,15	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11 12.22, 12.20, 12.19, 12.18, 12.17, 12.16
12,14	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15
12,13	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14

Aktuelle Quellversion	Upgrade-Ziele
12,12	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6, 14.5 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13
12,11	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.5, 14.4, 14.3 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8, 13.7 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12

Aktuelle Quellversion	Upgrade-Ziele
12,9	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7 13.21, 13.20, 13.18, 13.16, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9, 13.8, 13.7 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12, 12.11
11,21	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12 12.22, 12.20, 12.19, 12.18, 12.17, 12.16

Aktuelle Quellversion	Upgrade-Ziele
11.9	17.5, 17.4 16.9, 16.8, 16.6, 16.4, 16.3, 16.2, 16.1 15.13, 15.12, 15.10, 15.8, 15.7, 15.6, 15.5, 15.4, 15.3, 15.2 14.18, 14.17, 14.15, 14.13, 14.12, 14.11, 14.10, 14.9, 14.8, 14.7, 14.6 13.21, 13.20, 13.18, 13.16, 13.15, 13.14, 13.13, 13.12, 13.11, 13.10, 13.9 12.22, 12.20, 12.19, 12.18, 12.17, 12.16, 12.15, 12.14, 12.13, 12.12, 12.11, 12.09 11,21

Überprüfen Sie für jede Version, die Sie in Betracht ziehen, immer die Verfügbarkeit der DB-Instance-Klasse Ihres Clusters. Beispielsweise wird `db.r4` für Aurora PostgreSQL 13 nicht unterstützt. Wenn Ihr DB-Cluster von Aurora PostgreSQL derzeit eine db.r4-Instance-Klasse verwendet, müssen Sie auf db.r5 umstellen, bevor Sie ein Upgrade ausführen. Weitere Informationen zu DB-Instance-Klassen, einschließlich der Frage, welche Graviton2-basiert und welche Intel-basiert sind, finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Durchführen eines Hauptversions-Upgrades

Hauptversions-Upgrades können Datenbankänderungen enthalten, die möglicherweise nicht mit früheren Versionen der Datenbank abwärtskompatibel sind. Neue Funktionalität einer neuen Version kann dazu führen, dass Ihre vorhandenen Anwendungen nicht mehr ordnungsgemäß funktionieren. Zur Vermeidung von Problemen wendet Amazon Aurora Hauptversions-Upgrades nicht automatisch an. Wir empfehlen vielmehr, dass Sie ein Hauptversions-Upgrade sorgfältig planen, indem Sie die folgenden Schritte ausführen:

1. Wählen Sie die gewünschte Hauptversion aus der Liste der verfügbaren Ziele aus, die für Ihre Version in der Tabelle aufgeführt sind. Sie können eine genaue Liste der Versionen abrufen, die in Ihrer AWS-Region für Ihre aktuelle Version verfügbar sind, indem Sie die AWS CLI verwenden. Details hierzu finden Sie unter Abrufen einer Liste der verfügbaren Versionen in Ihrem AWS-Region.

2. Stellen Sie sicher, dass Ihre Anwendungen bei einer Testbereitstellung der neuen Version erwartungsgemäß funktionieren. Weitere Informationen zum vollständigen Prozess finden Sie unter Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion.
3. Nachdem Sie überprüft haben, dass Ihre Anwendungen bei der Testbereitstellung wie erwartet funktionieren, können Sie Ihren Cluster aktualisieren. Details hierzu finden Sie unter Upgrade der Aurora-PostgreSQL-Engine auf eine neue Hauptversion.

> **Note**
>
> Sie können ein Hauptversions-Upgrade von auf Babelfish für Aurora PostgreSQL 13 basierende Versionen ab 13.6 auf Versionen durchführen, die auf Aurora PostgreSQL 14 basieren, und zwar ab 14.6. Babelfish für Aurora PostgreSQL 13.4 und 13.5 unterstützen kein Hauptversions-Upgrade.

Sie können eine Liste der Engine-Versionen abrufen, die als Upgrade-Ziele für Ihren Aurora-PostgreSQL-DB-Cluster verfügbar sind, indem Sie Ihre AWS-Region mit dem folgenden AWS CLI-Befehl describe-db-engine-versions wie folgt abfragen.

Für Linux, macOS oder Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
  --output text
```

Für Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
  --output text
```

In einigen Fällen ist die Version, auf die Sie upgraden möchten, kein Ziel für Ihre aktuelle Version. Verwenden Sie in solchen Fällen die Informationen in versions table, um Nebenversions-Upgrades

durchzuführen, bis sich Ihr Cluster in einer Version befindet, die Ihr ausgewähltes Ziel in seiner Zielzeile enthält.

Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion

Jede neue Hauptversion enthält Verbesserungen am Abfrageoptimierer, die auf Leistungssteigerung ausgelegt sind. Ihr Workload kann jedoch Abfragen enthalten, die in der neuen Version zu einem Plan mit schlechter Leistung führen. Aus diesem Grund empfehlen wir Ihnen, die Leistung zu testen und zu überprüfen, bevor Sie ein Upgrade in der Produktion durchführen. Sie können die Stabilität des Abfrageplans über Versionen hinweg verwalten, indem Sie die Erweiterung Query Plan Management (QPM) verwenden, wie in Sicherstellen der Planstabilität nach einem größeren Versions-Upgrade ausführlich beschrieben.

Bevor Sie Ihre Produktions-DB-Cluster von Aurora PostgreSQL auf eine neue Hauptversion upgraden, wird dringend empfohlen, das Upgrade zu testen, um sicherzustellen, dass alle Ihre Anwendungen ordnungsgemäß funktionieren:

1. Halten Sie eine versionskompatible Parametergruppe bereit.

   Wenn Sie eine benutzerdefinierte DB-Instance- oder DB-Cluster-Parametergruppe verwenden, haben Sie zwei Möglichkeiten:

   a. Geben Sie die Standard-DB-Instance, die DB-Cluster-Parametergruppe oder beides für die neue DB-Engine-Version an.

   b. Erstellen Sie eine eigene benutzerdefinierte Parametergruppe für die neue DB-Engine-Version.

2. Suchen Sie nach ungültigen Datenbanken und löschen Sie alle vorhandenen.

   Die Spalte `datconnlimit` im Katalog `pg_database` enthält den Wert `-2`, um Datenbanken, die während eines `DROP DATABASE`-Vorgangs unterbrochen wurden, als ungültig zu kennzeichnen. Verwenden Sie die folgende Abfrage, um zu überprüfen, ob ungültige Datenbanken vorhanden sind.

```
SELECT
    datname
FROM
    pg_database
WHERE
    datconnlimit = - 2;
```

Wenn die Abfrage Datenbanknamen zurückgibt, sind diese Datenbanken ungültig. Verwenden Sie die `DROP DATABASE invalid_db_name`-Anweisung, um ungültige Datenbanken zu löschen. Sie können die folgende dynamische Anweisung ausführen, um alle ungültigen Datenbanken zu löschen.

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

3. Auf nicht unterstützte Verwendung prüfen:

   - Übernehmen Sie oder machen Sie alle offenen vorbereiteten Transaktionen rückgängig, bevor Sie ein Upgrade durchführen. Mit der folgenden Abfrage können Sie sicherstellen, dass für Ihre Instance keine geöffneten vorbereiteten Transaktionen vorhanden sind.

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

   - Entfernen Sie alle Anwendungen der reg*-Datentypen, bevor Sie versuchen, einen Upgrade durchzuführen. Bis auf `regtype` und `regclass` ist kein Upgrade der reg*-Datentypen möglich. Das Dienstprogramm pg_upgrade (das von Amazon Aurora zur Durchführung des Upgrades verwendet wird) kann diesen Datentyp nicht beibehalten. Weitere Informationen zu diesem Dienstprogramm finden Sie unter pg_upgrade in der PostgreSQL-Dokumentation.

     Um zu überprüfen, dass keine Anwendungen der nicht unterstützten reg*-Datentypen vorhanden sind, geben Sie für jede Datenbank die folgende Abfrage aus.

     ```
     SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
      pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid
           AND NOT a.attisdropped
           AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                              'pg_catalog.regprocedure'::pg_catalog.regtype,
                              'pg_catalog.regoper'::pg_catalog.regtype,
                              'pg_catalog.regoperator'::pg_catalog.regtype,
                              'pg_catalog.regconfig'::pg_catalog.regtype,
                              'pg_catalog.regdictionary'::pg_catalog.regtype)
           AND c.relnamespace = n.oid
     ```

```
      AND n.nspname NOT IN ('pg_catalog', 'information_schema');
```

- Wenn Sie ein Upgrade eines DB-Clusters mit Aurora PostgreSQL Version 10.18 oder höher durchführen, für den die Erweiterung `pgRouting` installiert ist, entfernen Sie diese Erweiterung, bevor Sie auf Version 12.4 oder höher aktualisieren.

  Wenn Sie ein Upgrade eines DB-Clusters mit Aurora-PostgreSQL-Version 10.x oder höher durchführen, für die die Erweiterung `pg_repack` Version 1.4.3 installiert ist, entfernen Sie diese Erweiterung, bevor Sie auf eine höhere Version aktualisieren.

4. Suchen Sie nach den Datenbanken Template1 und Template0.

   Für ein erfolgreiches Upgrade müssen die Datenbanken Template 1 und Template 0 vorhanden sein und sollten als Vorlage aufgeführt werden. Verwenden Sie den folgenden Befehl, um dies zu überprüfen:

```
SELECT datname, datistemplate FROM pg_database;

datname    | datistemplate
-----------+---------------
template0  | t
rdsadmin   | f
template1  | t
postgres   | f
```

   In der Befehlsausgabe sollte der Wert `datistemplate` für die Datenbanken template1 und template0 `t` lauten.

5. Entfernen von logischen Replikations-Slots.

   Der Upgrade-Vorgang kann nicht fortgesetzt werden, wenn der Aurora PostgreSQL DB-Cluster logische Replikations-Slots verwendet. Logische Replikations-Slots werden in der Regel für kurzfristige Datenmigrationsaufgaben verwendet, z. B. für die Migration von Daten mit AWS DMS oder für die Replikation von Tabellen aus der Datenbank in Data Lakes, BI-Tools oder andere Ziele. Vergewissern Sie sich vor dem Upgrade, dass Sie den Zweck der vorhandenen logischen Replikations-Slots kennen, und bestätigen Sie, dass sie gelöscht werden dürfen. Sie können mit der folgenden Abfrage nach logischen Replikationssteckplätzen suchen:

```
SELECT * FROM pg_replication_slots;
```

Wenn die logischen Replikations-Slots noch verwendet werden, sollten Sie sie nicht löschen, und Sie können mit dem Upgrade nicht fortfahren. Wenn die logischen Replikations-Slots jedoch nicht benötigt werden, können Sie sie mit folgendem SQL-Befehl löschen:

```
SELECT pg_drop_replication_slot(slot_name);
```

Für logische Replikationsszenarien, die die `pglogical`-Erweiterung verwenden, müssen außerdem Slots vom Herausgeberknoten gelöscht werden, damit ein Hauptversions-Upgrade auf diesem Knoten erfolgreich durchgeführt werden kann. Sie können den Replikationsprozess jedoch nach dem Upgrade vom Abonnentenknoten aus neu starten. Weitere Informationen finden Sie unter Wiederherstellung der logischen Replikation nach einem Hauptversions-Upgrade.

6. Führen Sie ein Backup durch.

   Der Aktualisierungsprozess erstellt während des Upgrades einen DB-Cluster-Snapshot des DB-Clusters. Wenn Sie vor dem Upgrade auch ein manuelles Backup durchführen möchten, finden Sie weitere Informationen unter Erstellen eines DB-Cluster-Snapshots.

7. Aktualisieren Sie bestimmte Erweiterungen auf die neueste verfügbare Version, bevor Sie das Upgrade der Hauptversion durchführen. Zu den Erweiterungen, die aktualisiert werden sollen, gehören folgende:

   - `pgRouting`
   - `postgis_raster`
   - `postgis_tiger_geocoder`
   - `postgis_topology`
   - `address_standardizer`
   - `address_standardizer_data_us`

   Führen Sie den folgenden Befehl für jede derzeit installierte Erweiterung aus.

   ```
   ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
   ```

   Weitere Informationen finden Sie unter Aktualisieren von PostgreSQL-Erweiterungen. Weitere Informationen zum Aktualisieren von PostGIS finden Sie unterSchritt 6: Upgrade der PostGIS-Erweiterung.

8. Wenn Sie ein Upgrade auf Version 11.x durchführen, löschen Sie die Erweiterungen, die nicht unterstützt werden, bevor Sie das Upgrade der Hauptversion durchführen. Die zu löschenden Erweiterungen umfassen:
   - `chkpass`
   - `tsearch2`
9. Löschen Sie `unknown`-Datentypen, abhängig von Ihrer Zielversion.

   PostgreSQL-Version 10 unterstützt den Datentyp `unknown` nicht. Wenn eine Datenbank der Version 9.6 den Datentyp `unknown` verwendet, wird bei einem Upgrade auf Version 10 eine Fehlermeldung wie die folgende angezeigt.

   ```
   Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
   The instance could not be upgraded because the 'unknown' data type is used in user
    tables.
   Please remove all usages of the 'unknown' data type and try again."
   ```

   Verwenden Sie den folgenden SQL-Code für jede Datenbank, um den `unknown`-Datentyp in Ihrer Datenbank zu finden, damit Sie solche Spalten entfernen oder in unterstützte Datentypen ändern können.

   ```
   SELECT n.nspname, c.relname, a.attname
       FROM pg_catalog.pg_class c,
       pg_catalog.pg_namespace n,
       pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
       a.atttypid = 'pg_catalog.unknown'::pg_catalog.regtype AND
       c.relkind IN ('r','m','c') AND
       c.relnamespace = n.oid AND
       n.nspname !~ '^pg_temp_' AND
       n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
    'information_schema');
   ```

10. Führen Sie ein Test-Upgrade durch.

    Es wird dringend empfohlen, das Upgrade der Hauptversion auf einem Duplikat Ihrer Produktionsdatenbank zu testen, bevor Sie versuchen, das Upgrade für Ihre Produktionsdatenbank auszuführen. Sie können die Ausführungspläne auf der duplizierten Testinstance auf mögliche Regressionen des Ausführungsplans überwachen und deren Leistung bewerten. Um ein Duplikat der Test-Instance zu erstellen, können Sie Ihre Datenbank

entweder aus einem aktuellen Snapshot wiederherstellen oder Ihre Datenbank klonen. Weitere Informationen finden Sie unter Wiederherstellung aus einem Snapshot oder Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster.

Weitere Informationen finden Sie unter Upgrade der Aurora-PostgreSQL-Engine auf eine neue Hauptversion.

11 Aktualisieren Sie Ihre Produktionsinstance.

Wenn das Hauptversions-Upgrade für den Testlauf erfolgreich ist, sollten Sie jetzt in der Lage sein, Ihre Produktionsdatenbank sicher zu aktualisieren. Weitere Informationen finden Sie unter Upgrade der Aurora-PostgreSQL-Engine auf eine neue Hauptversion.

> **Note**
>
> Während des Upgrade-Prozesses erstellt Aurora PostgreSQL einen DB-Cluster-Snapshot. Sie können während dieses Prozesses keine punktuelle Wiederherstellung Ihres Clusters durchführen. Sie können später eine Wiederherstellung nach dem Zeitpunkt vor dem Beginn des Upgrades und nach Abschluss des automatischen Snapshots Ihrer Instance durchführen. Sie können eine Point-in-Time-Wiederherstellung auf eine frühere Nebenversion jedoch nicht durchführen.

Für Informationen zu einem laufenden Upgrade können Sie mit Amazon RDS zwei Protokolle anzeigen, die von dem pg_upgrade-Dienstprogramm erstellt werden. Diese sind `pg_upgrade_internal.log` und `pg_upgrade_server.log`. Amazon Aurora hängt einen Zeitstempel an den Dateinamen für diese Protokolle an. Sie können diese Protokolle wie jedes andere Protokoll einsehen. Weitere Informationen finden Sie unter Überwachen von Amazon Aurora-Protokolldateien.

12 Aktualisieren Sie PostgreSQL-Erweiterungen. Bei einem PostgreSQL-Upgrade-Prozess werden keine PostgreSQL-Erweiterungen aktualisiert. Weitere Informationen finden Sie unter Aktualisieren von PostgreSQL-Erweiterungen.

Empfehlungen für die Zeit nach dem Upgrade

Nachdem Sie ein Upgrade der Hauptversion abgeschlossen haben, empfehlen wir Folgendes:

- Führen Sie die ANALYZE-Operation aus, um die Tabelle `pg_statistic` zu aktualisieren. Führen Sie diesen Schritt für jede Datenbank auf all Ihren PostgreSQL-DB-Instances durch. Optimizer-Statistiken werden während eines Hauptversionsupgrades nicht übertragen, daher müssen Sie alle Statistiken neu generieren, um Leistungsprobleme zu vermeiden. Führen Sie den Befehl ohne Parameter aus, um Statistiken für alle regulären Tabellen in der aktuellen Datenbank wie folgt zu generieren:

  ```
  ANALYZE VERBOSE;
  ```

  Das Flag `VERBOSE` ist optional, aber wenn Sie es verwenden, wird Ihnen der Fortschritt angezeigt. Weitere Informationen finden Sie unter ANALYZE in der PostgreSQL-Dokumentation.

  Wenn Sie bestimmte Tabellen analysieren, anstatt ANALYZE VERBOSE zu verwenden, führen Sie den Befehl ANALYZE für jede Tabelle wie folgt aus:

  ```
  ANALYZE table_name;
  ```

  Analysieren Sie bei partitionierten Tabellen immer die übergeordnete Tabelle. Dieser Prozess beinhaltet Folgendes:

  - Automatisches Stichprobenverfahren für Zeilen in allen Partitionen
  - Rekursives Aktualisieren der Statistiken für jede Partition
  - Verwalten wichtiger Planungsstatistiken auf übergeordneter Ebene

  In übergeordneten Tabellen werden zwar keine tatsächlichen Daten gespeichert, ihre Analyse ist jedoch für die Abfrageoptimierung von entscheidender Bedeutung. Wenn ANALYZE nur auf einzelnen Partitionen ausgeführt wird, kann dies zu einer schlechten Abfrageleistung führen, da der Optimierer nicht über die umfassenden Statistiken verfügt, die für eine effiziente partitionsübergreifende Planung erforderlich sind.

  > **Note**
  >
  > Führen Sie ANALYZE nach dem Upgrade auf Ihrem System aus, um Leistungsprobleme zu vermeiden.

- Wenn Sie auf PostgreSQL Version 10 aktualisiert haben, führen Sie `REINDEX` auf allen Hash-Indizes aus, die Sie besitzen. Hash-Indizes wurden in Version 10 geändert und müssen neu erstellt

werden. Um ungültige Hash-Indizes zu finden, führen Sie die folgende SQL-Anweisung für jede Datenbank aus, die Hash-Indizes enthält.

```
SELECT idx.indrelid::regclass AS table_name,
   idx.indexrelid::regclass AS index_name
FROM pg_catalog.pg_index idx
   JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
   JOIN pg_catalog.pg_am am ON am.oid = cls.relam
WHERE am.amname = 'hash'
AND NOT idx.indisvalid;
```

- Wir empfehlen, dass Sie Ihre Anwendung auf der aktualisierten Datenbank mit ähnlicher Workload testen, um sicherzustellen, dass alles wie erwartet funktioniert. Nachdem das Upgrade bestätigt wurde, können Sie diese Testinstance löschen.

Upgrade der Aurora-PostgreSQL-Engine auf eine neue Hauptversion

Wenn Sie den Upgrade-Prozess auf eine neue Hauptversion einleiten, erstellt Aurora PostgreSQL einen Snapshot des Aurora-DB-Clusters, bevor er Änderungen an Ihrem Cluster vornimmt. Dieser Snapshot wird nur für Hauptversions-Upgrades erstellt, nicht für Nebenversions-Upgrades. Wenn der Upgrade-Vorgang abgeschlossen ist, finden Sie diesen Snapshot unter den manuellen Snapshots, die unter Snapshots in der RDS-Konsole aufgeführt sind. Der Snapshot-Name enthält `preupgrade` als Präfix, den Namen Ihres Aurora-PostgreSQL-DB-Clusters, die Quellversion, die Zielversion sowie das Datum und den Zeitstempel, wie im folgenden Beispiel gezeigt.

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

Nach Abschluss des Upgrades können Sie den Snapshot, den Aurora erstellt und in Ihrer manuellen Snapshot-Liste erstellt und gespeichert hat, verwenden, um den DB-Cluster gegebenenfalls auf seine vorherige Version wiederherzustellen.

> **Tip**
>
> Im Allgemeinen bieten Snapshots viele Möglichkeiten, Ihren Aurora-DB-Cluster zu verschiedenen Zeitpunkten wiederherzustellen. Weitere Informationen hierzu finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot und Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit. Aurora PostgreSQL unterstützt jedoch nicht die Verwendung eines Snapshots für die Wiederherstellung auf eine frühere Nebenversion.

Während des Hauptversions-Upgrades weist Aurora ein Volume zu und klont den Quell-DB-Cluster von Aurora PostgreSQL. Wenn das Upgrade aus irgendeinem Grund fehlschlägt, verwendet Aurora PostgreSQL den Klon, um das Upgrade zurückzusetzen. Nachdem mehr als 15 Klone eines Quell-Volumes zugewiesen wurden, werden nachfolgende Klone zu vollständigen Kopien und dauern länger. Dies kann dazu führen, dass der Upgrade-Prozess ebenfalls länger dauert. Wenn Aurora PostgreSQL das Upgrade zurücksetzt, beachten Sie Folgendes:

- Möglicherweise sehen Sie Fakturierungseinträge und Metriken sowohl für das ursprüngliche Volume als auch für das geklonte Volume, das während des Upgrades zugewiesen wurde. Aurora PostgreSQL bereinigt das zusätzliche Volume, nachdem das Aufbewahrungsfenster für Cluster-Backups den Zeitpunkt des Upgrades überschritten hat.
- Die nächste regionsübergreifende Snapshot-Kopie aus diesem Cluster ist eine vollständige Kopie anstelle einer inkrementellen Kopie.

Um die DB-Instances, aus denen Ihr Cluster besteht, sicher zu aktualisieren, verwendet Aurora PostgreSQL das Dienstprogramm pg_upgrade. Nach Abschluss des Writer-Upgrades kommt es zu einem kurzen Ausfall aller Reader-Instances, während sie auf die neue Hauptversion aktualisiert werden. Weitere Informationen zu diesem PostgreSQL-Dienstprogramm finden Sie unter pg_upgrade in der PostgreSQL-Dokumentation.

Sie können Ihren Aurora-PostgreSQL-DB-Cluster auf eine neue Version upgraden, indem Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API verwenden.

Konsole

So führen Sie ein Upgrade der Engine-Version eines DB-Clusters durch

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den DB-Cluster aus, den Sie upgraden möchten.
3. Wählen Sie Ändern aus. Die Seite Modify DB cluster (DB-Cluster ändern) wird angezeigt.
4. Wählen Sie für Motorversion die neue Version.
5. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
6. Wählen Sie Apply immediately, um die Änderungen sofort anzuwenden. Die Auswahl dieser Option kann in einigen Fällen einen Ausfall verursachen. Weitere Informationen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

7. Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen. Wenn sie korrekt sind, wählen Sie Modify Cluster (Cluster ändern) aus, um Ihre Änderungen zu speichern.

   Oder klicken Sie auf Zurück, um Ihre Änderungen zu bearbeiten, oder auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

Mit dem AWS CLI-Befehl modify-db-cluster können Sie die Engine-Version eines DB-Clusters upgraden. Geben Sie die folgenden Parameter an:

- `--db-cluster-identifier` – der Name des DB-Clusters.
- `--engine-version` – die Versionsnummer der Datenbank-Engine, auf die das Upgrade durchgeführt wird Verwenden Sie den AWS CLI-Befehl describe-db-engine-versions, um weitere Informationen über gültige Engine-Versionen zu erhalten.
- `--allow-major-version-upgrade` – ein erforderliches Flag, wenn die Hauptversion des `--engine-version`-Parameters von der aktuellen Hauptversion des DB-Clusters abweicht
- `--no-apply-immediately` – Änderungen im nächsten Wartungszeitraum anwenden Verwenden Sie , um Änderungen sofort anzuwende `--apply-immediately`.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

RDS-API

Sie können die Engine-Version eines DB-Clusters upgraden, indem Sie die Operation ModifyDBCluster aufrufen. Geben Sie die folgenden Parameter an:

- `DBClusterIdentifier` – der Name des DB-Clusters, z. B *`mydbcluster`*
- `EngineVersion` – die Versionsnummer der Datenbank-Engine, auf die das Upgrade durchgeführt wird Über die Operation DescribeDBEngineVersions erhalten Sie Informationen über gültige Engine-Versionen.
- `AllowMajorVersionUpgrade` – ein erforderliches Flag, wenn die Hauptversion des `EngineVersion`-Parameters von der aktuellen Hauptversion des DB-Clusters abweicht
- `ApplyImmediately` – Änderungen sofort oder während des nächsten Wartungszeitraums anwenden Legen Sie den Wert auf fest, um Änderungen sofort anzuwende `true`. Legen Sie den Wert auf fest, um Änderungen im nächsten Wartungszeitraum durchzuführe `false`.

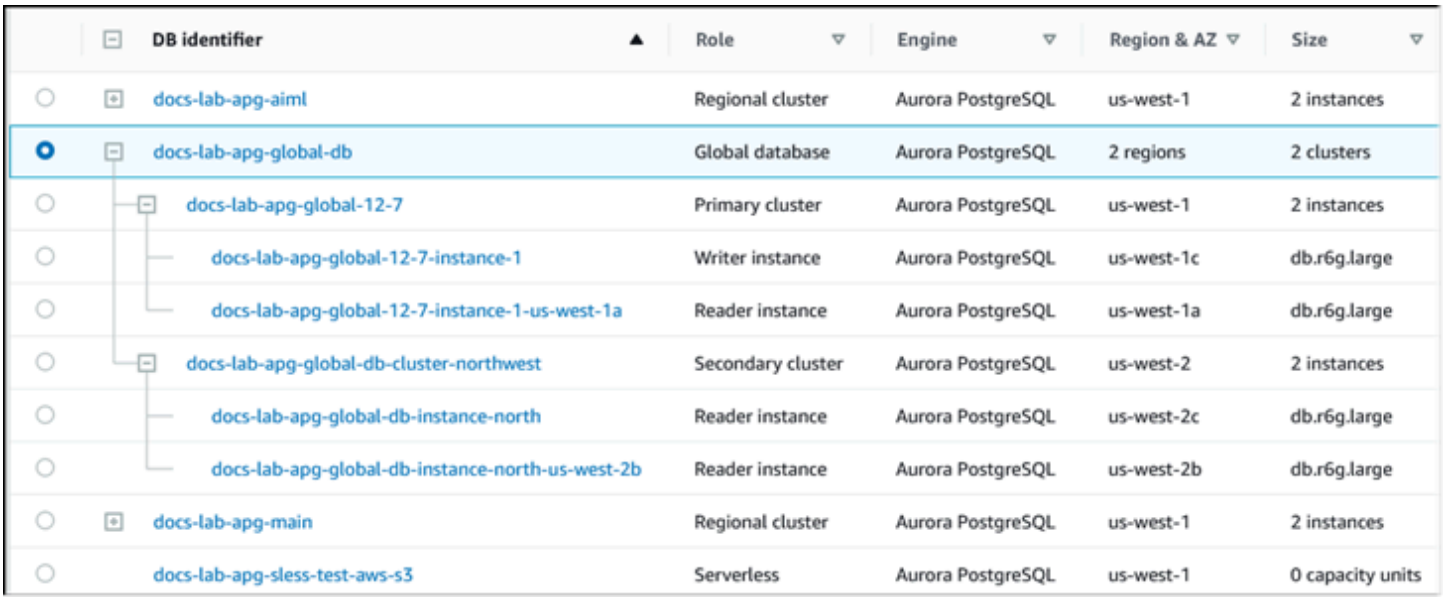
Hauptversions-Upgrades für globale Datenbanken

Bei einem globalen Aurora-Datenbank-Cluster aktualisiert der Upgrade-Prozess alle DB-Cluster, aus denen Ihre globale Aurora-Datenbank besteht, gleichzeitig. Damit wird sichergestellt, dass jeder dieselbe Aurora-PostgreSQL-Version ausführt. Außerdem wird sichergestellt, dass Änderungen an Systemtabellen, Datendateiformaten usw. automatisch auf alle sekundären Cluster repliziert werden.

Wenn Sie einen globalen Datenbank-Cluster auf eine neue Hauptversion von Aurora PostgreSQL aktualisieren möchten, empfehlen wir Ihnen, Ihre Anwendungen auf der aktualisierten Version zu testen, wie in Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion ausführlich beschrieben. Bereiten Sie Ihre DB-Cluster-Parametergruppe und die DB-Parametergruppeneinstellungen für jede AWS-Region in Ihrer globalen Aurora-Datenbank vor dem Upgrade wie in step 1. von Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion beschrieben vor.

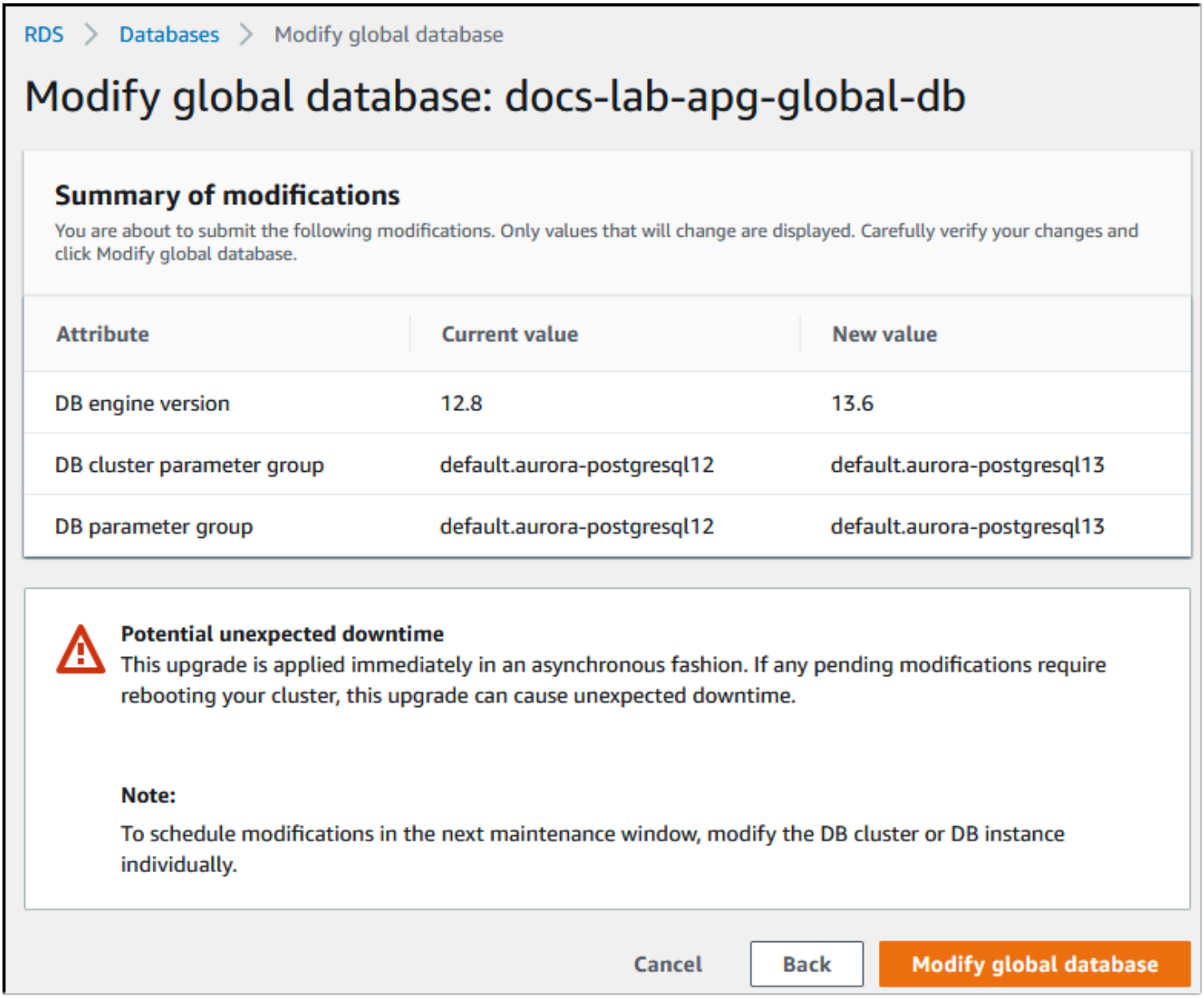
Wenn für den Aurora-PostgreSQL-Datenbank-Cluster ein Recovery Point Objective (RPO) für den Parameter `rds.global_db_rpo` festgelegt ist, müssen Sie den Parameter zurücksetzen, bevor Sie das Upgrade durchführen. Der Hauptversions-Upgrade-Prozess funktioniert nicht, wenn das RPO aktiviert ist. Dieser Parameter ist standardmäßig deaktiviert. Weitere Informationen über globale Aurora-PostgreSQL-Datenbanken und RPO finden Sie unter Verwalten von RPOs für Aurora PostgreSQL–basierte globale Datenbanken.

Nachdem Sie überprüft haben, dass Ihre Anwendungen bei der Testbereitstellung der neuen Version erwartungsgemäß ausgeführt werden, können Sie den Upgrade-Prozess starten. Lesen Sie dazu den Abschnitt Upgrade der Aurora-PostgreSQL-Engine auf eine neue Hauptversion. Wählen Sie unbedingt das Element Global database (Globale Datenbank) auf der obersten Ebene der Liste Databases (Datenbanken) in der RDS-Konsole aus, wie in der folgenden Abbildung dargestellt.



DB identifier	Role	Engine	Region & AZ	Size
docs-lab-apg-aiml	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-db	Global database	Aurora PostgreSQL	2 regions	2 clusters
docs-lab-apg-global-12-7	Primary cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-12-7-instance-1	Writer instance	Aurora PostgreSQL	us-west-1c	db.r6g.large
docs-lab-apg-global-12-7-instance-1-us-west-1a	Reader instance	Aurora PostgreSQL	us-west-1a	db.r6g.large
docs-lab-apg-global-db-cluster-northwest	Secondary cluster	Aurora PostgreSQL	us-west-2	2 instances
docs-lab-apg-global-db-instance-north	Reader instance	Aurora PostgreSQL	us-west-2c	db.r6g.large
docs-lab-apg-global-db-instance-north-us-west-2b	Reader instance	Aurora PostgreSQL	us-west-2b	db.r6g.large
docs-lab-apg-main	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-sless-test-aws-s3	Serverless	Aurora PostgreSQL	us-west-1	0 capacity units

Wie bei jeder Änderung können Sie bestätigen, dass der Prozess fortgesetzt werden soll, wenn Sie dazu aufgefordert werden.



Anstatt die Konsole zu verwenden, können Sie den Upgrade-Vorgang mit der AWS CLI oder der RDS-API starten. Wie bei der Konsole arbeiten Sie wie folgt auf dem globalen Aurora-Datenbankcluster und nicht auf einem seiner Bestandteile:

- Verwenden Sie den AWS CLI-Befehl modify-global-Cluster, um das Upgrade für Ihre globale Aurora--Datenbank mit der AWS CLI zu starten.
- Verwenden Sie die ModifyGlobalCluster-API zum Starten des Upgrades.

Durchführen eines Nebenversions-Upgrades

Sie können die folgenden Methoden verwenden, um die Nebenversion eines DB-Clusters zu aktualisieren oder einen DB-Cluster zu patchen:

Themen

- Vor dem Durchführen eines Nebenversions-Upgrades
- So führen Sie Upgrades von Nebenversionen durch und wenden Patches an
- Nebenversions-Upgrades und Zero-Downtime-Patching
- Einschränkungen des Patchens ohne Ausfallzeiten
- Durchführen eines Upgrades der Aurora-PostgreSQL-Engine auf eine neue Nebenversion

Vor dem Durchführen eines Nebenversions-Upgrades

Wir empfehlen Ihnen, die folgenden Aktionen durchzuführen, um die Ausfallzeit während eines Unterversion-Upgrades zu reduzieren:

- Die Wartung des Aurora-DB-Clusters sollte in Zeiten mit geringem Datenverkehr durchgeführt werden. Verwenden Sie Performance Insights, um diese Zeiträume zu identifizieren und die Wartungsfenster korrekt zu konfigurieren. Weitere Informationen zu Performance Insights finden Sie unter Überwachen der DB-Auslastung mit Performance Insights in Amazon RDS. Weitere Informationen zu DB-Cluster-Wartungsfenstern finden Sie unter Anpassen des bevorzugten DB-Cluster-Wartungsfensters.
- Verwenden Sie als bewährte Methode AWS SDKs, die exponentielles Backoff und Jitter unterstützen. Weitere Informationen finden Sie unter Exponentielles Backoff und Jitter.

So führen Sie Upgrades von Nebenversionen durch und wenden Patches an

Nebenversions-Upgrades und Patches werden in AWS-Regionen erst nach umfangreichen Tests zur Verfügung gestellt. Vor der Veröffentlichung von Upgrades und Patches testet Aurora PostgreSQL, um sicherzustellen, dass bekannte Sicherheitsprobleme, Fehler und andere Probleme, die nach der Veröffentlichung der Community-Nebenversion auftreten, die Stabilität der Aurora PostgreSQL-Flotte nicht beeinträchtigen.

Wenn Sie Upgrade einer Nebenversion automatisch durchführen aktivieren, führt Aurora PostgreSQL während Ihres festgelegten Wartungsfensters regelmäßig Upgrades für Ihren DB-Cluster durch.

Stellen Sie sicher, dass die Option Automatisches Nebenversions-Upgrade aktivieren für alle Instances Ihres Aurora-PostgreSQL-DB-Clusters aktiviert ist. Informationen zum Festlegen der Einstellung Automatisches Unterversion-Upgrade und zur Funktionsweise der Einstellung, wenn sie auf Cluster- und Instance-Ebene angewendet wird, finden Sie unter Automatische Nebenversions-Upgrades für Aurora-DB-Cluster.

Überprüfen Sie den Wert der Option Automatisches Nebenversions-Upgrade aktivieren für alle Ihre Aurora-PostgreSQL-DB-Cluster überprüfen, indem Sie den AWS CLI-Befehl describe-db-instances wie folgt verwenden.

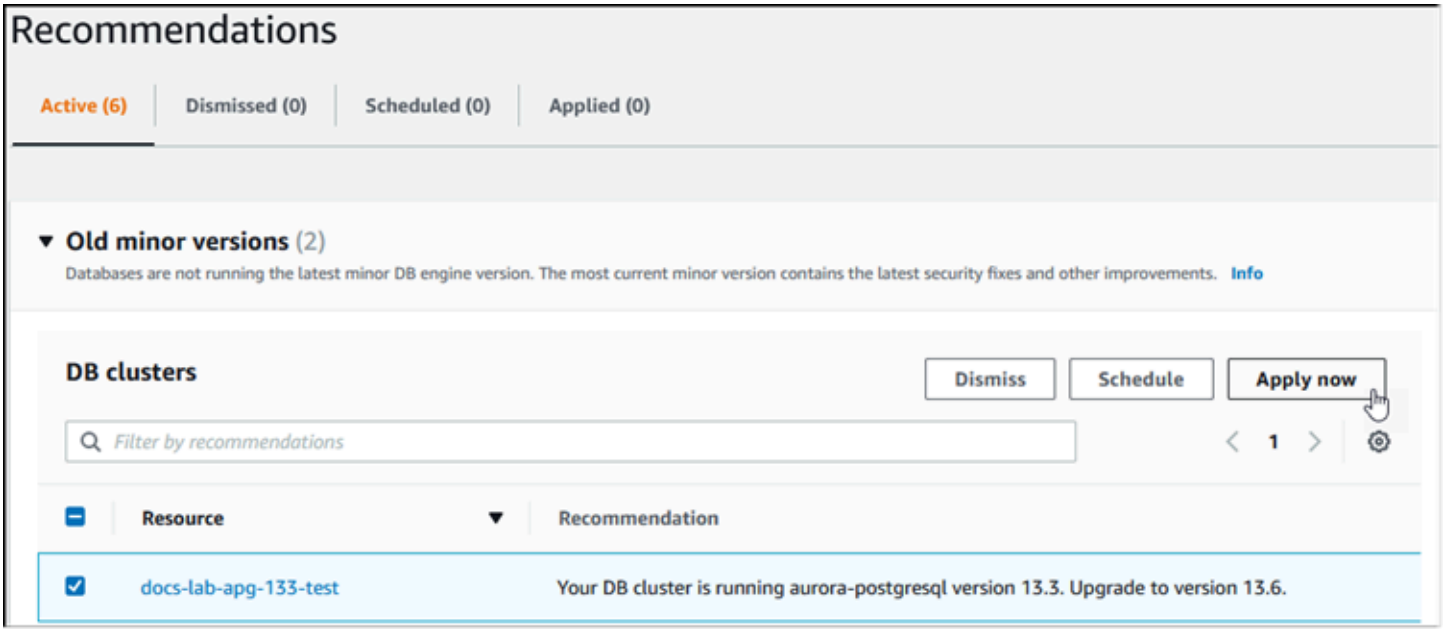
```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Diese Abfrage gibt eine Liste aller Aurora-DB-Cluster und ihrer Instances mit dem Wert `true` oder `false` für den Status der Einstellung `AutoMinorVersionUpgrade` zurück. Der Befehl geht davon aus, dass Sie Ihre AWS CLI mit Ihrer Standard-AWS-Region als Ziel konfiguriert haben.

Weitere Informationen zur AmVU-Option und darüber, wie Sie Ihren Aurora-DB-Cluster ändern, um diese u verwenden zu können, finden Sie unter Automatische Nebenversions-Upgrades für Aurora-DB-Cluster.

Sie können Ihren Aurora-PostgreSQL-DB-Cluster auf neue Nebenversionen aktualisieren, indem Sie entweder auf Wartungsaufgaben reagieren oder den Cluster so ändern, dass er die neue Version verwendet.

Sie können alle verfügbaren Upgrades oder Patches für Ihre Aurora-PostgreSQL-DB-Cluster identifizieren, indem Sie die RDS-Konsole verwenden und das Menü Recommendations (Empfehlungen) öffnen. Dort finden Sie eine Liste verschiedener Wartungsprobleme wie Old minor versions (Alte Nebenversionen) aus. Abhängig von Ihrer Produktionsumgebung können Sie mit Schedule (Planen) das Upgrade planen oder mit Apply now (Jetzt anwenden) sofort Maßnahmen ergreifen, wie im Folgenden gezeigt.



Weitere Informationen zum Verwalten eines Aurora-DB-Clusters, einschließlich der manuellen Anwendung von Patches und Nebenversions-Upgrades, finden Sie unter Warten eines Amazon Aurora-DB-Clusters.

Nebenversions-Upgrades und Zero-Downtime-Patching

Beim Upgrade eines AuroraPostgreSQL-DB-Clusters kann es zu einem Ausfall kommen. Beim Upgrade-Prozess wird die Datenbank heruntergefahren, während sie aktualisiert wird. Wenn Sie das Upgrade starten, während die Datenbank ausgelastet ist, verlieren Sie alle Verbindungen und Transaktionen, die der DB-Cluster verarbeitet. Wenn Sie warten, bis die Datenbank im Leerlauf ist, um das Upgrade durchzuführen, müssen Sie möglicherweise lange warten.

Die Funktion „Null-Downtime Patching“ (ZDP, Patchen ohne Ausfallzeiten) verbessert den Upgrade-Prozess. Mit dem ZDP können sowohl Upgrades als auch Patches mit minimalen Auswirkungen auf Ihren Aurora-PostgreSQL-DB-Cluster angewendet werden. ZDP wird verwendet, wenn Patches oder neuere Upgrades von Nebenversionen auf Aurora-PostgreSQL-Versionen und anderen höheren Versionen dieser Nebenversionen und neueren Hauptversionen angewendet werden. Das heißt, bei einem Upgrade auf neue Nebenversionen nach einer dieser Versionen wird ZDP verwendet.

Die folgende Tabelle zeigt die Aurora-PostgreSQL-Versionen und DB-Instance-Klassen, in denen ZDP verfügbar ist:

Version	db.r*-Instance-Klassen	db.t*-Instance-Klassen	db.x*-Instance-Klassen	db.serverless-Instance-Klasse
10.21 und höhere Versionen	Ja	Ja	Ja	N/A
11.16 und höhere Versionen	Ja	Ja	Ja	N/A
11.17 und höhere Versionen	Ja	Ja	Ja	N/A
12.11 und höhere Versionen	Ja	Ja	Ja	N/A
12.12 und höhere Versionen	Ja	Ja	Ja	N/A
13.7 und höhere Versionen	Ja	Ja	Ja	N/A
13.8 und höhere Versionen	Ja	Ja	Ja	Ja
14.3 und höhere Versionen	Ja	Ja	Ja	N/A
14.4 und höhere Versionen	Ja	Ja	Ja	N/A
14.5 und höhere Versionen	Ja	Ja	Ja	Ja

Version	db.r*-Instance-Klassen	db.t*-Instance-Klassen	db.x*-Instance-Klassen	db.serverless-Instance-Klasse
15.3 und höhere Versionen	Ja	Ja	Ja	Ja
16.1 und höhere Versionen	Ja	Ja	Ja	Ja

Während des Upgrade-Vorgangs mit ZDP sucht die Datenbank-Engine nach einem ruhigen Punkt, um alle neuen Transaktionen anzuhalten. Diese Aktion schützt die Datenbank bei Patches und Upgrades. Um sicherzustellen, dass Ihre Anwendungen bei unterbrochenen Transaktionen reibungslos laufen, empfehlen wir, die Logik für Wiederholversuche in Ihren Code zu integrieren. Dieser Ansatz stellt sicher, dass das System kurze Ausfallzeiten ohne Fehler bewältigen und die neuen Transaktionen nach dem Upgrade erneut versuchen kann.

Wenn ZDP erfolgreich abgeschlossen wird, werden Anwendungssitzungen, mit Ausnahme von Sitzungen mit unterbrochenen Verbindungen, beibehalten und die Datenbank-Engine wird während des laufenden Upgrades neu gestartet. Der Neustart der Datenbank-Engine kann zwar zu einem vorübergehenden Abfall des Durchsatzes führen, dieser dauert jedoch normalerweise nur wenige Sekunden oder maximal etwa 1 Minute.

In einigen Fällen ist das Zero-Downtime-Patching (ZDP) möglicherweise nicht erfolgreich. Parameteränderungen, die sich auf Ihrem DB-Cluster von Aurora PostgreSQL oder dessen Instances im Status `pending` befinden, beeinträchtigen ebenfalls ZDP.

Metriken und Ereignisse für ZDP-Operationen finden Sie auf der Seite Events (Ereignisse) in der Konsole. Zu den Ereignissen gehören der Start des ZDP-Upgrades und der Abschluss des Upgrades. In diesem Fall können Sie feststellen, wie lange der Prozess gedauert hat und wie viele Verbindungen während des Neustarts aufrecht erhalten und abgebrochen wurden. Details finden Sie in Ihrem Datenbank-Fehlerprotokoll.

Einschränkungen des Patchens ohne Ausfallzeiten

Die folgenden Einschränkungen gelten für das Patchen ohne Ausfallzeiten:

- ZDP versucht, die aktuellen Client-Verbindungen zur Aurora PostgreSQL-Writer-Instance während des Aurora-PostgreSQL-Upgrade-Prozesses beizubehalten. In den folgenden Fällen werden die Verbindungen jedoch unterbrochen, damit ZDP den Vorgang abschließen kann:

- Wenn lang dauernde Abfragen oder Transaktionen ausgeführt werden.
- Data Definition Language (DDL)-Anweisungen werden ausgeführt.
- Es werden temporäre Tabellen verwendet oder Tabellensperren sind aktiv.
- Alle Sitzungen überwachen Benachrichtigungskanäle.
- Ein Cursor mit dem Status 'WITH HOLD' wird verwendet.
- Es werden TLSv1.1-Verbindungen verwendet. Ab Aurora-PostgreSQL-Versionen höher als 16.1, 15.3, 14.8, 13.11, 12.15 und 11.20, wird ZDP mit TLSv1.3-Verbindungen unterstützt.

- ZDP wird in folgenden Fällen nicht unterstützt:
  - Wenn DB-Cluster von Aurora PostgreSQL als Aurora Serverless v1 konfiguriert sind.
  - Während des Upgrades von Aurora-Reader-Instances.
  - Während des Upgrades von Aurora-Reader-Instances, die Teil eines globalen Aurora-Datenbank-Clusters in einer sekundären Region sind.
  - Während Betriebssystem-Patches und Betriebssystem-Upgrades.

Durchführen eines Upgrades der Aurora-PostgreSQL-Engine auf eine neue Nebenversion

Sie können Ihren Aurora-PostgreSQL-DB-Cluster auf eine neue Nebenversion aktualisieren, indem Sie die Konsole, die AWS CLI oder die RDS-API verwenden. Bevor Sie das Aktualisieren durchführen, wird das Befolgen der selben bewährten Methoden empfohlen, die für Upgrades der Hauptversion empfohlen wird. Wie bei neuen Hauptversionen können auch neue Nebenversionen Optimizer-Verbesserungen aufweisen, z. B. Korrekturen, die Regressionen des Abfrageplans verursachen können. Um die Stabilität des Plans zu gewährleisten, wird die Verwendung der Erweiterung Query Plan Management (QPM) empfohlen, wie inSicherstellen der Planstabilität nach einem größeren Versions-Upgrade.

Konsole

So aktualisieren Sie die Engine-Version Ihres Aurora-PostgreSQL-DB-Clusters.

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den DB-Cluster aus, den Sie upgraden möchten.
3. Wählen Sie Ändern aus. Die Seite Modify DB cluster (DB-Cluster ändern) wird angezeigt.
4. Wählen Sie für Motorversion die neue Version.

5. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
6. Wählen Sie Apply immediately, um die Änderungen sofort anzuwenden. Die Auswahl dieser Option kann in einigen Fällen einen Ausfall verursachen. Weitere Informationen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.
7. Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen. Wenn sie korrekt sind, wählen Sie Modify Cluster (Cluster ändern) aus, um Ihre Änderungen zu speichern.

   Oder klicken Sie auf Zurück, um Ihre Änderungen zu bearbeiten, oder auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

Wenn Sie die Engine-Version eines DB-Clusters aktualisieren möchten, verwenden Sie den AWS CLI-Befehl modify-db-cluster mit den folgenden Parametern:

- `--db-cluster-identifier` – Der Name Ihres Aurora-PostgreSQL-DB-Clusters.
- `--engine-version` – die Versionsnummer der Datenbank-Engine, auf die das Upgrade durchgeführt wird Verwenden Sie den AWS CLI-Befehl describe-db-engine-versions, um weitere Informationen über gültige Engine-Versionen zu erhalten.
- `--no-apply-immediately` – Änderungen im nächsten Wartungszeitraum anwenden Verwenden Sie `--apply-immediately`, um Änderungen sofort anzuwenden.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --no-apply-immediately
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --no-apply-immediately
```

RDS-API

Sie können die Engine-Version eines DB-Clusters upgraden, indem Sie die Operation ModifyDBCluster aufrufen. Geben Sie die folgenden Parameter an:

- `DBClusterIdentifier` – der Name des DB-Clusters, z. B *`mydbcluster`*
- `EngineVersion` – die Versionsnummer der Datenbank-Engine, auf die das Upgrade durchgeführt wird Über die Operation DescribeDBEngineVersions erhalten Sie Informationen über gültige Engine-Versionen.
- `ApplyImmediately` – Änderungen sofort oder während des nächsten Wartungszeitraums anwenden Legen Sie den Wert auf fest, um Änderungen sofort anzuwende `true`. Legen Sie den Wert auf fest, um Änderungen im nächsten Wartungszeitraum durchzuführe `false`.

Aktualisieren von PostgreSQL-Erweiterungen

Wenn Sie Ihren DB-Cluster von Aurora PostgreSQL auf eine neue Haupt- oder Nebenversion aktualisieren, werden die PostgreSQL-Erweiterungen nicht gleichzeitig aktualisiert. Bei den meisten Erweiterungen aktualisieren Sie die Erweiterung, nachdem das Upgrade der Haupt- oder Nebenversion abgeschlossen ist. In einigen Fällen aktualisieren Sie die Erweiterung jedoch vor dem Upgrade der Aurora-PostgreSQL-DB-Engine. Weitere Informationen finden Sie unter list of extensions to update in Testen eines Upgrades Ihres Produktions-DB-Clusters auf eine neue Hauptversion.

Zum Installieren von PostgreSQL-Erweiterungen sind `rds_superuser`-Berechtigungen erforderlich. In der Regel delegiert ein `rds_superuser` Berechtigungen über bestimmte Erweiterungen an relevante Benutzer (Rollen), um die Verwaltung einer bestimmten Erweiterung zu erleichtern. Das heißt, dass die Aufgabe, alle Erweiterungen in Ihrem Aurora-PostgreSQL-DB-Cluster zu aktualisieren, viele verschiedene Benutzer (Rollen) betreffen kann. Beachten Sie dies, insbesondere wenn Sie den Upgrade-Prozess mithilfe von Skripten automatisieren möchten. Weitere Informationen über PostgreSQL-Berechtigungen und -Rollen finden Sie unter Sicherheit in Amazon Aurora PostgreSQL.

> **Note**
>
> Hinweise zum Aktualisieren der PostGIS-Erweiterung finden Sie unter Verwalten von Geodaten mit der PostGIS-Erweiterung (Schritt 6: Upgrade der PostGIS-Erweiterung).

Um die pg_repack-Erweiterung zu aktualisieren, entfernen Sie die Erweiterung und erstellen Sie eine neue Version in der aktualisierten DB-Instance. Weitere Informationen finden Sie unter pg_repack installation in der pg_repack-Dokumentation.

Verwenden Sie zur Aktualisierung einer Erweiterung nach einem Upgrade der Engine den Befehl ALTER EXTENSION UPDATE.

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

Um Ihre derzeit installierten Erweiterungen aufzulisten, verwenden Sie den PostgreSQL-Katalog pg_extension in dem folgenden Befehl.

```
SELECT * FROM pg_extension;
```

Um eine Liste der spezifischen Erweiterungsversionen anzuzeigen, die zur Installation verfügbar sind, verwenden Sie die PostgreSQL-Ansicht pg_available_extension_versions in dem folgenden Befehl.

```
SELECT * FROM pg_available_extension_versions;
```

Alternatives Blau/Grün-Upgradeverfahren

In einigen Situationen ist es Ihre oberste Priorität, eine sofortige Umstellung vom alten auf einen aktualisierten Cluster durchzuführen. In solchen Situationen können Sie einen mehrstufigen Prozess verwenden, der die alten und neuen Cluster side-by-side ausführt. Hier replizieren Sie Daten vom alten auf den neuen Cluster, bis der neuen Cluster zur Übernahme bereit ist. Details hierzu finden Sie unter Verwendung von Blue/Green Bereitstellungen für Datenbankaktualisierungen.

Verwenden einer Aurora PostgreSQL Long-Term Support (LTS, Langzeit-Support)-Version

Jede neue Aurora-PostgreSQL-Version bleibt für eine bestimmte Zeit zum Erstellen oder Aktualisieren eines DB-Clusters verfügbar. Nach diesem Zeitraum müssen Sie alle Cluster aktualisieren, die diese Version verwenden. Sie können Ihren Cluster vor Ablauf des Supportzeitraums manuell aktualisieren, oder Aurora kann ihn automatisch für Sie aktualisieren, wenn die Aurora-PostgreSQL-Version nicht mehr unterstützt wird.

Aurora bezeichnet bestimmte Aurora-PostgreSQL-Versionen als „Long-term Support“ (LTS, Langzeitsupport)-Versionen. Datenbank-Cluster, die LTS-Versionen verwenden, können länger dieselbe Version nutzen und weniger Upgradezyklen durchlaufen als Cluster, die Nicht-LTS-Versionen verwenden. LTS-Nebenversionen enthalten nur Bugfixes (über Patch-Versionen) für kritische Stabilitäts- und Sicherheitsprobleme. Eine LTS-Version enthält keine neuen Funktionen, die nach ihrer Einführung veröffentlicht wurden.

Einmal im Jahr werden DB-Cluster, die auf einer LTS-Nebenversion ausgeführt werden, auf die neueste Patch-Version der LTS-Version gepatcht. Wir führen diese Patches durch, um sicherzustellen, dass Sie von kumulativen Sicherheits- und Stabilitätsbehebungen profitieren. Wir können eine LTS-Nebenversion häufiger patchen, wenn kritische Bugfixes, z. B. für die Sicherheit, angewendet werden müssen.

> **Note**
>
> Um für die Dauer ihres Lebenszyklus auf einer LTS-Nebenversion zu bleiben, stellen Sie sicher, dass Sie das automatische Upgrade der Nebenversion für Ihre DB-Instances deaktivieren. Um zu vermeiden, dass Ihr DB-Cluster automatisch von der LTS-Nebenversion aktualisiert wird, deaktivieren Sie das Kontrollkästchen Automatische Aktualisierung von Unterversionen aktivieren für alle DB-Instances in Ihrem Aurora-Cluster.

Wir empfehlen, dass Sie für die meisten Ihrer Aurora-PostgreSQL-Cluster auf die neueste Version upgraden, anstatt die LTS-Version zu verwenden. Auf diese Weise wird Aurora als Managed verwalteter Service genutzt und Sie erhalten Zugriff auf die neuesten Funktionen und Bugfixes. Die LTS-Releases sind für Cluster mit folgenden Merkmalen gedacht:

- Sie können sich, abgesehen von seltenen Fällen, keine Upgrade-Ausfallzeiten Ihrer Aurora-PostgreSQL-Anwendung für kritische Patches leisten.
- Der Testzyklus für den Cluster und die zugehörigen Anwendungen dauert bei einer Aktualisierung der Aurora-PostgreSQL-Datenbank-Engine sehr lange.
- Die Datenbankversion für Ihren Aurora-PostgreSQL-Cluster enthält alle Funktionen der DB-Engine und Bugfixes, die Ihre Anwendung benötigt.

Die aktuellen LTS-Versionen für Aurora PostgreSQL lauten wie folgt:

- PostgreSQL 16.8. Es wurde am 07. April 2025 veröffentlicht. Weitere Informationen finden Sie unter PostgreSQL 16.8 in den Versionshinweisen für Aurora PostgreSQL.
- PostgreSQL 15.10. Es wurde am 27. Dezember 2024 veröffentlicht. Weitere Informationen finden Sie unter PostgreSQL 15.10 in den Versionshinweisen für Aurora PostgreSQL.
- PostgreSQL 14.6. Es wurde am 20. Januar 2023 veröffentlicht. Weitere Informationen finden Sie unter PostgreSQL 14.6 in den Versionshinweisen für Aurora PostgreSQL.
- PostgreSQL 13.9. Es wurde am 20. Januar 2023 veröffentlicht. Weitere Informationen finden Sie unter PostgreSQL 13.9 in den Versionshinweisen für Aurora PostgreSQL.
- PostgreSQL 12.9. Sie wurde am 25. Februar 2022 veröffentlicht. Weitere Informationen finden Sie unter PostgreSQL 12.9 in den Versionshinweisen für Aurora PostgreSQL.
- PostgreSQL 11.9 (Aurora PostgreSQL Version 3.4. Sie wurde am 11. Dezember 2020 veröffentlicht. Weitere Informationen zu dieser Version finden Sie unter PostgreSQL 11.9, Aurora PostgreSQL Version 3.4 in den Versionshinweisen für Aurora PostgreSQL.

Einzelheiten zu Support-Zeitplänen und Veröffentlichungszyklen für die LTS-Versionen finden Sie unter Veröffentlichungskalender für Aurora PostgreSQL.

Hinweise zum Identifizieren von Aurora- und Datenbank-Engine-Versionen finden Sie unter Identifizieren der Versionen von Amazon Aurora PostgreSQL.

Verwenden von Amazon Aurora PostgreSQL Limitless Database

Amazon Aurora PostgreSQL Limitless Database bietet automatisierte horizontale Skalierung zur Verarbeitung von Millionen von Schreibtransaktionen pro Sekunde, und verwaltet Petabyte an Daten. Gleichzeitig bleibt die Einfachheit des Betriebs in einer einzigen Datenbank erhalten. Mit Aurora PostgreSQL Limitless Database können Sie sich auf die Entwicklung umfangreicher Anwendungen konzentrieren, ohne zur Unterstützung Ihrer Workloads komplexe Lösungen für die Skalierung Ihrer Daten über mehrere DB-Instances hinweg erstellen und instandhalten zu müssen.

Themen

- Die Architektur von Aurora PostgreSQL Limitless Database
- Erste Schritte mit Aurora PostgreSQL Limitless Database
- Voraussetzungen und Überlegungen zu Aurora PostgreSQL Limitless Database
- Voraussetzungen für die Verwendung von Aurora PostgreSQL Limitless Database
- Erstellen eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet
- Arbeiten mit DB-Shard-Gruppen
- Erstellen von Tabellen in Aurora PostgreSQL Limitless Database
- Laden von Daten in Aurora PostgreSQL Limitless Database
- Abfragen einer Aurora PostgreSQL Limitless Database
- Verwalten einer Aurora PostgreSQL Limitless Database
- Überwachen einer Aurora PostgreSQL Limitless Database
- Sichern und Wiederherstellen von Aurora PostgreSQL Limitless Database
- Aktualisieren von Amazon Aurora PostgreSQL Limitless Database
- Referenz zu Aurora PostgreSQL Limitless Database

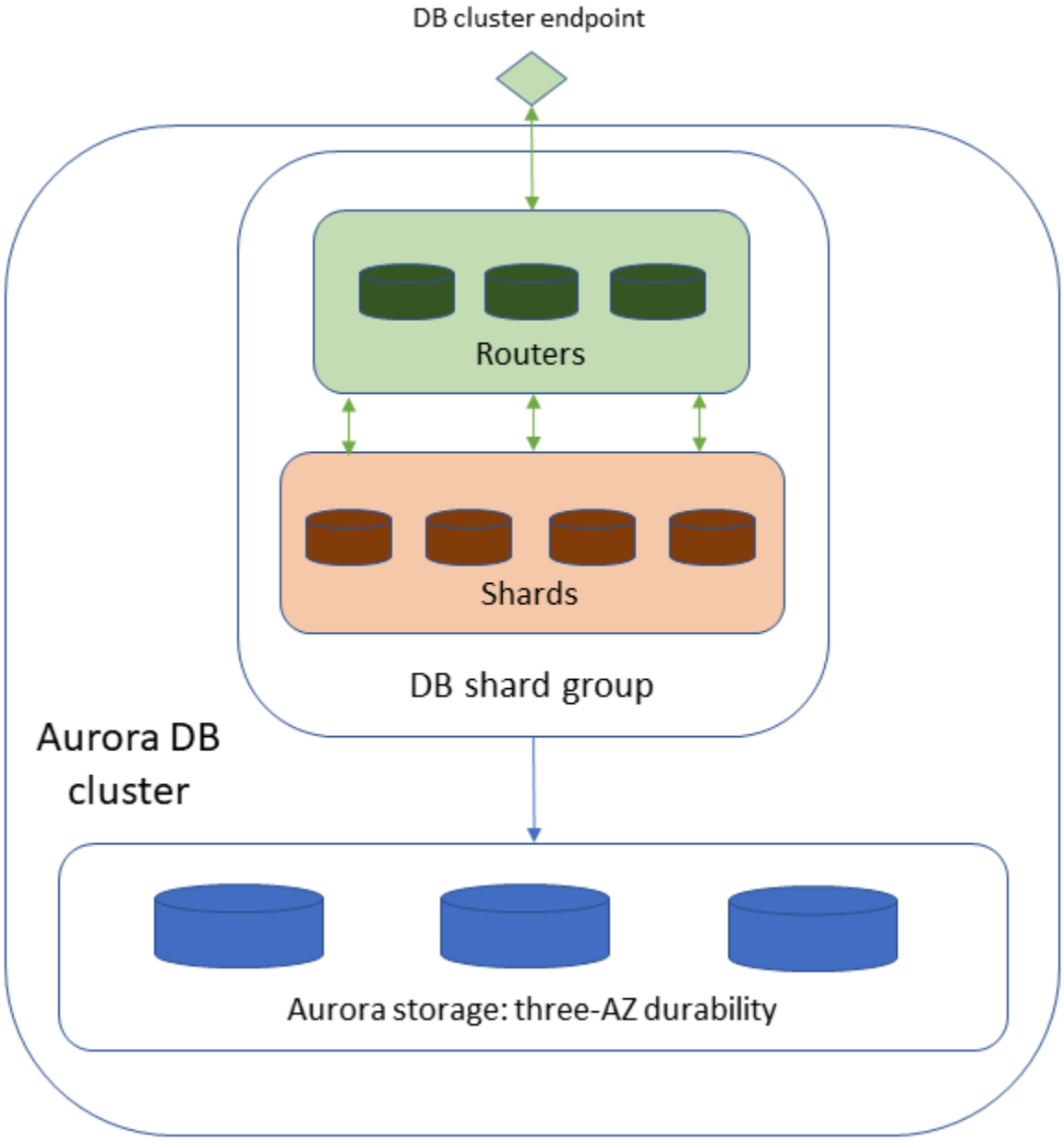
Die Architektur von Aurora PostgreSQL Limitless Database

Limitless Database verdankt ihre Skalierbarkeit einer zweischichtigen Architektur, die aus mehreren Datenbankknoten besteht. Knoten sind entweder Router oder Shards.

- Shards sind DB-Instances von Aurora PostgreSQL, die jeweils eine Teilmenge der Daten für Ihre Datenbank speichern und so eine gleichzeitige Verarbeitung ermöglichen, um einen höheren Schreibdurchsatz zu erzielen.
- Router verwalten die verteilte Struktur der Datenbank und liefern Datenbank-Clients das Bild einer zentralen Datenbank. Router verwalten Metadaten darüber, wo Daten gespeichert sind, analysieren eingehende SQL-Befehle und senden diese Befehle an Shards. Anschließend aggregieren sie Daten aus Shards, um ein einziges Ergebnis an den Client zurückzugeben, und verwalten verteilte Transaktionen, um für Einheitlichkeit in der gesamten verteilten Datenbank zu sorgen.

Aurora PostgreSQL Limitless Database unterscheidet sich von Aurora-DB-Cluster (Standard) dadurch, dass sie anstelle einer Writer-DB-Instance und von Reader-DB-Instances über eine DB-Shard-Gruppe verfügt. Alle Knoten, aus denen Ihre Limitless-Database-Architektur besteht, sind in der DB-Shard-Gruppe enthalten. Die einzelnen Shards und Router in der DB-Shard-Gruppe sind in Ihrem AWS-Konto nicht sichtbar. Sie verwenden den DB-Cluster-Endpunkt, um auf Limitless Database zuzugreifen.

Die folgende Abbildung zeigt die High-Level-Architektur von Aurora PostgreSQL Limitless Database.



Weitere Informationen zur Architektur von Aurora PostgreSQL Limitless Database und zu deren Verwendung finden Sie unter Achieving scale with Amazon Aurora PostgreSQL Limitless Database im AWS-Event-Kanal auf YouTube.

Weitere Informationen zur Architektur eines standardmäßigen Aurora-DB-Clusters finden Sie unter Amazon-Aurora-DB-Cluster.

Wichtige Begriffe zu Aurora PostgreSQL Limitless Database

DB-Shard-Gruppe

Ein Container für Limitless-Database-Knoten (Shards und Router).

Router

Ein Knoten, der SQL-Verbindungen von Clients akzeptiert, SQL-Befehle an Shards sendet, für systemweite Einheitlichkeit sorgt und Ergebnisse an Clients zurückgibt.

Shard

Ein Knoten, der eine Teilmenge von Sharded-Tabellen, vollständige Kopien von Referenztabellen und Standardtabellen speichert. Akzeptiert Abfragen von Routern, kann aber von den Clients nicht direkt kontaktiert werden.

Sharded-Tabelle

Eine Tabelle, deren Daten auf Shards partitioniert sind.

Shard-Schlüssel

Eine Spalte oder eine Gruppe von Spalten in einer Shard-Tabelle, die verwendet werden, um die Partitionierung zwischen Shards zu bestimmen.

Zusammengefasste Tabellen

Zwei Sharded-Tabellen, die denselben Shard-Schlüssel verwenden und explizit als zusammengefasst deklariert sind. Alle Daten für denselben Shard-Schlüsselwert werden an denselben Shard gesendet.

Referenztabelle

Eine Tabelle, deren Daten für jeden Shard vollständig kopiert wurden.

Standardtabelle

Der Standard-Tabellentyp in Limitless Database. Sie können Standardtabellen in Sharded- und Referenztabellen konvertieren.

Alle Standardtabellen werden auf demselben vom System ausgewählten Shard gespeichert, sodass zwischen Standardtabellen innerhalb eines einzigen Shard Verknüpfungen stattfinden können. Standardtabellen sind jedoch durch die maximale Kapazität des Shards (128 TiB) begrenzt. Dieser Shard speichert auch Daten aus Sharded- und Referenztabellen, sodass der effektive Grenzwert für Standardtabellen unter 128 TiB liegt.

Tabellentypen für Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database unterstützt drei Tabellentypen: Sharded, Referenz und Standard.

Bei Sharded-Tabellen sind die Daten auf alle Shards in der DB-Shard-Gruppe verteilt. Limitless Database veranlasst dies automatisch mithilfe eines Shard-Schlüssels, bei dem es sich um eine Spalte oder eine Gruppe von Spalten handelt, die Sie bei der Partitionierung der Tabelle festlegen. Alle Daten mit demselben Wert für den Shard-Schlüssel werden an denselben Shard gesendet. Sharding basiert auf Hash, nicht auf Bereichen oder Listen.

Geeignete Anwendungsfälle für Sharded-Tabellen:

- Die Anwendung arbeitet mit einer bestimmten Teilmenge von Daten.
- Die Tabelle ist sehr groß.
- Die Tabelle wächst möglicherweise schneller als andere Tabellen.

Sharded-Tabellen können zusammengefasst werden, was bedeutet, dass sie denselben Shard-Schlüssel verwenden und alle Daten aus beiden Tabellen mit demselben Shard-Schlüsselwert an denselben Shard gesendet werden. Wenn Sie Tabellen zusammenfassen und sie mithilfe des Shard-Schlüssels verknüpfen, kann die Verknüpfung auf einem einzigen Shard ausgeführt werden, da alle erforderlichen Daten auf diesem Shard vorhanden sind.

Für Referenztabellen gibt es eine vollständige Kopie all ihrer Daten auf jedem Shard in der DB-Shard-Gruppe. Referenztabellen werden in der Regel für kleinere Tabellen mit einem geringeren Schreibvolumen verwendet, die aber dennoch häufig verknüpft werden müssen und sich nicht für Sharding eignen. Beispiele für Referenztabellen sind Datumstabellen und Tabellen mit geographischen Daten wie Bundesland, Stadt und Postleitzahl.

Standardtabellen sind der Standardtabellentyp in Aurora PostgreSQL Limitless Database. Es handelt sich bei diesen Tabellen nicht um verteilte Tabellen. Aurora PostgreSQL Limitless Database unterstützt Verknüpfungen zwischen Standardtabellen und Standard-, Sharded- und Referenztabellen.

Fakturierung für Aurora PostgreSQL Limitless Database

Informationen zu den Gebühren für Aurora PostgreSQL Limitless Database finden Sie unter Abrechnung von DB-Instances für Aurora.

Informationen zur Preisgestaltung von Aurora finden Sie in der Aurora-Preisliste.

Erste Schritte mit Aurora PostgreSQL Limitless Database

Sie führen die folgenden Aktionen aus, um mit Aurora PostgreSQL Limitless Database zu beginnen:

1. Erstellen eines Aurora PostgreSQL-DB-Clusters und einer DB-Shard-Gruppe für Limitless Databases Weitere Informationen finden Sie unter Erstellen eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet.
2. Erstellen Sie Sharded- und Referenztabellen in der DB-Shard-Gruppe. Weitere Informationen finden Sie unter Erstellen von Tabellen in Aurora PostgreSQL Limitless Database.
3. Ändern Sie die Kapazität für Ihre DB-Shard-Gruppe, unterteilen Sie Shards und fügen Sie Router hinzu. Weitere Informationen finden Sie unter Arbeiten mit DB-Shard-Gruppen.
4. Laden Sie Daten in die DB-Shard-Gruppe. Weitere Informationen finden Sie unter Laden von Daten in Aurora PostgreSQL Limitless Database.
5. Führen Sie Abfragen und andere SQL-Anweisungen an der DB-Shard-Gruppe aus. Weitere Informationen finden Sie unter Abfragen einer Aurora PostgreSQL Limitless Database.
6. Überwachen Sie die Leistung von Limitless Database. Weitere Informationen finden Sie unter Überwachen einer Aurora PostgreSQL Limitless Database.

Voraussetzungen und Überlegungen zu Aurora PostgreSQL Limitless Database

Für Aurora PostgreSQL Limitless Database gelten die folgenden Voraussetzungen und Überlegungen.

Themen

- Voraussetzungen für Aurora PostgreSQL Limitless Database
- Überlegungen zu Aurora PostgreSQL Limitless Database
- Features, die in Aurora PostgreSQL Limitless Database nicht unterstützt werden

Voraussetzungen für Aurora PostgreSQL Limitless Database

Erfüllen Sie unbedingt diese Voraussetzungen für Aurora PostgreSQL Limitless Database.

- Aurora PostgreSQL Limitless Database ist in allen Ländern AWS-Regionen außer im asiatisch-pazifischen Raum (Taipeh) verfügbar.

  > **Note**
  >
  > Wenn Sie Ihren DB-Cluster von Aurora PostgreSQL Limitless Database in USA Ost (Nord-Virginia) erstellen, nehmen Sie die Availability Zone (AZ) `us-east-1e` nicht in Ihre DB-Subnetzgruppe auf. Aufgrund von Ressourcenbeschränkungen wird Aurora Serverless v2 (und daher Aurora PostgreSQL Limitless Database) in der AZ `us-east-1e` nicht unterstützt.

- Aurora PostgreSQL Limitless Database unterstützt nur die DB-Cluster-Speicherkonfiguration Aurora I/O-Optimized. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.
- Aurora PostgreSQL Limitless Database verwendet spezielle Versionen der DB-Engine von Aurora PostgreSQL für Aurora PostgreSQL Limitless Database:
  - `16.6.8-limitless`
  - `16.6-limitless`
  - `16.4-limitless`
- Ihr DB-Cluster darf keine Writer- oder Reader-DB-Instances haben.

- Sie müssen Erweiterte Überwachung und Performance-Insights verwenden. Die Aufbewahrungszeit von Performance-Insights muss mindestens 1 Monat (31 Tage) betragen.
- Sie müssen das PostgreSQL-Protokoll nach Amazon CloudWatch Logs exportieren.

> **Note**
>
> Für einige erforderliche Funktionen, wie z. B. Enhanced Monitoring, Performance Insights und CloudWatch Logs, fallen zusätzliche Gebühren an. Informationen zur Preisgestaltung von Aurora finden Sie in der Aurora-Preisliste.

Überlegungen zu Aurora PostgreSQL Limitless Database

Die folgenden Überlegungen gelten für DB-Shard-Gruppen in Aurora PostgreSQL Limitless Database:

- Sie können nur eine DB-Shard-Gruppe pro DB-Cluster haben.
- Sie können bis zu fünf DB-Shard-Gruppen pro Gruppe einrichten. AWS-Region

  Daher können Sie bis zu fünf Aurora PostgreSQL Limitless Database-DB-Cluster pro haben. AWS-Region Weitere Informationen finden Sie unter Kontingente in Amazon Aurora.
- Sie können die maximale Kapazität einer DB-Shard-Gruppe auf 16—6144 festlegen. ACUs Für Kapazitätsgrenzen über 6144 wenden Sie sich an. ACUs AWS

  Die anfängliche Anzahl von Routern und Shards wird durch die maximale Kapazität bestimmt, die Sie bei der Erstellung einer DB-Shard-Gruppe festgelegt haben. Weitere Informationen finden Sie unter Korrelieren der maximalen Kapazität der DB-Shard-Gruppe mit der Anzahl der erstellten Router und Shards.
- Die Anzahl der Router und Shards ändert sich nicht, wenn Sie die maximale Kapazität einer DB-Shard-Gruppe ändern.
- Stellen Sie sicher, dass das DB-Subnetz, in dem Sie die DB-Shard-Gruppe erstellen, über genügend freie IP-Adressen verfügt, um Verbindungen mit der DB-Shard-Gruppe herzustellen. Sie benötigen eine IP-Adresse für jeden Router und bis zu drei IP-Adressen für jeden Shard in der DB-Shard-Gruppe.

Weitere Informationen zur Anzahl der Router, die beim Generieren einer DB-Shard-Gruppe erstellt werden, finden Sie unter Korrelieren der maximalen Kapazität der DB-Shard-Gruppe mit der Anzahl der erstellten Router und Shards.

- Wenn Sie Ihre DB-Shard-Gruppe öffentlich zugänglich machen, richten Sie in Ihrer VPC unbedingt ein Internet-Gateway ein.
- Sie verwenden SQL-Funktionen, um Shards zu unterteilen und Router hinzuzufügen.
- Das Zusammenführen von Shards wird nicht unterstützt.
- Sie können keine einzelnen Shards und Router löschen.
- Sie können Shard-Schlüssel in keiner Weise ändern (`UPDATE`-Operationen an ihnen ausführen), auch nicht ihre Werte in Tabellenzeilen.

  Um einen Shard-Schlüssel zu ändern, löschen Sie ihn und erstellen Sie ihn anschließend neu.
- Die Isolationsstufen REPEATABLE READ, READ COMMITTED und READ UNCOMMITTED werden unterstützt. Sie können die Isolationsstufe nicht auf „serialisierbar“ festlegen.
- Manche SQL-Befehle werden nicht unterstützt. Weitere Informationen finden Sie unter Referenz zu Aurora PostgreSQL Limitless Database.
- Nicht alle PostgreSQL-Erweiterungen werden unterstützt. Weitere Informationen finden Sie unter Erweiterungen.
- Beim Erstellen einer Shard-Gruppe oder beim Hinzufügen neuer Shard-Gruppenknoten (Shards oder Router) werden diese Knoten in einer der Availability Zones (AZs) erstellt, die dem DB-Cluster zur Verfügung stehen. Sie können keine bestimmte AZ für einzelne Knoten auswählen.
- Wenn Sie eine Rechenredundanz von 2 verwenden (zwei Compute-Standbys für die DB-Shard-Gruppe), stellen Sie sicher, dass Ihre DB-Subnetzgruppe mindestens drei hat. AZs
- Aurora PostgreSQL Limitless Database unterstützt bis zu 54 Zeichen für Namen von Sharded-Tabellen.

Die folgenden Überlegungen gelten für DB-Cluster in Aurora PostgreSQL Limitless Database:

- Wir empfehlen Ihnen, AWS verwaltete Richtlinien zu verwenden, um die Berechtigungen für Ihre Datenbank und Anwendungen auf diejenigen zu beschränken, die Kunden für ihre Anwendungsfälle benötigen. Weitere Informationen finden Sie unter Bewährte Methoden für Richtlinien.
- Wenn Sie Ihren DB-Cluster von Aurora PostgreSQL Limitless Database erstellen, legen Sie nur Skalierungsparameter für die DB-Shard-Gruppe fest.

- Wenn Sie Ihren DB-Cluster löschen müssen, müssen Sie zuerst die DB-Shard-Gruppe löschen.
- Aurora PostgreSQL Limitless Database kann nicht als Replikationsquelle dienen.

Features, die in Aurora PostgreSQL Limitless Database nicht unterstützt werden

Die folgenden Features von Aurora PostgreSQL werden in Aurora PostgreSQL Limitless Database nicht unterstützt:

- Active Directory (Kerberos)-Authentifizierung
- DevOpsAmazon-Guru
- Amazon ElastiCache
- Amazon Blue/Green RDS-Bereitstellungen
- Amazon-RDS-Proxy
- Aurora Auto Scaling (automatisches Hinzufügen von Reader-Instances zum DB-Cluster)
- Globale Aurora-Datenbank
- Aurora Machine Learning
- Aurora-Empfehlungen
- Aurora Serverless v1
- Null-ETL-Integrationen von Aurora
- AWS Backup
- AWS Lambda Integration
- AWS Secrets Manager
- Babelfish für Aurora PostgreSQL
- Klonen von DB-Clustern
- Benutzerdefinierte Endpunkte
- Datenbankaktivitätsstreams
- Read Replicas
- RDS-Daten-API

Voraussetzungen für die Verwendung von Aurora PostgreSQL Limitless Database

Um Aurora PostgreSQL Limitless Database verwenden zu können, müssen Sie zunächst die folgenden Aufgaben ausführen.

Themen

- Aktivieren von DB-Shard-Gruppenoperationen

Aktivieren von DB-Shard-Gruppenoperationen

Bevor Sie eine DB-Shard-Gruppe erstellen können, müssen Sie DB-Shard-Gruppenoperationen aktivieren.

- Fügen Sie der IAM-Richtlinie der IAM-Rolle des Benutzers, der auf Aurora PostgreSQL Limitless Database zugreift, den folgenden Abschnitt hinzu:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDBShardGroup",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBShardGroup",
                "rds:DescribeDBShardGroups",
                "rds:DeleteDBShardGroup",
                "rds:ModifyDBShardGroup",
                "rds:RebootDBShardGroup"
            ],
            "Resource": [
                "arn:aws:rds:*:*:shard-group:*",
                "arn:aws:rds:*:*:cluster:*"
            ]
        }
    ]
}
```

Erstellen eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet

Sie erstellen einen neuen Aurora-DB-Cluster mit der Limitless-Database-Version von Aurora PostgreSQL und fügen dem Cluster eine DB-Shard-Gruppe hinzu. Wenn Sie eine DB-Shard-Gruppe hinzufügen, geben Sie die maximale Rechenkapazität für die gesamte DB-Shard-Gruppe (Summe der Kapazitäten für alle Router und Shards) in Aurora-Kapazitätseinheiten (ACUs) an. Jede ACU ist eine Kombination aus etwa 2 Gibibyte (GiB) Arbeitsspeicher, entsprechender CPU und Netzwerkleistung. Skalierung erhöht oder verringert die Kapazität für Ihre DB-Shard-Gruppe, abhängig von der Workload Ihrer Anwendung, ähnlich wie bei Aurora Serverless v2.

Themen

- Korrelieren der maximalen Kapazität der DB-Shard-Gruppe mit der Anzahl der erstellten Router und Shards
- Erstellen Ihres DB Clusters

Korrelieren der maximalen Kapazität der DB-Shard-Gruppe mit der Anzahl der erstellten Router und Shards

Die anfängliche Anzahl von Routern und Shards in einer DB-Shard-Gruppe wird durch die maximale Kapazität bestimmt, die Sie bei der Erstellung der DB-Shard-Gruppe festgelegt haben. Je höher die maximale Kapazität, desto mehr Router und Shards werden in der DB-Shard-Gruppe erstellt.

Jeder Knoten (Shard oder Router) hat seinen eigenen aktuellen Kapazitätswert, der ebenfalls in ACUs gemessen wird.

- Limitless Database skaliert einen Knoten auf eine höhere Kapazität, wenn ihre aktuelle Kapazität zu gering ist, um die Last zu bewältigen. Knoten werden jedoch nicht hochskaliert, wenn die Gesamtkapazität den maximalen Wert erreicht hat.
- Limitless Database skaliert den Knoten auf eine geringere Kapazität herunter, wenn dessen aktuelle Kapazität höher als erforderlich ist. Knoten werden jedoch nicht herunterskaliert, wenn die Gesamtkapazität dem Minimum entspricht.

Die folgende Tabelle zeigt die Korrelation zwischen der maximalen Kapazität der DB-Shard-Gruppe in Aurora-Kapazitätseinheiten (ACUs) und der Anzahl der erstellten Knoten (Router und Shards).

> **Note**
>
> Diese Werte können sich ändern.
> Wenn Sie die Rechenredundanz auf einen Wert ungleich Null setzen, wird die Gesamtzahl der Shards verdoppelt oder verdreifacht. Dies führt zu zusätzlichen Kosten.
> Die Knoten in Rechen-Standbys werden auf die gleiche Kapazität wie der Writer hoch- und herunterskaliert. Sie legen den Kapazitätsbereich für die Standbys nicht separat fest.

Knoten insgesamt	Router	Shards	Standard-Mindestkapazität (ACUs)	Maximaler Kapazität sbereich (ACUs)
4	2	2	16	16–400
5	2	3	20	401–500
6	2	4	24	501–600
7	3	4	28	601–700
8	3	5	32	701–800
9	3	6	36	801–900
10	4	6	40	901–1 000
11	4	7	44	1 001–1 100
12	4	8	48	1 101–1 200
13	5	8	52	1 201–1 300
14	5	9	56	1 301–1 400
15	5	10	60	1 401–1 500
16	6	10	64	1 501–1 600
17	6	11	68	1 601–1 700

Knoten insgesamt	Router	Shards	Standard-Mindestkapazität (ACUs)	Maximaler Kapazitätsbereich (ACUs)
18	6	12	72	1 701–1 800
19	7	12	76	1 801–1 900
20	7	13	80	1 901–2 000
21	7	14	84	2 001–2 100
22	8	14	88	2 101–2 200
23	8	15	92	2 201–2 300
24	8	16	96	2 301–6 144

Die auf maximaler Kapazität basierende dynamische Konfiguration für die DB-Shard-Gruppe ist nur während der Erstellung verfügbar. Die Anzahl der Router und Shards bleibt gleich, wenn die maximale Kapazität geändert wird. Weitere Informationen finden Sie unter Ändern der Kapazität einer DB-Shard-Gruppe.

Sie können SQL-Befehle verwenden, um Shards und Router zu einer DB-Shard-Gruppe hinzuzufügen. Weitere Informationen finden Sie hier:

- Unterteilen eines Shards in einer DB-Shard-Gruppe
- Hinzufügen eines Routers zu einer DB-Shard-Gruppe

> **Note**
>
> Shards oder Router können nicht gelöscht werden.

Erstellen Ihres DB Clusters

Nutzen Sie die folgenden Verfahren, um einen DB-Cluster von Aurora PostgreSQL zu erstellen, der Aurora PostgreSQL Limitless Database verwendet.

Sie können entweder die AWS-Managementkonsole oder die AWS CLI nutzen, um Ihren DB-Cluster zu erstellen, der Aurora PostgreSQL Limitless Database verwendet. Sie erstellen den primären DB-Cluster und die DB-Shard-Gruppe.

Konsole

Wenn Sie die AWS-Managementkonsole zum Erstellen des primären DB-Clusters verwenden, wird die DB-Shard-Gruppe im gleichen Verfahren ebenfalls erstellt.

So erstellen Sie den DB-Cluster mithilfe der Konsole

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Create database (Datenbank erstellen) aus.

   Die Seite Datenbank erstellen wird angezeigt.
3. Wählen Sie als Engine-Typ Aurora (PostgreSQL-kompatibel).
4. Wählen Sie für Version eine der folgenden Optionen aus:

   - Aurora PostgreSQL mit Limitless Database (kompatibel mit PostgreSQL 16.4)
   - Aurora PostgreSQL mit Limitless Database (kompatibel mit PostgreSQL 16.6)
5. Für Aurora PostgreSQL Limitless Database:

a. Geben Sie die Kennung der DB-Shard-Gruppe ein.

> **Important**
>
> Nachdem Sie die DB-Shard-Gruppe erstellt haben, können Sie die ID des DB-Clusters oder der DB-Shard-Gruppe nicht mehr ändern.

b. Für den Kapazitätsbereich der DB-Shard-Gruppe:

   i. Geben Sie die Minimale Kapazität (ACUs) ein. Verwenden Sie einen Wert von mindestens 16 ACUs.

      Für eine Entwicklungsumgebung ist der Standardwert 16 ACUs. Für eine Produktionsumgebung ist der Standardwert 24 ACUs.

   ii. Geben Sie die Maximale Kapazität (ACUs) ein. Verwenden Sie einen Wert von mindestens 16 und höchstens 6 144 ACUs.

      Für eine Entwicklungsumgebung ist der Standardwert 64 ACUs. Für eine Produktionsumgebung ist der Standardwert 384 ACUs.

   Weitere Informationen finden Sie unter Korrelieren der maximalen Kapazität der DB-Shard-Gruppe mit der Anzahl der erstellten Router und Shards.

c. Wählen Sie für Bereitstellung von DB-Shard-Gruppen aus, ob Standbys für die DB-Shard-Gruppe erstellt werden sollen:

- Keine Compute-Redundanz: Erstellt eine DB-Shard-Gruppe ohne Standbys für jeden Shard. Dies ist der Standardwert.
- Compute-Redundanz mit einem einzigen Failover-Ziel: Erstellt eine DB-Shard-Gruppe mit einem Compute-Standby in einer anderen Availability Zone (AZ).
- Compute-Redundanz mit zwei Failover-Zielen: Erstellt eine DB-Shard-Gruppe mit zwei Compute-Standbys in zwei verschiedenen AZs.

> **Note**
>
> Wenn Sie die Compute-Redundanz auf einen Wert ungleich Null setzen, wird die Gesamtzahl der Shard-DB-Instances verdoppelt oder verdreifacht. Bei diesen zusätzlichen DB-Instances handelt es sich um Compute-Standbys, die auf dieselbe Kapazität wie die Writer-Instance hoch- und herunterskaliert werden. Sie legen den Kapazitätsbereich für die Standbys nicht separat fest. Daher verdoppeln und verdreifachen sich die ACU-Auslastung und die Kosten entsprechend. Den genauen ACU-Verbrauch, der sich aus der Compute-Redundanz ergibt, finden Sie in der Metrik `DBShardGroupComputeRedundancyCapacity` unter DBShardGroup-Metriken.

d. Wählen Sie aus, ob die DB-Shard-Gruppe öffentlich zugänglich gemacht werden soll.

> **Note**
>
> Sie können diese Einstellung nicht mehr ändern, nachdem Sie die DB-Shard-Gruppe erstellt haben.

6. Für Konnektivität:

a. (Optional) Wählen Sie Verbindung zu einer EC2-Compute-Ressource herstellen und dann eine vorhandene EC2-Instance aus, oder erstellen Sie eine neue.

> **Note**
>
> Wenn Sie eine Verbindung zu einer EC2-Instance herstellen, können Sie die DB-Shard-Gruppe nicht öffentlich zugänglich machen.

b. Wählen Sie als Netzwerktyp entweder IPv4 oder den Dual-Stack-Modus aus.

c. Wählen Sie die Virtual Private Cloud (VPC) und die DB-Subnetzgruppe aus, oder verwenden Sie die Standardeinstellungen.

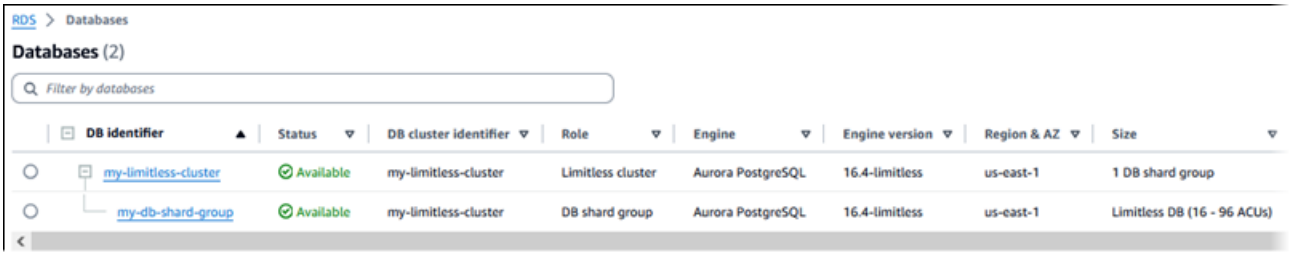
> **Note**
>
> Wenn Sie Ihren DB-Cluster von Limitless Database in USA Ost (Nord-Virginia) erstellen, nehmen Sie die Availability Zone (AZ) `us-east-1e` nicht in Ihre DB-Subnetzgruppe auf. Aufgrund von Ressourcenbeschränkungen wird Aurora Serverless v2 – und somit auch Limitless Database – in der AZ `us-east-1e` nicht unterstützt.

d. Wählen Sie die VPC-Sicherheitsgruppe (Firewall) aus, oder verwenden Sie die Standardeinstellung.

7. Wählen Sie für die Datenbankauthentifizierung entweder Passwort-Authentifizierung oder Passwort- und IAM-Datenbank-Authentifizierung aus.
8. Stellen Sie für die Überwachung sicher, dass die Kontrollkästchen Performance Insights aktivieren und Erweiterte Überwachung aktivieren aktiviert sind.

   Wählen Sie für Performance Insights eine Aufbewahrungszeit von mindestens einem Monat.
9. Erweitern Sie die letzte Zusätzliche Konfiguration auf der Seite.
10. Achten Sie darauf, dass für Protokollexporte das Kontrollkästchen PostgreSQL-Protokoll aktiviert ist.
11. Geben Sie nach Bedarf weitere Einstellungen an. Weitere Informationen finden Sie unter Einstellungen für Aurora-DB-Cluster.
12. Wählen Sie Create database (Datenbank erstellen) aus.

Nachdem der primäre DB-Cluster und die DB-Shard-Gruppe erstellt wurden, werden sie auf der Seite Datenbanken angezeigt.


RDS > Databases
Databases (2)
Filter by databases
DB identifier
Status
DB cluster identifier
Role
Engine
Engine version
Region & AZ
Size
my-limitless-cluster
Available
my-limitless-cluster
Limitless cluster
Aurora PostgreSQL
16.4-limitless
us-east-1
1 DB shard group
my-db-shard-group
Available
my-limitless-cluster
DB shard group
Aurora PostgreSQL
16.4-limitless
us-east-1
Limitless DB (16 - 96 ACUs)

CLI

Wenn Sie die AWS CLI verwenden, um einen DB-Cluster zu erstellen, der Aurora PostgreSQL Limitless Database verwendet, führen Sie die folgenden Aufgaben aus:

1. Erstellen Sie den primären DB-Cluster.
2. Erstellen Sie die DB-Shard-Gruppe.

Erstellen Sie den primären DB-Cluster

Die folgenden Parameter sind erforderlich, um den DB-Cluster zu erstellen:

- `--db-cluster-identifier`: Der Name Ihres DB-Clusters.
- `--engine`: Der DB-Cluster muss die `aurora-postgresql`-DB-Engine verwenden.
- `--engine-version`: Der DB-Cluster muss eine der DB-Engine-Versionen verwenden:
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`: Der DB-Cluster muss die Speicherkonfiguration des `aurora-iopt1`-DB-Clusters verwenden.
- `--cluster-scalability-type`: Gibt den Skalierbarkeitsmodus des Aurora-DB-Clusters an. Wenn dieser Parameter auf `limitless` gesetzt ist, arbeitet der Cluster als Aurora PostgreSQL Limitless Database. Wenn er auf `standard` (Standard) gesetzt ist, verwendet der Cluster die normale DB-Instance-Erstellung.

  > **Note**
  >
  > Sie können diese Einstellung nicht mehr ändern, nachdem Sie den DB-Cluster erstellt haben.

- `--master-username`: Der Name des Masterbenutzers für diesen DB-Cluster.
- `--master-user-password`: Das Passwort für den Masterbenutzer.
- `--enable-performance-insights`: Sie müssen Performance Insights aktivieren.
- `--performance-insights-retention-period`: Die Aufbewahrungsfrist für Performance Insights muss mindestens 31 Tage betragen.

- `--monitoring-interval`: Das Intervall in Sekunden zwischen den Punkten, an denen erweiterte Überwachungsmetriken für den DB-Cluster erfasst werden. Dieser Wert darf nicht `0` sein.
- `--monitoring-role-arn`: Der Amazon-Ressourcenname (ARN) für die IAM-Rolle, die es RDS erlaubt, erweiterte Überwachungsmetriken an Amazon CloudWatch Logs zu senden.
- `--enable-cloudwatch-logs-exports`: Sie müssen `postgresql`-Protokolle zu CloudWatch-Protokollen exportieren.

Die folgenden Parameter sind optional:

- `--db-subnet-group-name`: Die DB-Subnetzgruppe, die mit dem DB-Cluster verknüpft werden soll. Dies bestimmt auch die VPC, die mit dem DB-Cluster verknüpft wird

  > **Note**
  >
  > Wenn Sie Ihren DB-Cluster von Limitless Database in USA Ost (Nord-Virginia) erstellen, nehmen Sie die Availability Zone (AZ) `us-east-1e` nicht in Ihre DB-Subnetzgruppe auf. Aufgrund von Ressourcenbeschränkungen wird Aurora Serverless v2 – und somit auch Limitless Database – in der AZ `us-east-1e` nicht unterstützt.

- `--vpc-security-group-ids`: Eine Liste der VPC-Sicherheitsgruppen, die mit dem DB-Cluster verknüpft werden sollen.
- `--performance-insights-kms-key-id`: Die Kennung AWS KMS key für die Verschlüsselung von Performance-Insights-Daten. Wenn Sie keinen KMS-Schlüssel angeben, wird der Standardschlüssel für Ihr AWS-Konto verwendet.
- `--region`: Die AWS-Region, in der Sie den DB-Cluster erstellen. Es muss eine Region sein, die Aurora PostgreSQL Limitless Database unterstützt.

Um die Standard-VPC- und VPC-Sicherheitsgruppe zu verwenden, lassen Sie die Optionen `--db-subnet-group-name` und `--vpc-security-group-ids` weg.

So erstellen Sie den primären DB-Cluster

- ```
  aws rds create-db-cluster \
      --db-cluster-identifier my-limitless-cluster \
      --engine aurora-postgresql \
      --engine-version 16.6-limitless \
  ```
  ```

```
    --storage-type aurora-iopt1 \
    --cluster-scalability-type limitless \
    --master-username myuser \
    --master-user-password mypassword \
    --db-subnet-group-name mysubnetgroup \
    --vpc-security-group-ids sg-c7e5b0d2 \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
    --enable-cloudwatch-logs-exports postgresql
```

Weitere Informationen finden Sie unter create-db-cluster.

Erstellen Sie die DB-Shard-Gruppe

Als Nächstes erstellen Sie die DB-Shard-Gruppe in dem DB-Cluster, den Sie soeben erstellt haben. Die folgenden Parameter sind erforderlich:

- `--db-shard-group-identifier`: Der Name Ihrer DB-Shard-Gruppe.

  Die Kennung der DB-Shard-Gruppe hat die folgenden Einschränkungen:

  - Muss in dem AWS-Konto und in der AWS-Region, in der Sie sie erstellen, eindeutig sein.
  - Muss zwischen 1 und 63 Buchstaben, Zahlen oder Bindestriche enthalten.
  - Das erste Zeichen muss ein Buchstabe sein.
  - Darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.
  -

    Important

    Nachdem Sie die DB-Shard-Gruppe erstellt haben, können Sie die ID des DB-Clusters oder der DB-Shard-Gruppe nicht mehr ändern.

- `--db-cluster-identifier`: Der Name des DB-Clusters, in dem Sie die DB-Shard-Gruppe erstellen.
- `--max-acu`: Die maximale Kapazität Ihrer DB-Shard-Gruppe. Sie muss zwischen 16 und 6 144 ACUs liegen. Für Kapazitätsgrenzen von mehr als 6 144 ACUs wenden Sie sich bitte an AWS.

  Die anfängliche Anzahl von Routern und Shards wird durch die maximale Kapazität bestimmt, die Sie bei der Erstellung der DB-Shard-Gruppe festgelegt haben. Je höher die maximale Kapazität, desto mehr Router und Shards werden in der DB-Shard-Gruppe erstellt. Weitere Informationen finden Sie unter Korrelieren der maximalen Kapazität der DB-Shard-Gruppe mit der Anzahl der erstellten Router und Shards.

Die folgenden Parameter sind optional:

- `--compute-redundancy`: Ob Standbys für die DB-Shard-Gruppe erstellt werden sollen. Folgende Parameterwerte sind möglich:
  - `0`: Erstellt eine DB-Shard-Gruppe ohne Standbys für jeden Shard. Dies ist der Standardwert.
  - `1`: Erstellt eine DB-Shard-Gruppe mit einem Compute-Standby in einer anderen Availability Zone (AZ).

- 2: Erstellt eine DB-Shard-Gruppe mit zwei Compute-Standbys in zwei verschiedenen AZs.

> **Note**
>
> Wenn Sie die Rechenredundanz auf einen Wert ungleich Null setzen, wird die Gesamtzahl der Shards verdoppelt oder verdreifacht. Dies führt zu zusätzlichen Kosten.
> Die Knoten in Rechen-Standbys werden auf die gleiche Kapazität wie der Writer hoch- und herunterskaliert. Sie legen den Kapazitätsbereich für die Standbys nicht separat fest.

- `--min-acu`: Die minimale Kapazität Ihrer DB-Shard-Gruppe. Sie muss mindestens 16 ACUs umfassen, was dem Standardwert entspricht.
- `--publicly-accessible|--no-publicly-accessible`: Ob der DB-Shard-Gruppe öffentlich zugängliche IP-Adressen zugewiesen werden sollen. Der Zugriff auf die DB-Shard-Gruppe wird von den von dem Cluster verwendeten Sicherheitsgruppen kontrolliert.

  Der Standardwert ist `--no-publicly-accessible`.

> **Note**
>
> Sie können diese Einstellung nicht mehr ändern, nachdem Sie die DB-Shard-Gruppe erstellt haben.

So erstellen Sie die DB-Shard-Gruppe

- ```
  aws rds create-db-shard-group \
      --db-shard-group-identifier my-db-shard-group \
      --db-cluster-identifier my-limitless-cluster \
      --max-acu 1000
  ```
  ```

Arbeiten mit DB-Shard-Gruppen

Sie führen die folgenden Aufgaben aus, um eine DB-Shard-Gruppe für Aurora PostgreSQL Limitless Database hinzuzufügen und zu verwalten.

Themen

- Herstellen einer Verbindung zu Ihrem Aurora PostgreSQL Limitless Database-DB-Cluster
- Ermitteln der Anzahl von Routern und Shards in einer DB-Shard-Gruppe
- Beschreibung von DB-Shard-Gruppen
- Neustarten einer DB-Shard-Gruppe
- Ändern der Kapazität einer DB-Shard-Gruppe
- Unterteilen eines Shards in einer DB-Shard-Gruppe
- Hinzufügen eines Routers zu einer DB-Shard-Gruppe
- Löschen einer DB-Shard-Gruppe
- Hinzufügen einer DB-Shard-Gruppe zu einem vorhandenen DB-Cluster von Aurora PostgreSQL Limitless Database

Herstellen einer Verbindung zu Ihrem Aurora PostgreSQL Limitless Database-DB-Cluster

Um mit Aurora PostgreSQL Limitless Database zu arbeiten, stellen Sie eine Verbindung zum Writer oder Reader des Cluster-Endpunkts her. Sie können `psql` oder jedes andere Verbindungsdienstprogramm verwenden, das mit PostgreSQL funktioniert:

```
$ psql -h DB_cluster_endpoint -p port_number -U database_username -d postgres_limitless
```

Im folgenden Beispiel wird der Endpunkt für den DB-Cluster verwendet, den Sie in CLI erstellt haben.

```
$ psql -h my-limitless-cluster.cluster-ckifpdyyyxxx.us-east-1.rds.amazonaws.com -p 5432
 -U postgres -d postgres_limitless
```

> **Note**
>
> Die Standarddatenbank für die DB-Shard-Gruppe in Aurora PostgreSQL Limitless Database ist `postgres_limitless`.

Verwenden des Limitless Connection-Plug-ins

Beim Herstellen einer Verbindung mit Aurora PostgreSQL Limitless Database stellen Clients eine Verbindung über den Cluster-Endpunkt her und werden über Amazon Route 53 an einen Transaktionsrouter weitergeleitet. Route 53 ist jedoch in seiner Fähigkeit zum Lastenausgleich eingeschränkt und kann ungleichmäßige Arbeitslasten auf Transaktionsroutern ermöglichen. Das Limitless Connection-Plug-in für den AWSJDBC-Treiber behebt dieses Problem, indem es einen clientseitigen Lastenausgleich mit Lasterkennung durchführt. Weitere Informationen zum AWS JDBC-Treiber finden Sie unter Herstellen einer Verbindung mit Aurora PostgreSQL mithilfe des AWS-JDBC-Treibers.

Ermitteln der Anzahl von Routern und Shards in einer DB-Shard-Gruppe

Sie können die folgende Abfrage verwenden, um die Anzahl der Router und Shards zu ermitteln:

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
 1             | router
 2             | router
 3             | shard
 4             | shard
 5             | shard
 6             | shard
```

Beschreibung von DB-Shard-Gruppen

Verwenden Sie den AWS CLI-Befehl `describe-db-shard-groups`, um Ihre DB-Shard-Gruppen zu beschreiben. Der folgende Parameter ist optional:

- `--db-shard-group-identifier` – Der Name einer DB-Shard-Gruppe.

Im folgenden Beispiel wird eine bestimmte DB-Shard-Gruppe beschrieben.

```
aws rds describe-db-shard-groups --db-shard-group-identifier my-db-shard-group
```

Die Ausgabe sieht in etwa folgendermaßen aus:

```
{
    "DBShardGroups": [
        {
            "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
            "DBShardGroupIdentifier": "my-shard-group",
            "DBClusterIdentifier": "my-limitless-cluster",
            "MaxACU": 1000.0,
            "ComputeRedundancy": 0,
            "Status": "available",
            "PubliclyAccessible": false,
            "Endpoint": "my-limitless-cluster.limitless-ccetp2abcdef.us-
east-1.rds.amazonaws.com"
        }
```

```
    ]
}
```

Neustarten einer DB-Shard-Gruppe

Manchmal müssen Sie Ihre DB-Shard-Gruppe neu starten, beispielsweise wenn sich der Parameter `max_connections`aufgrund einer Änderung der maximalen Kapazität ändert.

Sie können AWS-Managementkonsole oder AWS CLI verwenden, um die Kapazität einer DB-Shard-Gruppe zu ändern.

Konsole

Gehen Sie wie folgt vor:

Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

1. Navigieren Sie zur Seite Datenbanken.
2. Wählen Sie die DB-Shard-Gruppe aus, die Sie neu starten möchten.
3. Wählen Sie unter Aktionen die Option Neustart aus.
4. Wählen Sie Bestätigen aus.

CLI

Verwenden Sie den AWS CLI-Befehl `reboot-db-shard-group` mit den folgenden erforderlichen Parameter, um eine DB-Parametergruppe zu kopieren:

- `--db-shard-group-identifier` – Der Name einer DB-Shard-Gruppe.

Im folgenden Beispiel wird eine DB-Shard-Gruppe neu gestartet.

```
aws rds reboot-db-shard-group --db-shard-group-identifier my-db-shard-group
```

Ändern der Kapazität einer DB-Shard-Gruppe

Sie können die AWS-Managementkonsole oder die AWS CLI verwenden, um die Kapazität einer DB-Shard-Gruppe zu ändern.

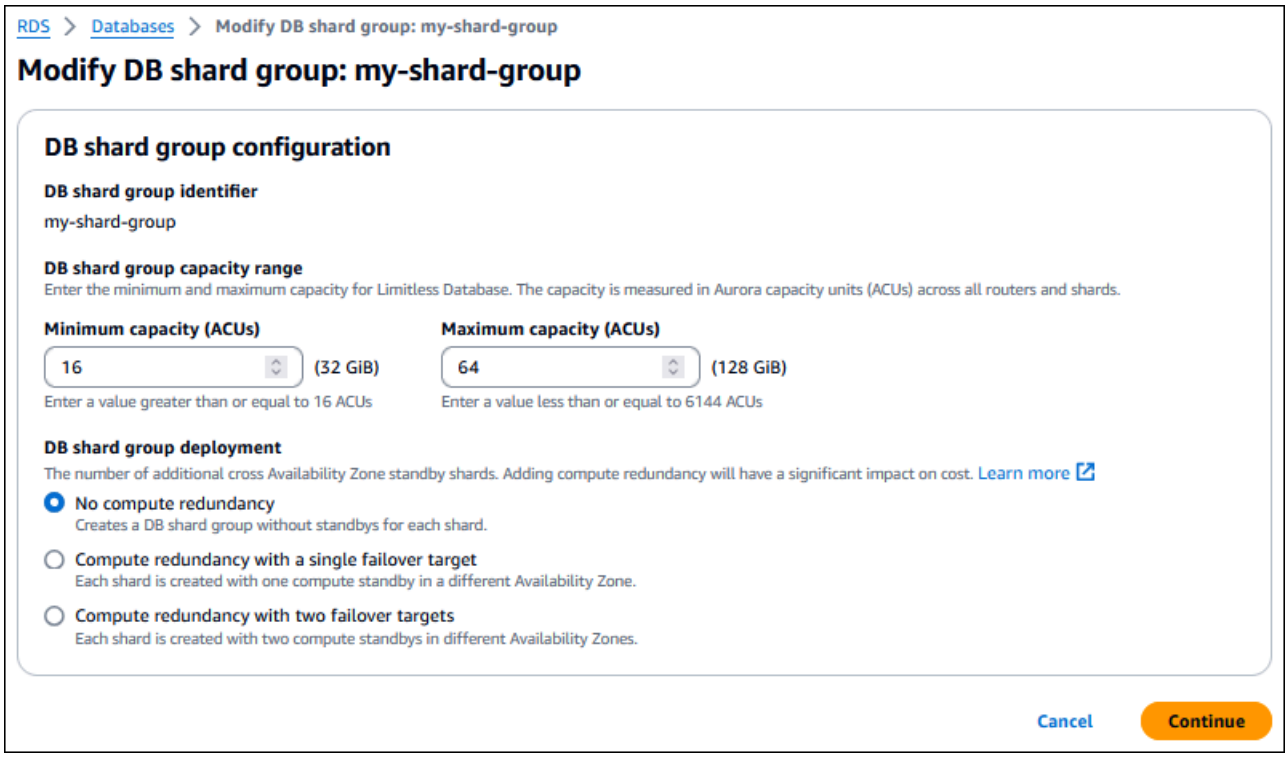
Konsole

Gehen Sie wie folgt vor:

Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

1. Navigieren Sie zur Seite Databases (Datenbanken).
2. Wählen Sie die DB-Shard-Gruppe aus, die Sie ändern möchten.
3. Wählen Sie für Actions (Aktionen) die Option Modify (Ändern) aus.

   Die Seite DB-Shard-Gruppe ändern wird angezeigt.



4. Geben Sie einen neuen Wert für Minimale Kapazität (ACUs) ein, z. B. **`100`**.
5. Geben Sie einen neuen Wert für Maximale Kapazität (ACUs) ein, z. B. **`1000`**.
6. Klicken Sie auf Weiter.

   Es wird eine Bestätigungsseite mit der Zusammenfassung Ihrer Änderungen angezeigt.

7. Überprüfen Sie Ihre Änderungen und wählen Sie dann DB-Shard-Gruppe ändern aus.

CLI

Verwenden Sie den AWS CLI-Befehl `modify-db-shard-group` mit den folgenden Parametern:

- `--db-shard-group-identifier`: Der Name der DB-Shard-Gruppe.
- `--max-acu`: Die neue maximale Kapazität der DB-Shard-Gruppe. Sie können die maximale Kapazität der DB-Shard-Gruppe auf 16 bis 6 144 ACUs festlegen. Für Kapazitätsgrenzen von mehr als 6 144 ACUs wenden Sie sich bitte an AWS.

  Die Anzahl der Router und Shards ändert sich nicht.
- `--min-acu`: Die neue minimale Kapazität Ihrer DB-Shard-Gruppe. Sie muss mindestens 16 ACUs umfassen, was dem Standardwert entspricht.

Das folgende CLI-Beispiel ändert den Kapazitätsbereich einer DB-Shard-Gruppe auf 100 bis 1 000 ACUs.

```
aws rds modify-db-shard-group \
    --db-shard-group-identifier my-db-shard-group \
    --min-acu 100 \
    --max-acu 1000
```

Unterteilen eines Shards in einer DB-Shard-Gruppe

Sie können einen Shard in einer DB-Shard-Gruppe manuell in zwei kleinere Shards unterteilen. Dies wird als ein vom Benutzer initiierter Shard-Split bezeichnet.

Aurora PostgreSQL Limitless Database kann Shards auch unterteilen, wenn sie über sehr große Datenmengen verfügen oder sehr stark genutzt werden. Dies wird als ein vom System initiierter Shard-Split bezeichnet.

Themen

- Voraussetzungen
- Unterteilen eines Shards
- Nachverfolgen von Shard-Splits
- Finalisieren von Shard-Splits
- Abbrechen eines Shard-Splits

Voraussetzungen

Für vom Benutzer initiierte Shard-Splits gelten die folgenden Voraussetzungen:

- Sie müssen über eine DB-Shard-Gruppe verfügen.
- Die DB-Shard-Gruppe darf nicht leer sein: Sie muss mindestens eine Sharded-Tabelle enthalten.
- Ein Benutzer muss die Berechtigung `rds_aurora_limitless_cluster_admin` haben. Der `rds_superuser` hat dieses Recht; daher hat es auch der Masterbenutzer. Der `rds_superuser` kann anderen Benutzern das Privileg gewähren:

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

- Sie müssen die Subcluster-ID (Knoten-ID) des Shards kennen, den Sie unterteilen möchten. Sie können die ID mithilfe der folgenden Abfrage identifizieren:

```
SELECT * FROM rds_aurora.limitless_subclusters;

 subcluster_id | subcluster_type
---------------+-----------------
```

```
1               | router
2               | router
3               | shard
4               | shard
5               | shard
6               | shard
```

Wenn Sie möchten, dass das System Shard-Splits initiiert, legen Sie die folgenden DB-Cluster-Parameter in einer benutzerdefinierten DB-Cluster-Parametergruppe fest, die Ihrem DB-Cluster zugeordnet ist:

Parameter	Wert
`rds_aurora.limitless_enable_auto_scale`	on
`rds_aurora.limitless_auto_scale_options`	Entweder `split_shard` oder `add_router,split_shard`
`rds_aurora.limitless_finalize_split_shard_mode`	Dieser Parameter bestimmt, wie vom System initiierte Shard-Splits finalisiert werden. Der Wert kann eine der folgenden Formen annehmen: • `user_initiated` : Sie entscheiden, wann der Shard-Split finalisiert werden soll. Dies ist der Standardwert. • `immediate` : Shard-Splits werden sofort finalisiert. Weitere Informationen finden Sie unter Finalisieren von Shard-Splits. **Note** Dieser Parameter gilt nur für vom System initiierte Shard-Splits.

Weitere Informationen finden Sie unter DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster.

Unterteilen eines Shards

Verwenden Sie die Funktion `rds_aurora.limitless_split_shard`, um einen Shard in einer DB-Shard-Gruppe zu unterteilen. Diese Funktion startet einen Auftrag zum Unterteilen des Shards, der asynchron ausgeführt wird.

```
SELECT rds_aurora.limitless_split_shard('subcluster_id');
```

Warten Sie nach erfolgreicher Übermittlung des Auftrags auf die Rückgabe einer Auftrags-ID, zum Beispiel:

```
SELECT rds_aurora.limitless_split_shard('3');

    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> Gleichzeitige Shard-Splits werden nicht unterstützt. Führen Sie die Aufträge nacheinander aus und schließen Sie jeden Auftrag ab, bevor Sie einen weiteren Split veranlassen.

Nachverfolgen von Shard-Splits

Sie können die Auftrags-ID verwenden, um einen Shard-Split-Auftrag nachzuverfolgen. Führen Sie die folgende Abfrage aus, um einen bestimmten Auftrag zu beschreiben und weitere Informationen dazu zu erhalten:

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(job_id);
```

Zum Beispiel:

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs(1691300000000);
```

```
    job_id     |    action   |      job_details      | status  |    submission_time
 |                 message
---------------+-------------+-----------------------+---------
+-----------------------+-------------------------------------------
 1691300000000 | SPLIT_SHARD | Split Shard 3 by User | SUCCESS | 2023-08-06 05:33:20+00
 | Scaling job succeeded.                 +
               |             |                       |         |
 | New shard instance with ID 7 was created.
(1 row)
```

Die Abfrage gibt einen Fehler zurück, wenn Sie einen nicht vorhandenen Auftrag als Eingabe übermitteln.

```
SELECT * from rds_aurora.limitless_list_shard_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

Sie können den Status aller Shard-Split-Aufträge verfolgen, indem Sie dieselbe Abfrage ohne Auftrags-ID ausführen, zum Beispiel:

```
SELECT * FROM rds_aurora.limitless_list_shard_scale_jobs();

    job_id     |   action    |  job_details          |   status    |    submission_time
     |                  message
---------------+-------------+-----------------------+-------------
+------------------------
+-------------------------------------------------------------
 1691200000000 | SPLIT_SHARD | Split Shard 3 by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
 1691300000000 | SPLIT_SHARD | Split Shard 4 by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded. +
               |             |                       |             |
     | New shard instance with ID 7 was created.
 1691400000000 | SPLIT_SHARD | Split Shard 5 by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add shard job 1691400000000.
               |             |                       |             |
     | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | SPLIT_SHARD | Split Shard 5 by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

Der Auftragsstatus kann einer der folgenden Werte sein:

- `IN_PROGRESS`: Der Shard-Split-Auftrag wurde eingereicht und ist in Bearbeitung. Sie können nur einen Auftrag nach dem anderen ausführen.
- `PENDING`: Der Shard-Split-Auftrag wartet darauf, dass Sie ihn finalisieren. Weitere Informationen finden Sie unter Finalisieren von Shard-Splits.
- `CANCELLATION_IN_PROGRESS`: Der Shard-Split-Auftrag wird vom Benutzer abgebrochen.
- `CANCELED`: Der Shard-Split-Auftrag wurde vom Benutzer oder vom System erfolgreich abgebrochen.
- `SUCCESS`: Der Shard-Split-Auftrag wurde erfolgreich abgeschlossen. Das Feld `message` enthält die Instance-ID des neuen Shards.
- `FAILED`: Der Shard-Split-Auftrag ist fehlgeschlagen. Das Feld `message` enthält die Einzelheiten zu dem Fehler und alle Maßnahmen, die zu dem fehlgeschlagenen Auftrag ergriffen werden können.

Finalisieren von Shard-Splits

Das Finalisieren ist der letzte Schritt des Shard-Split-Prozesses. Er verursacht etwas Ausfallzeit. Wenn Sie einen Shard-Split-Auftrag starten, erfolgt die Finalisierung sofort, nachdem der Auftrag erfolgreich abgeschlossen wurde.

Manchmal unterteilt das System Shards auf Grundlage ihrer Workload, wenn Sie vom System initiierte Shard-Splits mithilfe des Parameters `rds_aurora.limitless_enable_auto_scale` aktiviert haben.

In diesem Fall können Sie wählen, ob die Finalisierung sofort oder zu einem von Ihnen gewünschten Zeitpunkt erfolgt. Sie verwenden den DB-Cluster-Parameter `rds_aurora.limitless_finalize_split_shard_mode`, um zu bestimmen, wann dies geschieht:

- Wenn Sie den Wert auf `immediate` setzen, geschieht dies sofort.
- Wenn Sie den Wert auf `user_initiated` setzen, müssen Sie den Shard-Split-Auftrag manuell abschließen.

  Ein RDS-Ereignis wird an Sie gesendet, und der Status des Shard-Split-Auftrags wechselt in `PENDING`.

Bei der Einstellung `user_initiated` verwenden Sie die Funktion `rds_aurora.limitless_finalize_split_shard`, um den Shard-Split-Auftrag zu finalisieren:

```
SELECT * FROM rds_aurora.limitless_finalize_split_shard(job_id);
```

> **Note**
>
> Diese Funktion gilt nur für Shard-Splits, die vom System initiiert werden, nicht von Ihnen.

Abbrechen eines Shard-Splits

Sie können einen vom Benutzer oder vom System initiierten Shard-Split mit dem Status `IN_PROGRESS` oder `PENDING` abbrechen. Sie benötigen die Auftrags-ID, um den Auftrag abzubrechen.

```
SELECT * from rds_aurora.limitless_cancel_shard_scale_jobs(job_id);
```

Es wird keine Ausgabe zurückgegeben, es sei denn, es liegt ein Fehler vor. Sie können den Abbruch mithilfe einer Abfrage zur Auftragsverfolgung überwachen.

Hinzufügen eines Routers zu einer DB-Shard-Gruppe

Sie können einen Router zu einer DB-Shard-Gruppe hinzufügen.

Themen

- Voraussetzungen
- Hinzufügen eines Routers
- Nachverfolgen von Router-Hinzufügungen
- Abbrechen einer Router-Hinzufügung

Voraussetzungen

Folgende Voraussetzungen gelten für das Hinzufügen eines Routers:

- Sie müssen über eine DB-Shard-Gruppe verfügen.
- Ein Benutzer muss die Berechtigung `rds_aurora_limitless_cluster_admin` haben. Der `rds_superuser` hat dieses Recht; daher hat es auch der Masterbenutzer. Der `rds_superuser` kann anderen Benutzern das Privileg gewähren:

```
/* Logged in as the master user or a user with rds_superuser privileges */
CREATE USER username;
GRANT rds_aurora_limitless_cluster_admin to username;
```

> **Note**
>
> Wenn Sie das Standard-CA-Zertifikat Ihres AWS-Kontos ändern, nachdem die DB-Shard-Gruppe erstellt wurde, verwendet der neue Router das neue CA-Zertifikat, das sich vom CA-Zertifikat des vorhandenen Routers unterscheidet. Abhängig von Ihrem Trust Store können einige Verbindungen fehlschlagen.

- Wenn Sie möchten, dass das System das Hinzufügen von Routern initiiert, legen Sie die folgenden DB-Cluster-Parameter in einer benutzerdefinierten DB-Cluster-Parametergruppe fest, die Ihrem DB-Cluster zugeordnet ist:

Parameter	Wert
`rds_aurora.limitless_enable_auto_scale`	`on`
`rds_aurora.limitless_auto_scale_options`	Entweder `add_router` oder `add_router,split_shard`

Weitere Informationen finden Sie unter DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster.

Hinzufügen eines Routers

Verwenden Sie die Funktion `rds_aurora.limitless_add_router`, um einen Router hinzuzufügen. Diese Funktion startet einen Auftrag zum Hinzufügen eines Routers, der asynchron ausgeführt wird.

```
SELECT rds_aurora.limitless_add_router();
```

Warten Sie nach erfolgreicher Übermittlung des Auftrags auf die Rückgabe einer Auftrags-ID, zum Beispiel:

```
    job_id
---------------
 1691300000000
(1 row)
```

> **Note**
>
> Gleichzeitige Aufträge zum Hinzufügen von Routern werden nicht unterstützt. Führen Sie die Aufträge nacheinander aus und schließen Sie jeden Auftrag ab, bevor Sie eine weitere Hinzufügung einleiten.

Nachverfolgen von Router-Hinzufügungen

Sie können die Auftrags-ID verwenden, um einen Auftrag zum Hinzufügen eines Routers nachzuverfolgen. Führen Sie die folgende Abfrage aus, um einen bestimmten Auftrag zu beschreiben und weitere Informationen dazu zu erhalten:

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(job_id);
```

Zum Beispiel:

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs(1691300000000);

    job_id     |   action   |        job_details       | status  |    submission_time
   |                   message
---------------+------------+--------------------------+---------
+------------------------+-------------------------------------------
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                  +
               |            |                          |         |
   | New router instance with ID 7 was created.
(1 row)
```

Die Abfrage gibt einen Fehler zurück, wenn Sie einen nicht vorhandenen Auftrag als Eingabe übermitteln.

```
SELECT * from rds_aurora.limitless_list_router_scale_jobs(1691300000001);

ERROR:  no job found with the job ID provided
```

Sie können den Status aller Router-Hinzufügungen verfolgen, indem Sie dieselbe Abfrage ohne Auftrags-ID ausführen, zum Beispiel:

```
SELECT * FROM rds_aurora.limitless_list_router_scale_jobs();

    job_id     |   action   |        job_details       |   status    |
 submission_time     |                  message
---------------+------------+--------------------------+-------------
+------------------------+-------------------------------------------
 1691200000000 | ADD_ROUTER | Add 1 new Router by User | IN_PROGRESS | 2023-08-05
 01:46:40+00 |
```

```
 1691300000000 | ADD_ROUTER | Add 1 new Router by User | SUCCESS     | 2023-08-06
 05:33:20+00 | Scaling job succeeded.                +
               |            |                          |             |
      | New router instance with ID 7 was created.
 1691400000000 | ADD_ROUTER | Add 1 new Router by User | FAILED      | 2023-08-07
 09:20:00+00 | Error occurred for the add router job 1691400000000.
               |            |                          |             |
      | Retry the command. If the issue persists, contact AWS Support.
 1691500000000 | ADD_ROUTER | Add 1 new Router by User | CANCELED    | 2023-08-07
 09:20:00+00 | Scaling job was cancelled.
(4 rows)
```

Der Auftragsstatus kann einer der folgenden Werte sein:

- `IN_PROGRESS`: Der Auftrag zum Hinzufügen eines Routers wurde übermittelt und ist in Bearbeitung. Sie können nur einen Auftrag nach dem anderen ausführen.
- `CANCELLATION_IN_PROGRESS`: Der Auftrag zum Hinzufügen eines Routers wird vom Benutzer abgebrochen.
- `CANCELED`: Der Auftrag zum Hinzufügen eines Routers wurde vom Benutzer oder vom System erfolgreich abgebrochen.
- `SUCCESS`: Der Auftrag zum Hinzufügen eines Routers wurde erfolgreich abgeschlossen. Das Feld `message` enthält die Instance-ID des neuen Routers.
- `FAILED`: Der Auftrag zum Hinzufügen eines Routers ist fehlgeschlagen. Das Feld `message` enthält die Einzelheiten zu dem Fehler und alle Maßnahmen, die zu dem fehlgeschlagenen Auftrag ergriffen werden können.

> **Note**
>
> Es gibt keinen Status `PENDING`, da Router-Hinzufügungen nicht finalisiert werden müssen. Sie verursachen keine Ausfallzeiten.

Abbrechen einer Router-Hinzufügung

Sie können einen Router-Hinzufügung mit dem Status `IN_PROGRESS` abbrechen. Sie benötigen die Auftrags-ID, um den Auftrag abzubrechen.

```
SELECT * from rds_aurora.limitless_cancel_router_scale_jobs(job_id);
```

Es wird keine Ausgabe zurückgegeben, es sei denn, es liegt ein Fehler vor. Sie können den Abbruch mithilfe einer Abfrage zur Auftragsverfolgung überwachen.

Löschen einer DB-Shard-Gruppe

Sie können eine DB-Shard-Gruppe bei Bedarf löschen. Beim Löschen einer DB-Shard-Gruppe werden die Rechenknoten (Shards und Router) gelöscht, nicht jedoch der Speicher.

Konsole

Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

1. Navigieren Sie zur Seite Databases (Datenbanken).
2. Wählen Sie die DB-Shard-Gruppe aus, die Sie löschen möchten.
3. Klicken Sie bei Actions auf Delete.
4. Geben Sie **`delete me`** in das Feld ein und wählen Sie dann Löschen.

Die DB-Shard-Gruppe wird gelöscht.

AWS CLI

Zum Löschen einer DB-Shard-Gruppe verwenden Sie den AWS CLI-Befehl `delete-db-shard-group` mit folgendem Parameter:

- `--db-shard-group-identifier`: Der Name der DB-Shard-Gruppe.

Im folgenden Beispiel wird eine DB-Shard-Gruppe in dem DB-Cluster von Aurora PostgreSQL gelöscht, die Sie zuvor erstellt haben.

```
aws rds delete-db-shard-group --db-shard-group-identifier my-db-shard-group
```

Hinzufügen einer DB-Shard-Gruppe zu einem vorhandenen DB-Cluster von Aurora PostgreSQL Limitless Database

Sie können eine DB-Shard-Gruppe in einem vorhandenen DB-Cluster erstellen, wenn Sie beispielsweise einen DB-Cluster wiederherstellen oder die DB-Shard-Gruppe gelöscht hatten.

Weitere Informationen zu den Anforderungen an den primären DB-Cluster und die DB-Shard-Gruppe finden Sie unter Voraussetzungen und Überlegungen zu Aurora PostgreSQL Limitless Database.

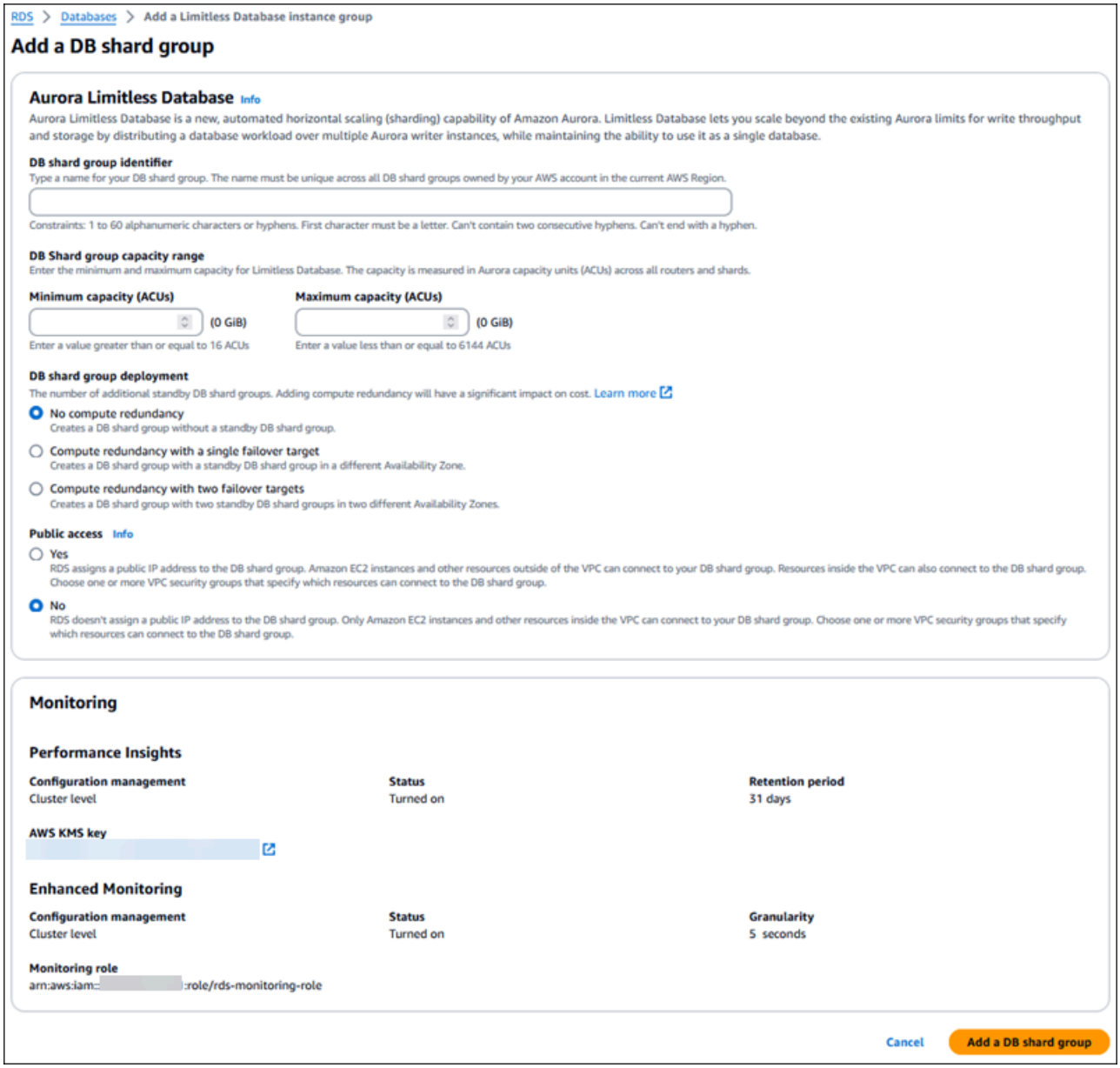
> **Note**
>
> Sie können nur eine DB-Shard-Gruppe pro Cluster haben.
> Der DB-Cluster von Limitless Database muss sich im Status `available` befinden, bevor Sie eine DB-Shard-Gruppe erstellen können.

Konsole

Sie können die AWS-Managementkonsole verwenden, um einem vorhandenen DB-Cluster eine DB-Shard-Gruppe hinzuzufügen.

So fügen Sie eine DB-Shard-Gruppe hinzu

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Navigieren Sie zur Seite Databases (Datenbanken).
3. Wählen Sie den DB-Cluster von Limitless Database aus, dem Sie eine DB-Shard-Gruppe hinzufügen möchten.
4. Wählen Sie für Aktionen die Option DB-Shard-Gruppe hinzufügen aus.



5. Geben Sie die Kennung der DB-Shard-Gruppe ein.

> **Important**
>
> Nachdem Sie die DB-Shard-Gruppe erstellt haben, können Sie die ID des DB-Clusters oder der DB-Shard-Gruppe nicht mehr ändern.

6. Geben Sie die Minimale Kapazität (ACUs) ein. Verwenden Sie einen Wert von mindestens 16 ACUs.
7. Geben Sie die Maximale Kapazität (ACUs) ein. Verwenden Sie einen Wert zwischen 16 und 6 144 ACUs.

Weitere Informationen finden Sie unter Korrelieren der maximalen Kapazität der DB-Shard-Gruppe mit der Anzahl der erstellten Router und Shards.

8. Wählen Sie für Bereitstellung von DB-Shard-Gruppen aus, ob Standbys für die DB-Shard-Gruppe erstellt werden sollen:

   - Keine Compute-Redundanz: Erstellt eine DB-Shard-Gruppe ohne Standbys für jeden Shard. Dies ist der Standardwert.
   - Compute-Redundanz mit einem einzigen Failover-Ziel: Erstellt eine DB-Shard-Gruppe mit einem Compute-Standby in einer anderen Availability Zone (AZ).
   - Compute-Redundanz mit zwei Failover-Zielen: Erstellt eine DB-Shard-Gruppe mit zwei Compute-Standbys in zwei verschiedenen AZs.

9. Wählen Sie aus, ob die DB-Shard-Gruppe öffentlich zugänglich gemacht werden soll.

   > **Note**
   >
   > Sie können diese Einstellung nicht mehr ändern, nachdem Sie die DB-Shard-Gruppe erstellt haben.

10. Wählen Sie DB-Shard-Gruppe hinzufügen aus.

AWS CLI

Verwenden Sie den AWS CLI-Befehl `create-db-shard-group`, um eine neue DB-Shard-Gruppe zu erstellen.

Die folgenden Parameter sind erforderlich:

- `--db-cluster-identifier`: Der DB-Cluster, zu dem die DB-Shard-Gruppe gehört.
- `--db-shard-group-identifier`: Der Name der DB-Shard-Gruppe.

  Die Kennung der DB-Shard-Gruppe hat die folgenden Einschränkungen:
  - Muss in dem AWS-Konto und in der AWS-Region, in der Sie sie erstellen, eindeutig sein.
  - Muss zwischen 1 und 63 Buchstaben, Zahlen oder Bindestriche enthalten.
  - Das erste Zeichen muss ein Buchstabe sein.
  - Darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.

  > **Important**
  >
  > Nachdem Sie die DB-Shard-Gruppe erstellt haben, können Sie die ID des DB-Clusters oder der DB-Shard-Gruppe nicht mehr ändern.

- `--max-acu`: Die maximale Kapazität der DB-Shard-Gruppe. Verwenden Sie einen Wert zwischen 16 und 6 144 ACUs.

Die folgenden Parameter sind optional:

- `--compute-redundancy`: Ob Standbys für die DB-Shard-Gruppe erstellt werden sollen. Folgende Parameterwerte sind möglich:
  - `0`: Erstellt eine DB-Shard-Gruppe ohne Standbys für jeden Shard. Dies ist der Standardwert.
  - `1`: Erstellt eine DB-Shard-Gruppe mit einem Compute-Standby in einer anderen Availability Zone (AZ).
  - `2`: Erstellt eine DB-Shard-Gruppe mit zwei Compute-Standbys in zwei verschiedenen AZs.

  > **Note**
  >
  > Wenn Sie die Rechenredundanz auf einen Wert ungleich Null setzen, wird die Gesamtzahl der Knoten verdoppelt oder verdreifacht. Dies führt zu zusätzlichen Kosten.

- `--min-acu`: Die minimale Kapazität Ihrer DB-Shard-Gruppe. Sie muss mindestens 16 ACUs umfassen, was dem Standardwert entspricht.
- `--publicly-accessible|--no-publicly-accessible`: Ob der DB-Shard-Gruppe öffentlich zugängliche IP-Adressen zugewiesen werden sollen. Der Zugriff auf die DB-Shard-Gruppe wird von den von dem Cluster verwendeten Sicherheitsgruppen kontrolliert.

  Der Standardwert ist `--no-publicly-accessible`.

  > **Note**
  >
  > Sie können diese Einstellung nicht mehr ändern, nachdem Sie die DB-Shard-Gruppe erstellt haben.

Im folgenden Beispiel wird eine DB-Shard-Gruppe in einem DB-Cluster von Aurora-PostgreSQL erstellt.

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-db-cluster \
    --db-shard-group-identifier my-new-shard-group \
    --max-acu 1000
```

Die Ausgabe sieht in etwa folgendermaßen aus:

```
{
    "Status": "CREATING",
    "Endpoint": "my-db-cluster.limitless-ckifpdyyyxxx.us-east-1.rds.amazonaws.com",
    "PubliclyAccessible": false,
    "DBClusterIdentifier": "my-db-cluster",
    "MaxACU": 1000.0,
    "DBShardGroupIdentifier": "my-new-shard-group",
    "DBShardGroupResourceId": "shardgroup-8986d309a93c4da1b1455add17abcdef",
    "ComputeRedundancy": 0
}
```

Erstellen von Tabellen in Aurora PostgreSQL Limitless Database

Es gibt in Aurora PostgreSQL Limitless Database drei Arten von Tabellen, die Ihre Daten enthalten:

- Standard: Sie sind der Standardtabellentyp in Aurora PostgreSQL Limitless Database. Sie erstellen Standardtabellen mit dem Befehl CREATE TABLE und können DDL-Operationen (Data Description Language) und DML-Operationen (Data Manipulation Language) an ihnen ausführen.

  Standardtabellen sind keine verteilten Tabellen. Sie werden auf einem der vom System intern ausgewählten Shards gespeichert.
- Sharded: Diese Tabellen sind auf mehrere Shards verteilt. Die Daten werden auf Grundlage der Werte der angegebenen Spalten in der Tabelle auf die Shards aufgeteilt. Dieser Satz von Spalten wird als Shard-Schlüssel bezeichnet.
- Referenz: Diese Tabellen werden auf allen Shards repliziert. Sie werden für selten geänderte Referenzdaten wie Produktkataloge und Postleitzahlen verwendet.

  Auf Shards können Join-Abfragen zwischen Referenz- und Shard-Tabellen ausgeführt werden, wodurch unnötige Datenbewegungen zwischen Shards und Routern vermieden werden.

Es gibt zwei Möglichkeiten, Limitless-Tabellen zu erstellen:

- Erstellen von Limitless-Tabellen mithilfe von Variablen: Verwenden Sie diese Methode, wenn Sie neue Sharded- und Referenz-Tabellen erstellen möchten.
- Konvertieren von Standardtabellen in Limitless-Tabellen: Verwenden Sie diese Methode, wenn Sie bestehende Standardtabellen in Sharded-und Referenztabellen konvertieren möchten.

Wir bieten auch Beispielschemata für Aurora PostgreSQL Limitless Database.

Erstellen von Limitless-Tabellen mithilfe von Variablen

Sie können mit Variablen Sharded- und Referenztabellen erstellen, indem Sie den Modus zur Tabellenerstellung festlegen. Die von Ihnen erstellten Tabellen verwenden dann diesen Modus, bis Sie einen anderen festlegen.

Verwenden Sie die folgenden Variablen, um Sharded- und Referenztabellen zu erstellen:

- `rds_aurora.limitless_create_table_mode`: Legen Sie diese Sitzungsvariable auf `sharded` oder `reference` fest. Der Standardwert dieser Variable lautet `standard`.
- `rds_aurora.limitless_create_table_shard_key`: Legen Sie für diese Sitzungsvariable ein Array von Spaltennamen fest, die als Shard-Schlüssel verwendet werden sollen. Diese Variable wird ignoriert, wenn `rds_aurora.limitless_create_table_mode` nicht `sharded` ist.

  Formatieren Sie den Wert als `untyped array literal`, ähnlich wie beim Einfügen von Literalen in eine Array-Spalte. Weitere Informationen finden Sie unter Arrays in der PostgreSQL-Dokumentation.
- `rds_aurora.limitless_create_table_collocate_with`: Legen Sie für diese Sitzungsvariable einen bestimmten Tabellennamen fest, um neu erstellte Tabellen dieser Tabelle zuzuordnen.

  Wenn zwei oder mehr Tabellen mit demselben Shard-Schlüssel „gesharded“ werden, können Sie diese Tabellen explizit einander zuordnen (zusammenfassen). Wenn zwei oder mehr Tabellen zusammengefasst werden, werden Zeilen aus diesen Tabellen mit denselben Shard-Schlüsselwerten auf demselben Shard platziert. Die Zusammenfassung hilft dabei, einige Operationen auf einen einzigen Shard zu beschränken, was zu einer besseren Leistung führt.

> **Note**
>
> Alle primären und eindeutigen Schlüssel müssen den Shard-Schlüssel enthalten. Das bedeutet, dass der Shard-Schlüssel eine Teilmenge des primären oder eindeutigen Schlüssels ist.
> Limitless-Tabellen haben einige Einschränkungen. Weitere Informationen finden Sie unter DDL-Einschränkungen und andere Informationen für Aurora PostgreSQL Limitless Database.

Themen

- Beispiele für die Verwendung von Variablen zur Erstellung von Limitless-Tabellen
- Tabellenansichten für Aurora PostgreSQL Limitless Database

Beispiele für die Verwendung von Variablen zur Erstellung von Limitless-Tabellen

In den folgenden Beispielen wird gezeigt, wie Sie diese Variablen verwenden, um Sharded- und Referenztabellen zu erstellen.

Erstellen Sie eine Sharded-Tabelle mit dem Namen `items` und dem Shard-Schlüssel `id`.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items(id int, val int, item text);
COMMIT;
```

Erstellen Sie eine Sharded-Tabelle mit dem Namen `items` und einem Shard-Schlüssel, der sich aus den Spalten `item_id` und `item_cat` zusammensetzt.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
CREATE TABLE items(item_id int, item_cat varchar, val int, item text);
COMMIT;
```

Erstellen Sie eine Sharded-Tabelle mit dem Namen `item_description` und einem Shard-Schlüssel, der sich aus den `item_id` und `item_cat` zusammensetzt, und ordnen Sie sie der Tabelle `items` aus dem vorherigen Beispiel zu.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='sharded';
SET LOCAL rds_aurora.limitless_create_table_shard_key='{"item_id", "item_cat"}';
SET LOCAL rds_aurora.limitless_create_table_collocate_with='items';
CREATE TABLE item_description(item_id int, item_cat varchar, color_id int);
COMMIT;
```

Erstellen Sie eine Referenztabelle mit dem Namen `colors`.

```
BEGIN;
SET LOCAL rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE colors(color_id int primary key, color varchar);
COMMIT;
```

Verwenden Sie die folgende Anweisung, um die Sitzungsvariable `rds_aurora.limitless_create_table_mode` auf `standard` zurückzusetzen:

```
RESET rds_aurora.limitless_create_table_mode;
```

Nachdem Sie diese Variable zurückgesetzt haben, werden Tabellen als Standardtabellen erstellt, was der Standardeinstellung entspricht. Weitere Informationen zu Standardtabellen finden Sie unter Konvertieren von Standardtabellen in Limitless-Tabellen.

Tabellenansichten für Aurora PostgreSQL Limitless Database

Informationen zu Limitless-Database-Tabellen finden Sie anhand der folgenden Ansichten.

rds_aurora.limitless_tables

Die Ansicht `rds_aurora.limitless_tables` enthält Informationen über Limitless-Tabellen und deren Typen.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | standard    | active       | standard    |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     |
 HASH (order_id)
         2 |     18579 | public      | customer    | active       | sharded     |
 HASH (cust_id)
(4 rows)
```

rds_aurora.limitless_table_collocations

Die Ansicht `rds_aurora.limitless_table_collocations` enthält Informationen über zusammengefasste Sharded-Tabellen. Zum Beispiel sind die Tabellen `orders` und `customers` zusammengefasst und haben die gleiche `collocation_id`. Die Tabellen `users` und `followers` sind zusammengefasst und haben die gleiche `collocation_id`.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;

 collocation_id | schema_name | table_name
----------------+-------------+------------
          16002 | public      | orders
          16002 | public      | customers
          16005 | public      | users
          16005 | public      | followers
(4 rows)
```

rds_aurora.limitless_table_collocation_distributions

Das Ansicht `rds_aurora.limitless_table_collocation_distributions` zeigt die Schlüsselverteilung für jede Kollokation.

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
          16002 |             6 | -9223372036854775808 | -4611686018427387904
          16002 |             5 | -4611686018427387904 |                    0
          16002 |             4 |                    0 |  4611686018427387904
          16002 |             3 |  4611686018427387904 |  9223372036854775807
          16005 |             6 | -9223372036854775808 | -4611686018427387904
          16005 |             5 | -4611686018427387904 |                    0
          16005 |             4 |                    0 |  4611686018427387904
          16005 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

Konvertieren von Standardtabellen in Limitless-Tabellen

Sie können Standardtabellen in Sharded- oder Referenztabellen konvertieren. Während der Konvertierung werden Daten aus der Standardtabelle in die verteilte Tabelle verschoben. Anschließend wird die Quell-Standardtabelle gelöscht. Daten werden mit dem Befehl `INSERT INTO SELECT FROM` verschoben.

Inhalt

- Erstellen von Sharded-Tabellen
- Erstellen von zusammengefassten Tabellen
- Erstellen von Referenztabellen

Erstellen von Sharded-Tabellen

Sie erstellen Sharded-Tabellen, indem Sie das Verfahren `rds_aurora.limitless_alter_table_type_sharded` für Standardtabellen ausführen. Dieses Verfahren verwendet eine Standardtabelle und eine Liste von Spalten und verteilt dann die angegebene Tabelle, wobei die Spaltenliste als Shard-Schlüssel verwendet wird. Das Verfahren wird synchron ausgeführt und erwirbt eine `ACCESS EXCLUSIVE`-Sperre für die Tabelle.

Nachdem das Verfahren erfolgreich abgeschlossen wurde, wird die Quell-Standardtabelle gelöscht, und eine Shard-Tabelle mit demselben Namen wird verfügbar.

Das Verfahren `rds_aurora.limitless_alter_table_type_sharded` verwendet die folgende Syntax:

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('schema.table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn']);
```

Das Verfahren erfordert die folgenden Parameter:

- `schema`: Das Datenbankschema, das die Tabelle enthält, die „gesharded“ werden soll. Wenn das Schema nicht angegeben ist, verwendet das Verfahren den `search_path`.
- `table`: Die Tabelle, die „gesharded“ werden soll.
- `shard_key`*n*: Eine Reihe von Tabellenspalten, die als Shard-Schlüssel verwendet werden sollen.

  Shard-Schlüsselwerte sind Zeichenkettenliterale und unterscheiden daher zwischen Groß- und Kleinschreibung. Wenn ein Shard-Schlüssel ein einfaches Anführungszeichen (')

enthält, verwenden Sie ein anderes einfaches Anführungszeichen, um es aufzuheben. Wenn beispielsweise eine Tabellenspalte `customer's id` genannt wird, verwenden Sie `customer''s id` als Shard-Schlüssel. Backslashes (\) und doppelte Anführungszeichen (") müssen nicht aufgehoben werden.

> **Note**
>
> Alle primären und eindeutigen Schlüssel müssen den Shard-Schlüssel enthalten. Das bedeutet, dass der Shard-Schlüssel eine Teilmenge des primären oder eindeutigen Schlüssels ist.
> In Shard-Tabellen unterstützt die Einschränkung `CHECK` keine Ausdrücke.
> Weitere Informationen finden Sie unter Beschränkungen.

So erstellen Sie eine Sharded-Tabelle

Das folgende Beispiel zeigt, wie Sie die Sharded-Tabelle `customer` mit dem Shard-Schlüssel `customer_id` erstellen.

1. Erstellen Sie die Standardtabelle.

```
CREATE TABLE customer (customer_id INT PRIMARY KEY NOT NULL, zipcode INT, email
 VARCHAR);
```

2. Konvertieren Sie die Standardtabelle in eine Sharded-Tabelle.

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.customer',
 ARRAY['customer_id']);

postgres=> \d

                  List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
```

```
(6 rows)
```

Erstellen von zusammengefassten Tabellen

Wenn zwei oder mehr Tabellen mit demselben Shard-Schlüssel „gesharded“ werden, können Sie diese Tabellen explizit einander zuordnen (zusammenfassen). Wenn zwei oder mehr Tabellen zusammengefasst werden, werden Zeilen aus diesen Tabellen mit denselben Shard-Schlüsselwerten auf demselben Shard platziert. Die Zusammenfassung hilft dabei, einige Operationen auf einen einzigen Shard zu beschränken, was zu einer besseren Leistung führt.

Sie verwenden das Verfahren `rds_aurora.limitless_alter_table_type_sharded` mit der folgenden Syntax:

```
postgres=> CALL
 rds_aurora.limitless_alter_table_type_sharded('schema.collocated_table',
 ARRAY['shard_key1', 'shard_key2', ... 'shard_keyn'], 'schema.sharded_table');
```

Das Verfahren erfordert die folgenden Parameter:

- `schema`: Das Datenbankschema, das die zusammenzufassenden Tabellen enthält. Wenn das Schema nicht angegeben ist, verwendet das Verfahren den `search_path`.
- `collocated_table`: Die Tabelle, die mit der anderen zusammengefasst werden soll.
- `shard_key`*n*: Eine Reihe von Tabellenspalten, die als Shard-Schlüssel verwendet werden sollen.

  Sie müssen denselben Shard-Schlüssel wie für die ursprüngliche Sharded-Tabelle verwenden, einschließlich derselben Spaltennamen und Spaltentypen.
- `sharded_table`: Die Sharded-Tabelle, mit der Sie die `collocated_table` zusammenfassen.

So erstellen Sie eine zusammengefasste Tabelle

1. Erstellen Sie die erste Sharding-Tabelle, indem Sie das Verfahren in Erstellen von Sharded-Tabellen befolgen.
2. Erstellen Sie die Standardtabelle für die zusammengefasste Tabelle.

   ```
   CREATE TABLE mytable2 (customer_id INT PRIMARY KEY NOT NULL, column1 INT, column2
    VARCHAR);
   ```
3. Konvertieren Sie die Standardtabelle in eine zusammengefasste Tabelle.

```
postgres=> CALL rds_aurora.limitless_alter_table_type_sharded('public.mytable2',
ARRAY['customer_id'], 'public.customer');

postgres=> \d

                   List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | mytable2     | partitioned table | postgres_limitless
 public | mytable2_fs1 | foreign table     | postgres_limitless
 public | mytable2_fs2 | foreign table     | postgres_limitless
 public | mytable2_fs3 | foreign table     | postgres_limitless
 public | mytable2_fs4 | foreign table     | postgres_limitless
 public | mytable2_fs5 | foreign table     | postgres_limitless
(12 rows)
```

Erstellen von Referenztabellen

Sie erstellen Referenztabellen, indem Sie das Verfahren `rds_aurora.limitless_alter_table_type_reference` für Standardtabellen ausführen. Dieses Verfahren repliziert eine bestimmte Tabelle auf alle Shards in der DB-Shard-Gruppe und ändert den Tabellentyp in „Referenz“. Das Verfahren wird synchron ausgeführt und erwirbt eine `ACCESS EXCLUSIVE`-Sperre für die Tabelle.

Nachdem das Verfahren erfolgreich abgeschlossen wurde, wird die Quell-Standardtabelle gelöscht, und eine Referenztabelle mit demselben Namen wird verfügbar.

Das Verfahren `rds_aurora.limitless_alter_table_type_reference` verwendet die folgende Syntax:

```
postgres=> CALL rds_aurora.limitless_alter_table_type_reference('schema.table');
```

Das gespeicherte Verfahren erfordert die folgenden Parameter:

- `schema`: Das Datenbankschema, das die Tabelle enthält, die repliziert werden soll. Wenn das Schema nicht angegeben ist, verwendet das Verfahren den `search_path`.
- `table`: Die Tabelle, die repliziert werden soll.

> **Note**
>
> Die Standardtabelle, aus der Sie die Referenztabelle erstellen, muss über einen Primärschlüssel verfügen.
> In Referenztabellen unterstützt die Einschränkung CHECK keine Ausdrücke.
> Die vorherige Funktion, `limitless_table_alter_type_reference`, ist veraltet.

So erstellen Sie eine Referenztabelle

Das folgende Beispiel zeigt, wie Sie die Referenztabelle `zipcodes` erstellen.

1. Erstellen Sie die Standardtabelle.

```
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);
```

2. Konvertieren Sie die Standardtabelle in eine Referenztabelle.

```
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcodes');

postgres=> \d

                   List of relations
 Schema |     Name     |       Type        |       Owner
--------+--------------+-------------------+--------------------
 public | customer     | partitioned table | postgres_limitless
 public | customer_fs1 | foreign table     | postgres_limitless
 public | customer_fs2 | foreign table     | postgres_limitless
 public | customer_fs3 | foreign table     | postgres_limitless
 public | customer_fs4 | foreign table     | postgres_limitless
 public | customer_fs5 | foreign table     | postgres_limitless
 public | zipcodes     | foreign table     | postgres_limitless
(7 rows)
```

Die Ausgabe zeigt die Sharded-Tabelle `customer` und die Referenztabelle `zipcodes`.

Beispielschemata für Aurora PostgreSQL Limitless Database

Wir stellen Ihnen die folgenden Beispielschemata für Aurora PostgreSQL Limitless Database zur Verfügung:

- Beispielschema für Limitless E-Commerce
- Limitless pgbench

Sie können diese Schemata verwenden, um schnell eine Beispieldatenbank zu erstellen und Daten in Tabellen von Aurora PostgreSQL Limitless Database zu laden. Weitere Informationen finden Sie im GitHub-Repository.

Laden von Daten in Aurora PostgreSQL Limitless Database

Sie können Daten mithilfe des `COPY`-Befehls oder des Dienstprogramms zum Laden von Daten in Tabellen von Aurora PostgreSQL Limitless Database laden.

> **Note**
>
> Sie können Daten in Standard-, Sharded- und Referenztabellen laden.

Inhalt

- Verwenden des COPY-Befehls in Aurora PostgreSQL Limitless Database
  - Verwenden des COPY-Befehls, um Daten in Aurora PostgreSQL Limitless Database zu laden
    - Unterteilen von Daten in mehrere Dateien
  - Kopieren von Limitless-Database-Daten in eine Datei mit dem COPY-Befehl
- Verwenden des Dienstprogramms zum Laden von Daten für Aurora PostgreSQL Limitless Database
  - Einschränkungen
  - Voraussetzungen
  - Vorbereiten der Quelldatenbank
  - Vorbereiten der Zieldatenbank
  - Erstellen von Datenbank-Anmeldeinformationen
    - Erstellen Sie die Quelldatenbank-Anmeldeinformationen
    - Erstellen Sie die Anmeldeinformationen für die Zieldatenbank
  - Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs mithilfe eines Skripts
    - Setup-Skript für das Datenladeprogramm
    - Ausgabe aus dem Setup-Skript des Dienstprogramms zum Laden von Daten
    - Bereinigen von fehlerhaften Ressourcen
  - Manuelles Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs
    - Erstellung des vom Kunden verwalteten AWS KMS key
    - Erstellen der Datenbank-Secrets
    - Erstellen der IAM-Rolle
    - Aktualisierung der vom Kunden verwalteten AWS KMS key

    - Hinzufügen der Berechtigungsrichtlinien für die IAM-Rolle
- Laden von Daten aus einem DB-Cluster von Aurora PostgreSQL oder einer DB-Instance von RDS für Postgre
- Überwachen des Ladens von Daten
    - Auflisten von Aufträgen zum Laden von Daten
    - Anzeigen der Details von Aufträgen zum Laden von Daten mithilfe der Auftrags-ID
    - Überwachen der Amazon-CloudWatch-Protokollgruppe
    - Überwachen von RDS-Ereignissen
- Abbrechen eines Auftrags zum Laden von Daten

Verwenden des COPY-Befehls in Aurora PostgreSQL Limitless Database

Sie können die Funktion\copy im Dienstprogramm `psql` verwenden, um Daten in und aus Aurora PostgreSQL Limitless Database zu importieren und zu exportieren.

Verwenden des COPY-Befehls, um Daten in Aurora PostgreSQL Limitless Database zu laden

Aurora PostgreSQL Limitless Database ist mit der \copy-Funktionalität im Dienstprogramm `psql` zum Importieren von Daten kompatibel.

In Limitless Database wird wie in Aurora PostgreSQL Folgendes nicht unterstützt:

- Direkter SSH-Zugriff auf DB-Instances: Sie können eine Datendatei (z. B. im CSV-Format) nicht auf den DB-Instance-Host kopieren und `COPY` von der Datei aus ausführen.
- Lokale Dateien auf der DB-Instance verwenden: Verwenden Sie `COPY ... FROM STDIN` und `COPY ... TO STDOUT`.

Der `COPY`-Befehl in PostgreSQL bietet Optionen für die Arbeit mit lokalen Dateien (`FROM/TO`) und die Übertragung von Daten über eine Verbindung zwischen dem Client und dem Server (`STDIN/STDOUT`). Weitere Informationen finden Sie unter COPY in der PostgreSQL-Dokumentation.

Der `\copy`-Befehl im PostgreSQL-Dienstprogramm `psql` funktioniert mit lokalen Dateien auf dem Computer, auf dem Sie den `psql`-Client ausführen. Er ruft den entsprechenden `COPY ... FROM STDIN`- oder `COPY ... FROM STDOUT`-Befehl auf dem Remote-Server (z. B. Limitless Database) auf, zu dem Sie eine Verbindung herstellen. Er liest Daten aus der lokalen Datei in `STDIN` ein oder schreibt an sie aus `STDOUT`.

Unterteilen von Daten in mehrere Dateien

Daten werden auf mehreren Shards in Aurora PostgreSQL Limitless Database gespeichert. Um das Laden von Daten mit `\copy` zu beschleunigen, können Sie Ihre Daten in mehrere Dateien unterteilen. Importieren Sie Daten dann unabhängig für jede Datendatei, indem Sie separate `\copy`-Befehle parallel ausführen.

Sie haben beispielsweise eine Eingabedatendatei im CSV-Format mit 3 Millionen zu importierenden Zeilen. Sie können die Datei in Blöcke aufteilen, die jeweils 200 000 Zeilen (15 Blöcke) enthalten:

```
split -l200000 data.csv data_ --additional-suffix=.csv -d
```

Dies führt zu Dateien von `data_00.csv` bis `data_14.csv`. Anschließend können Sie Daten mit 15 parallelen `\copy`-Befehlen importieren, zum Beispiel:

```
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table from '/tmp/data_00.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_01.csv';" postgres_limitless &
...
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_13.csv';" postgres_limitless &
psql -h dbcluster.limitless-111122223333.aws-region.rds.amazonaws.com -U username -c
 "\copy test_table FROM '/tmp/data_14.csv';" postgres_limitless
```

Mit dieser Technik wird dieselbe Datenmenge etwa zehnmal schneller importiert als mit einem einzigen `\copy`-Befehl.

Kopieren von Limitless-Database-Daten in eine Datei mit dem COPY-Befehl

Sie können den Befehl \copy verwenden, um Daten aus einer Limitless-Tabelle in eine Datei zu kopieren, wie im folgenden Beispiel gezeigt:

```
postgres_limitless=> \copy test_table TO '/tmp/test_table.csv' DELIMITER ',' CSV
 HEADER;
```

Verwenden des Dienstprogramms zum Laden von Daten für Aurora PostgreSQL Limitless Database

Aurora bietet ein Dienstprogramm, um Daten aus einem DB-Cluster von Aurora-PostgreSQL oder einer DB-Instance von RDS für PostgreSQL direkt in Limitless Database zu laden.

Sie führen die folgenden Schritte aus, um das Dienstprogramm zum Laden von Daten zu verwenden:

1. Voraussetzungen
2. Vorbereiten der Quelldatenbank
3. Vorbereiten der Zieldatenbank
4. Erstellen von Datenbank-Anmeldeinformationen
5. Eine der beiden folgenden Komponenten:
   - Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs mithilfe eines Skripts (empfohlen)
   - Manuelles Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs
6. Laden von Daten aus einem DB-Cluster von Aurora PostgreSQL oder einer DB-Instance von RDS für Postgre

Einschränkungen

Das Dienstprogramm zum Laden von Daten hat die folgenden Einschränkungen:

- Die folgenden Datentypen werden nicht unterstützt: `enum`, `ARRAY`, `BOX`, `CIRCLE`, `LINE`, `LSEG`, `PATH`, `PG_LSN`, `PG_SNAPSHOT`, `POLYGON`, `TSQUERY`, `TSVECTOR` und `TXID_SNAPSHOT`.
- An erster Stelle stehende Nullen (`0`) werden beim Laden aus dem `VARBIT`-Datentyp entfernt.
- Die Datenmigration schlägt fehl, wenn die Zieltabellen Fremdschlüssel enthalten.
- Das Laden von Daten aus Multi-AZ-DB-Cluster von RDS für PostgreSQL wird nicht unterstützt.

Voraussetzungen

Für das Dienstprogramm zum Laden von Daten gelten die folgenden Voraussetzungen:

- Die Quelldatenbank verwendet Aurora PostgreSQL oder RDS für PostgreSQL Version 11.x und höher.

- Die Quelldatenbank befindet sich im selben AWS-Konto und in derselben AWS-Region wie die Shard-Gruppe der Ziel-DB.
- Der Quell-DB-Cluster oder die Quell-DB-Instance befindet sich im Status `available`.
- Tabellen in der Quelldatenbank und in der Limitless-Datenbank haben dieselben Tabellennamen, Spaltennamen und Spaltendatentypen.
- Die Quell- und Zieltabellen haben Primärschlüssel, die dieselben Spalten und Spaltenreihenfolgen verwenden.
- Sie benötigen eine Umgebung, in der Sie eine Verbindung zu einer Limitless-Datenbank herstellen können, um Befehle zum Laden von Daten ausführen zu können. Folgende Befehle sind verfügbar:
  - `rds_aurora.limitless_data_load_start`
  - `rds_aurora.limitless_data_load_cancel`
- Für CDC:
  - Die Quelldatenbank und die Shard-Gruppe der Ziel-DB müssen dieselbe DB-Subnetzgruppe, VPC-Sicherheitsgruppe und denselben Datenbankport verwenden. Diese Setups sind für Netzwerkverbindungen zur Quelldatenbank und zu den Routern in der DB-Shard-Gruppe vorgesehen.
  - Sie müssen in der Quelldatenbank logische Replikation aktivieren. Der Benutzer der Quelldatenbank muss über Leserechte für die logische Replikation verfügen.

Vorbereiten der Quelldatenbank

Um auf die Quelldatenbank für das Laden von Daten zugreifen zu können, müssen Sie eingehenden Netzwerkverkehr zulassen. Führen Sie die folgenden Schritte aus.

So lassen Sie Netzwerkverkehr zur Quelldatenbank zu

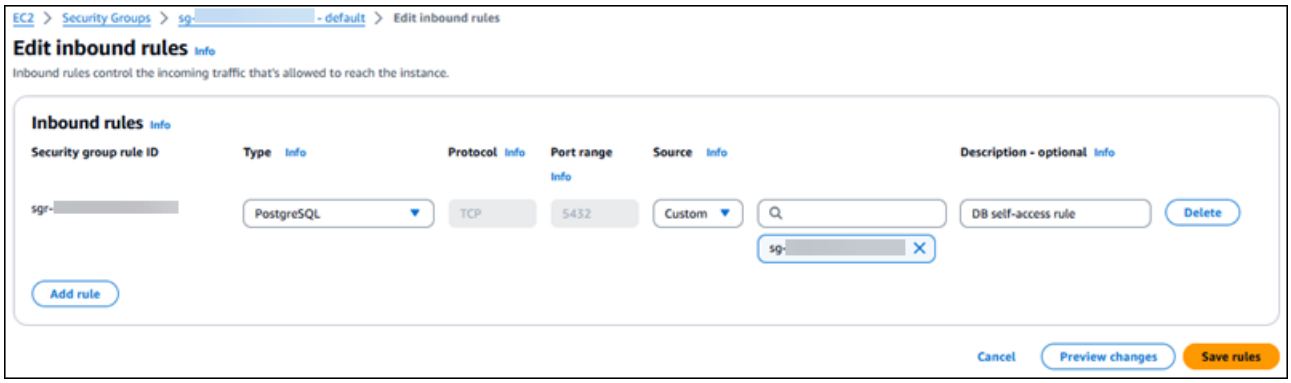
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/EC2/.
2. Navigieren Sie zur Seite Sicherheitsgruppen.
3. Wählen Sie die Sicherheitsgruppen-ID für die Sicherheitsgruppe, die vom Quell-DB-Cluster oder der Quell-DB-Instance verwendet wird.

   Die Sicherheitsgruppen-ID lautet beispielsweise `sg-056a84f1712b77926`.
4. Gehen Sie auf der Registerkarte Regeln für eingehenden Datenverkehr wie folgt vor:

a. Wählen Sie Edit inbound rules (Regeln für eingehenden Datenverkehr bearbeiten) aus.

b. Fügen Sie eine neue Regel für eingehenden Datenverkehr für den Quell-DB-Cluster oder die Quell-DB-Instance hinzu:

- Portbereich: Datenbankport für die Quelldatenbank, in der Regel `5432`
- Sicherheitsgruppen-ID: `sg-056a84f1712b77926` in diesem Beispiel

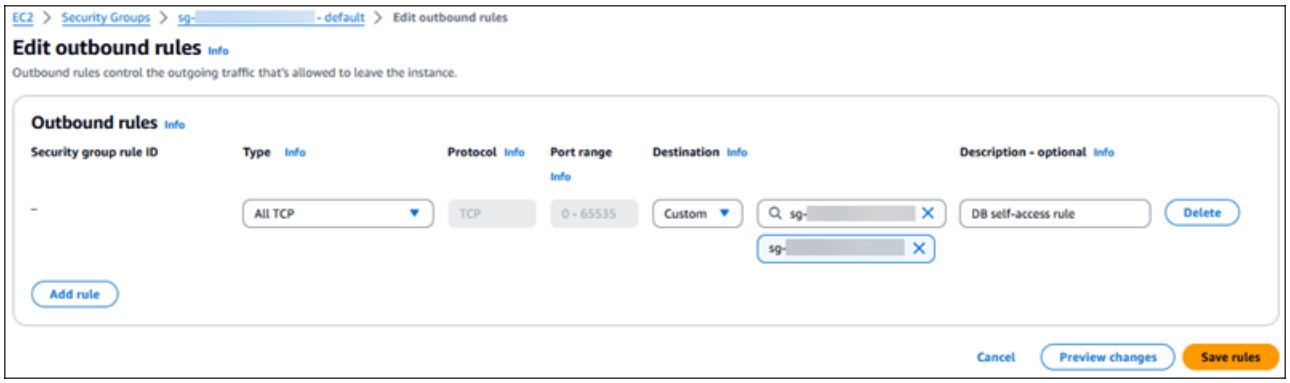


5. Gehen Sie auf der Registerkarte Regeln für ausgehenden Datenverkehr wie folgt vor:

a. Wählen Sie Edit outbound rules (Regeln für ausgehenden Datenverkehr bearbeiten).

b. Fügen Sie eine neue Regel für ausgehenden Datenverkehr für den Quell-DB-Cluster oder die Quell-DB-Instance hinzu:

- Datenbank-Port: `All traffic` (beinhaltet Ports `0-65535`)
- Sicherheitsgruppen-ID: `sg-056a84f1712b77926` in diesem Beispiel



6. Melden Sie sich an der AWS-Managementkonsole an und öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.

7. Navigieren Sie zur Seite Netzwerk-ACLs.
8. Fügen Sie die standardmäßige Netzwerk-ACL-Konfiguration hinzu, wie unter Standard-Netzwerk-ACL beschrieben.

Vorbereiten der Zieldatenbank

Folgen Sie den Verfahren unter Erstellen von Tabellen in Aurora PostgreSQL Limitless Database, um die Zieltabellen in der DB-Shard-Gruppe zu erstellen.

Ihre Zieltabellen müssen dieselben Schemata, Tabellennamen und Primärschlüssel wie die Quelltabellen haben.

Erstellen von Datenbank-Anmeldeinformationen

Sie müssen Datenbankbenutzer in den Quell- und Zieldatenbanken erstellen und den Benutzern die erforderlichen Rechte gewähren. Weitere Informationen dazu finden Sie im Abschnitt CREATE USER und GRANT der PostgreSQL-Dokumentation.

Erstellen Sie die Quelldatenbank-Anmeldeinformationen

Der Quelldatenbankbenutzer wird in dem Befehl übergeben, mit dem Laden zu beginnen. Dieser Benutzer muss über die Berechtigung verfügen, die Replikation von der Quelldatenbank aus durchzuführen.

1. Verwenden Sie den Datenbank-Masterbenutzer (oder einen anderen Benutzer mit dieser `rds_superuser`-Rolle), um einen Quelldatenbankbenutzer mit `LOGIN`-Berechtigungen zu erstellen.

```
CREATE USER source_db_username WITH PASSWORD 'source_db_user_password';
```

2. Weisen Sie Ihrem Quelldatenbankbenutzer die `rds_superuser`-Rolle zu.

```
GRANT rds_superuser to source_db_username;
```

3. Wenn Sie den Modus `full_load_and_cdc` verwenden, weisen Sie die `rds_replication`-Rolle Ihrem Quelldatenbankbenutzer zu. Die Rolle `rds_replication` erteilt Berechtigungen zur Verwaltung von logischen Slots und zum Streamen von Daten mithilfe von logischen Slots.

```
GRANT rds_replication to source_db_username;
```

Erstellen Sie die Anmeldeinformationen für die Zieldatenbank

Der Zieldatenbankbenutzer benötigt die Berechtigung, in die Zieltabellen in der DB-Shard-Gruppe zu schreiben.

1. Verwenden Sie den Datenbank-Masterbenutzer (oder einen anderen Benutzer mit dieser `rds_superuser`-Rolle), um einen Zieldatenbankbenutzer mit `LOGIN`-Berechtigungen zu erstellen.

```
CREATE USER destination_db_username WITH PASSWORD 'destination_db_user_password';
```

2. Weisen Sie Ihrem Zieldatenbankbenutzer die `rds_superuser`-Rolle zu.

```
GRANT rds_superuser to destination_db_username;
```

Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs mithilfe eines Skripts

Das Setup-Skript erstellt eine vom Kunden verwaltete Rolle AWS KMS key, eine AWS Identity and Access Management (IAM) Rolle und zwei AWS Secrets Manager Geheimnisse.

Führen Sie die folgenden Schritte aus, um das Setup-Skript zu verwenden:

1. Stellen Sie sicher, dass Sie das AWS CLI installiert und mit Ihren AWS-Konto Anmeldeinformationen konfiguriert haben.
2. Installieren Sie den JSON-Prozessor der `jq`-Befehlszeile. Weitere Informationen finden Sie unter qlang/jq.
3. Kopieren Sie die Datei data_loading_script.zip auf Ihren Computer und extrahieren Sie die Datei `data_load_aws_setup_script.sh` daraus.
4. Bearbeiten Sie das Skript und ersetzen Sie die folgenden Platzhaltervariablen durch die entsprechenden Werte:
   - Ihr AWS-Konto
   - Das AWS-Region
   - Anmeldeinformationen für die Quelldatenbank
   - Anmeldeinformationen für die Zieldatenbank
5. Öffnen Sie ein neues Terminal auf Ihrem Computer und führen Sie den folgenden Befehl aus:

```
bash ./data_load_aws_setup_script.sh
```

Setup-Skript für das Datenladeprogramm

Wir stellen den Text der Datei `data_load_aws_setup_script.sh` hier als Referenz zur Verfügung.

```
#!/bin/bash
# Aurora Limitless data loading - AWS resources setup script #
# Set up the account credentials in advance. #
# Update the following script variables. #

###################################
#### Start of variable section ####
```

```
ACCOUNT_ID="12-digit_AWS_account_ID"
REGION="AWS_Region"
DATE=$(date +'%m%d%H%M%S')
RANDOM_SUFFIX="${DATE}"
SOURCE_SECRET_NAME="secret-source-${DATE}"
SOURCE_USERNAME="source_db_username"
SOURCE_PASSWORD="source_db_password"
DESTINATION_SECRET_NAME="secret-destination-${DATE}"
DESTINATION_USERNAME="destination_db_username"
DESTINATION_PASSWORD="destination_db_password"
DATA_LOAD_IAM_ROLE_NAME="aurora-data-loader-${RANDOM_SUFFIX}"
TMP_WORK_DIR="./tmp_data_load_aws_resource_setup/"

#### End of variable section ####
################################

# Main logic start
echo "DATE - [${DATE}]"
echo "RANDOM_SUFFIX - [${RANDOM_SUFFIX}]"
echo 'START!'

mkdir -p $TMP_WORK_DIR

# Create the symmetric KMS key for encryption and decryption.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_key_response.txt"
aws kms create-key --region $REGION | tee $TMP_FILE_PATH
KMS_KEY_ARN=$(cat $TMP_FILE_PATH | jq -r '.KeyMetadata.Arn')
aws kms create-alias \
    --alias-name alias/"${DATA_LOAD_IAM_ROLE_NAME}-key" \
    --target-key-id $KMS_KEY_ARN \
    --region $REGION

# Create the source secret.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_source_secret_response.txt"
aws secretsmanager create-secret \
    --name $SOURCE_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$SOURCE_USERNAME\",\"password\":\"$SOURCE_PASSWORD
\"}" \
    --region $REGION \
    | tee $TMP_FILE_PATH
SOURCE_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the destination secret.
```

```
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_destination_secret_response.txt"
aws secretsmanager create-secret \
    --name $DESTINATION_SECRET_NAME \
    --kms-key-id $KMS_KEY_ARN \
    --secret-string "{\"username\":\"$DESTINATION_USERNAME\",\"password\":
\"$DESTINATION_PASSWORD\"}" \
    --region $REGION \
    | tee $TMP_FILE_PATH
DESTINATION_SECRET_ARN=$(cat $TMP_FILE_PATH | jq -r '.ARN')

# Create the RDS trust policy JSON file.
# Use only rds.amazonaws.com for RDS PROD use cases.
TRUST_POLICY_PATH="${TMP_WORK_DIR}rds_trust_policy.json"
echo '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}' > $TRUST_POLICY_PATH

# Create the IAM role.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_create_iam_role_response.txt"
aws iam create-role \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --assume-role-policy-document "file://${TRUST_POLICY_PATH}" \
    --tags Key=assumer,Value=aurora_limitless_table_data_load \
    --region $REGION \
    | tee $TMP_FILE_PATH
IAM_ROLE_ARN=$(cat $TMP_FILE_PATH | jq -r '.Role.Arn')

# Create the permission policy JSON file.
PERMISSION_POLICY_PATH="${TMP_WORK_DIR}data_load_permission_policy.json"
permission_json_policy=$(cat &lt;&lt;EOF
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "$SOURCE_SECRET_ARN",
                "$DESTINATION_SECRET_ARN"
            ]
        },
        {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "$KMS_KEY_ARN"
        },
        {
```

```
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
EOF
)
echo $permission_json_policy > $PERMISSION_POLICY_PATH

# Add the inline policy.
aws iam put-role-policy \
    --role-name $DATA_LOAD_IAM_ROLE_NAME \
    --policy-name aurora-limitless-data-load-policy \
    --policy-document "file://${PERMISSION_POLICY_PATH}" \
    --region $REGION

# Create the key policy JSON file.
KEY_POLICY_PATH="${TMP_WORK_DIR}data_load_key_policy.json"
key_json_policy=$(cat &lt;&lt;EOF
{
    "Id": "key-aurora-limitless-data-load-$RANDOM_SUFFIX",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::$ACCOUNT_ID:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "$IAM_ROLE_ARN"
            },
            "Action": [
```

```
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*"
        }
    ]
}
EOF
)
echo $key_json_policy > $KEY_POLICY_PATH

# Add the key policy.
TMP_FILE_PATH="${TMP_WORK_DIR}tmp_put_key_policy_response.txt"
sleep 10 # sleep 10 sec for IAM role ready
aws kms put-key-policy \
    --key-id $KMS_KEY_ARN \
    --policy-name default \
    --policy "file://${KEY_POLICY_PATH}" \
    --region $REGION \
    | tee $TMP_FILE_PATH

echo 'DONE!'

echo "ACCOUNT_ID : [${ACCOUNT_ID}]"
echo "REGION : [${REGION}]"
echo "RANDOM_SUFFIX : [${RANDOM_SUFFIX}]"
echo "IAM_ROLE_ARN : [${IAM_ROLE_ARN}]"
echo "SOURCE_SECRET_ARN : [${SOURCE_SECRET_ARN}]"
echo "DESTINATION_SECRET_ARN : [${DESTINATION_SECRET_ARN}]"

# Example of a successful run:
# ACCOUNT_ID : [012345678912]
# REGION : [ap-northeast-1]
# RANDOM_SUFFIX : [0305000703]
# IAM_ROLE_ARN : [arn:aws:iam::012345678912:role/aurora-data-loader-0305000703]
# SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-source-0305000703-yQDtow]
# DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:012345678912:secret:secret-destination-0305000703-5d5Jy8]

# If you want to manually clean up failed resource,
# please remove them in the following order:
# 1. IAM role.
```

```
 # aws iam delete-role-policy --role-name Test-Role --policy-name ExamplePolicy --
region us-east-1
# aws iam delete-role --role-name Test-Role --region us-east-1
# 2. Source and destination secrets.
# aws secretsmanager delete-secret --secret-id MyTestSecret --force-delete-without-
recovery --region us-east-1
# 3. KDM key.
# aws kms schedule-key-deletion --key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab --pending-window-in-days 7
 --region us-east-1
```

Ausgabe aus dem Setup-Skript des Dienstprogramms zum Laden von Daten

Das folgende Beispiel zeigt die Ausgabe nach erfolgreicher Ausführung des Skripts.

```
% bash ./data_load_aws_setup_script.sh
DATE - [0305000703]
RANDOM_SUFFIX - [0305000703]
START!
{
    "KeyMetadata": {
        "AWSAccountId": "123456789012",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:ap-
northeast-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2024-03-05T00:07:49.852000+00:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow",
    "Name": "secret-source-0305000703",
```

```
    "VersionId": "a017bebe-a71b-4220-b923-6850c2599c26"
}
{
    "ARN": "arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
destination-0305000703-5d5Jy8",
    "Name": "secret-destination-0305000703",
    "VersionId": "32a1f989-6391-46b1-9182-f65d242f5eb6"
}
{
    "Role": {
        "Path": "/",
        "RoleName": "aurora-data-loader-0305000703",
        "RoleId": "AROAYPX63ITQOYORQSC6U",
        "Arn": "arn:aws:iam::123456789012:role/aurora-data-loader-0305000703",
        "CreateDate": "2024-03-05T00:07:54+00:00",
        "AssumeRolePolicyDocument": {
           "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "rds.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Tags": [
            {
                "Key": "assumer",
                "Value": "aurora_limitless_table_data_load"
            }
        ]
    }
}
DONE!
ACCOUNT_ID : [123456789012]
REGION : [ap-northeast-1]
RANDOM_SUFFIX : [0305000703]
IAM_ROLE_ARN : [arn:aws:iam::123456789012:role/aurora-data-loader-0305000703]
SOURCE_SECRET_ARN : [arn:aws:secretsmanager:ap-northeast-1:123456789012:secret:secret-
source-0305000703-yQDtow]
```

```
DESTINATION_SECRET_ARN : [arn:aws:secretsmanager:ap-
northeast-1:123456789012:secret:secret-destination-0305000703-5d5Jy8]
```

Bereinigen von fehlerhaften Ressourcen

Wenn Sie fehlerhafte Ressourcen manuell bereinigen möchten, entfernen Sie sie in der folgenden Reihenfolge:

1. IAM-Rolle, zum Beispiel:

```
aws iam delete-role-policy \
--role-name Test-Role \
--policy-name ExamplePolicy

aws iam delete-role \
--role-name Test-Role
```

2. Quell- und Ziel-Secrets, zum Beispiel:

```
aws secretsmanager delete-secret \
--secret-id MyTestSecret \
--force-delete-without-recovery
```

3. KMS-Schlüssel, zum Beispiel:

```
aws kms schedule-key-deletion \
--key-id arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab
 \
--pending-window-in-days 7
```

Danach können Sie das Skript erneut ausführen.

Manuelles Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs

Das manuelle Verfahren zum Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs umfasst die folgenden Schritte:

1. Erstellung des vom Kunden verwalteten AWS KMS key
2. Hinzufügen der Berechtigungsrichtlinien für die IAM-Rolle
3. Erstellen der Datenbank-Secrets
4. Erstellen der IAM-Rolle
5. Aktualisierung der vom Kunden verwalteten AWS KMS key

Dieser Vorgang ist optional und umfasst dieselben Aufgaben wie in Einrichten der Datenbankauthentifizierung und des Ressourcenzugriffs mithilfe eines Skripts. Wir empfehlen die Verwendung des Skripts.

Erstellung des vom Kunden verwalteten AWS KMS key

Folgen Sie den Verfahren unter Erstellen symmetrischer Verschlüsselungsschlüssel, um einen kundenverwalteten KMS-Schlüssel zu erstellen. Sie können auch einen vorhandenen Schlüssel verwenden, wenn er diese Anforderungen erfüllt.

So erstellen Sie einen kundenverwalteten KMS-Schlüssel

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die AWS KMS Konsole unter https://console.aws.amazon.com/kms.
2. Navigieren Sie zur Seite Kundenverwaltete Schlüssel.
3. Klicken Sie auf Create key.
4. Gehen Sie auf der Seite Schlüssel konfigurieren wie folgt vor:
    a. Wählen Sie als Schlüsseltyp die Option Symmetrisch aus.
    b. Wählen Sie für Schlüsselnutzung die Option Verschlüsseln und entschlüsseln aus.
    c. Wählen Sie Weiter aus.
5. Geben Sie auf der Seite Bezeichnungen hinzufügen einen Alias wie **`limitless`** ein und wählen Sie dann Weiter aus.

6. Vergewissern Sie sich, dass auf der Seite Schlüsselverwaltungsberechtigungen definieren das Kontrollkästchen Schlüsseladministratoren das Löschen dieses Schlüssels erlauben aktiviert ist, und wählen Sie dann Weiter aus.
7. Klicken Sie auf der Seite Schlüsselnutzungsberechtigungen definieren auf Weiter.
8. Klicken Sie auf der Seite Prüfen auf Beenden.

   Sie aktualisieren die Schlüsselrichtlinie später.

Notieren Sie sich die Amazon-Ressourcennamen (ARN) des KMS-Schlüssels, der in Hinzufügen der Berechtigungsrichtlinien für die IAM-Rolle verwendet werden soll.

Informationen zur Verwendung des AWS CLI zur Erstellung des vom Kunden verwalteten KMS-Schlüssels finden Sie unter create-key und create-alias.

Erstellen der Datenbank-Secrets

Damit das Datenladeprogramm auf die Quell- und Zieldatenbanktabellen zugreifen kann, erstellen Sie zwei geheime Daten in AWS Secrets Manager: einen für die Quelldatenbank und einen für die Zieldatenbank. In diesen Secrets werden die Benutzernamen und Passwörter für den Zugriff auf die Quell- und Zieldatenbanken gespeichert.

Folgen Sie den Verfahren unter AWS Secrets Manager -Secret erstellen, um die Schlüssel- und Wertepaar-Secrets zu erstellen.

So erstellen Sie die Datenbank-Secrets

1. Öffnen Sie die Secrets Manager Manager-Konsole unter https://console.aws.amazon.com/secretsmanager/.
2. Wählen Sie Store a new secret (Ein neues Secret speichern).
3. Gehen Sie auf der Seite Secret-Typ auswählen wie folgt vor:

   a. Wählen Sie für Secret-Typ die Option Anderer Secret-Typ aus.
   b. Wählen Sie für Schlüssel/Wert-Paare die Registerkarte Klartext aus.
   c. Geben Sie den folgenden JSON-Code ein, wobei *sourcedbreader* und *sourcedbpassword* die Anmeldeinformationen des Benutzers der Quelldatenbank aus Erstellen Sie die Quelldatenbank-Anmeldeinformationen sind.

```
{
```

```
    "username":"sourcedbreader",
    "password":"sourcedbpassword"
}
```

d. Wählen Sie für Verschlüsselungsschlüssel beispielsweise den KMS-Schlüssel aus, den Sie in Erstellung des vom Kunden verwalteten AWS KMS key erstellt haben, z. B. `limitless`.

e. Wählen Sie Weiter aus.

4. Geben Sie auf der Seite Secret konfigurieren einen Secret-Namen ein, wie z. B. **`source_DB_secret`**, und wählen Sie dann Weiter aus.
5. Wählen Sie auf der Seite Drehung konfigurieren – optional die Option Weiter aus.
6. Wählen Sie auf der Seite Review (Überprüfung) Store (Speichern) aus.
7. Wiederholen Sie dieses Verfahren für das Secret der Zieldatenbank:

   a. Geben Sie den folgenden JSON-Code ein, wobei *`destinationdbwriter`* und *`destinationdbpassword`* die Anmeldeinformationen des Benutzers der Zieldatenbank aus Erstellen Sie die Anmeldeinformationen für die Zieldatenbank sind.

```
{
    "username":"destinationdbwriter",
    "password":"destinationdbpassword"
}
```

   b. Geben Sie einen Secret-Namen ein, z. B. **`destination_DB_secret`**.

Notieren Sie sich ARNs die Geheimnisse, in denen Sie verwenden möchtenHinzufügen der Berechtigungsrichtlinien für die IAM-Rolle.

Erstellen der IAM-Rolle

Für das Laden von Daten müssen Sie Zugriff auf AWS Ressourcen gewähren. Um Zugriff zu gewähren, erstellen Sie die IAM-Rolle `aurora-data-loader`, indem Sie die Schritte unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen IAM-Benutzer ausführen.

So erstellen Sie die IAM-Rolle

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Navigieren Sie zur Seite Rollen.

3. Wählen Sie Rolle erstellen aus.
4. Gehen Sie auf der Seite Vertrauenswürdige Entität auswählen wie folgt vor:

   a. Wählen Sie für Vertrauenstyp der Entität die Option Benutzerdefinierte Vertrauensrichtlinie aus.

   b. Geben Sie für die benutzerdefinierte Vertrauensrichtlinie den folgenden JSON-Code ein:

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

   c. Wählen Sie Weiter aus.
5. Wählen Sie auf der Seite Add permissions (Berechtigungen hinzufügen) die Option Next (Weiter) aus.
6. Gehen Sie auf der Seite Benennen, prüfen und erstellen wie folgt vor:

   a. Geben Sie für Rollenname **`aurora-data-loader`** oder einen von Ihnen bevorzugten Namen ein.

   b. Wählen Sie Tag hinzufügen aus und geben Sie das folgende Tag ein:

      - Schlüssel: **`assumer`**
      - Value (Wert): **`aurora_limitless_table_data_load`**

> **Important**
>
> Aurora PostgreSQL Limitless Database kann nur eine IAM-Rolle annehmen, die über dieses Tag verfügt.

c. Wählen Sie Rolle erstellen aus.

Aktualisierung der vom Kunden verwalteten AWS KMS key

Folgen Sie den Verfahren unter Ändern einer Schlüsselrichtlinie, um die IAM-Rolle `aurora-data-loader` zur Standard-Schlüsselrichtlinie hinzuzufügen.

So fügen Sie die IAM-Rolle zur Schlüsselrichtlinie hinzu

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die AWS KMS Konsole unter https://console.aws.amazon.com/kms.
2. Navigieren Sie zur Seite Kundenverwaltete Schlüssel.
3. Wählen Sie den KMS-Schlüssel aus, den Sie in Erstellung des vom Kunden verwalteten AWS KMS key erstellt haben, z. B. `limitless`.
4. Wählen Sie auf der Registerkarte Schlüsselrichtlinie für Schlüsselbenutzer die Option Hinzufügen aus.
5. Wählen Sie im Fenster „Schlüsselbenutzer hinzufügen“ beispielsweise den Namen der IAM-Rolle ausErstellen der IAM-Rolle, in der Sie sie erstellt haben. aurora-data-loader
6. Wählen Sie Hinzufügen aus.

Hinzufügen der Berechtigungsrichtlinien für die IAM-Rolle

Sie müssen der erstellten IAM-Rolle Berechtigungen hinzufügen. Auf diese Weise kann das Dienstprogramm zum Laden von Daten von Aurora PostgreSQL Limitless Database auf verwandte AWS -Ressourcen zugreifen, um Netzwerkverbindungen aufzubauen und die Secrets der Anmeldeinformationen der Quell- und Ziel-DB abzurufen.

Weitere Informationen finden Sie unter Ändern einer Rolle.

So fügen Sie Berechtigungsrichtlinien hinzu

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Navigieren Sie zur Seite Rollen.
3. Wählen Sie beispielsweise die IAM-Rolle ausErstellen der IAM-Rolle, in der Sie erstellt haben. aurora-data-loader
4. Wählen Sie auf der Registerkarte Berechtigungen für Berechtigungsrichtlinien die Option Berechtigungen hinzufügen und dann Inline-Richtlinie erstellen aus.
5. Wählen Sie auf der Seite Berechtigungen angeben den JSON-Editor aus.
6. Kopieren Sie die folgende Vorlage und fügen Sie sie in den JSON-Editor ein. Ersetzen Sie dabei die Platzhalter durch die ARNs für Ihre Datenbankgeheimnisse und den KMS-Schlüssel.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Ec2Permission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfacePermissions",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeNetworkAcls"
            ],
            "Resource": "*"
        },
        {
```

```
            "Sid": "SecretsManagerPermissions",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:source_DB_secret-ABC123",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:destination_DB_secret-456DEF"
            ]
        },       {
            "Sid": "KmsPermissions",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:123456789012:key/aa11bb22-
####-####-####-fedcba123456"
        },
        {
            "Sid": "RdsPermissions",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

7. Prüfen Sie, ob Fehler vorliegen, und beheben Sie sie.
8. Wählen Sie Weiter aus.
9. Geben Sie auf der Seite Überprüfen und erstellen einen Richtliniennamen ein wie z. B. **`data_loading_policy`**, und wählen Sie anschließend Richtlinie erstellen aus.

Laden von Daten aus einem DB-Cluster von Aurora PostgreSQL oder einer DB-Instance von RDS für Postgre

Nachdem Sie die Ressourcen- und Authentifizierungseinrichtung abgeschlossen haben, stellen Sie eine Verbindung zum Cluster-Endpunkt her und rufen Sie die gespeicherte Prozedur `rds_aurora.limitless_data_load_start` von einer Limitless-Datenbank wie `postgres_limitless` aus auf. Die Limitless-Datenbank ist eine Datenbank in der DB-Shard-Gruppe, in die Sie Daten migrieren möchten.

Diese Funktion stellt im Hintergrund asynchron eine Verbindung zu der im Befehl angegebenen Quelldatenbank her, liest die Daten aus der Quelle und lädt die Daten auf die Shards. Für eine bessere Leistung werden die Daten mithilfe paralleler Threads geladen. Die Funktion ruft einen Point-in-Time-Tabellen-Snapshot ab, indem sie einen `SELECT`-Befehl zum Lesen der Daten der im Befehl angegebenen Tabelle(n) ausführt.

Sie können Daten in Sharded-, Referenz- und Standardtabellen laden.

Sie können Daten in `rds_aurora.limitless_data_load_start`-Aufrufen auf Datenbank-, Schema- oder Tabellenebene laden.

- Datenbank: Sie können bei jedem Aufruf jeweils eine Datenbank laden. Die Anzahl der Schemata oder Tabellen innerhalb der Datenbank ist dabei nicht begrenzt.
- Schema: Sie können bei jedem Aufruf maximal 15 Schemata laden. Die Anzahl der Tabellen innerhalb jedes Schemas ist dabei nicht begrenzt.
- Tabelle: Sie können bei jedem Aufruf maximal 15 Tabellen laden.

> **Note**
>
> Diese Funktion verwendet keine Amazon-RDS-Snapshots oder Point-in-Time-Isolierung der Datenbank. Aus Gründen der tabellenübergreifenden Konsistenz empfehlen wir, die Quelldatenbank zu klonen und auf diese geklonte Datenbank als Quelle zu verweisen.

Die gespeicherte Prozedur hat die folgende Syntax.

```
CALL rds_aurora.limitless_data_load_start('source_type',
    'source_DB_cluster_or_instance_ID',
    'source_database_name',
    'streaming_mode',
    'data_loading_IAM_role_arn',
    'source_DB_secret_arn',
    'destination_DB_secret_arn',
    'ignore_primary_key_conflict_boolean_flag',
    'is_dry_run',
    (optional parameter) schemas/tables => ARRAY['name1', 'name2', ...]);
```

Es werden folgende Eingabeparameter verwendet:

- `source_type`: Der Quelltyp: `aurora_postgresql` oder `rds_postgresql`
- `source_DB_cluster_or_instance_ID`: Die ID des DB-Clusters von Aurora-PostgreSQL oder die ID der DB-Instance von RDS für PostgreSQL
- `source_database_name`: Der Name der Quelldatenbank, z. B. *`Postgres`*
- `streaming_mode`: Ob die Erfassung von Datenänderungen (Change Data Capture, CDC) einbezogen werden soll: `full_load` oder `full_load_and_cdc`
- `data_loading_IAM_role_arn`: Der Amazon-Ressourcenname (ARN) der IAM-Rolle für `aurora-data-loader`
- `source_DB_secret_arn`: Der geheime ARN der Quell-DB
- `destination_DB_secret_arn`: Der geheime ARN der Ziel-DB
- `ignore_primary_key_conflict_boolean_flag`: Ob fortgefahren werden soll, wenn ein Primärschlüssel-Konflikt auftritt:
  - Wenn auf `true` gesetzt, werden beim Laden von Daten neue Änderungen für Zeilen mit einem Primärschlüssel-Konflikt ignoriert.
  - Wenn auf `false` gesetzt, werden beim Laden von Daten die vorhandenen Zeilen in Zieltabellen überschrieben, wenn ein Primärschlüssel-Konflikt auftritt.
- `is_dry_run`: Ob getestet werden soll, ob der Auftrag zum Laden von Daten eine Verbindung zu den Quell- und Zieldatenbanken herstellen kann:
  - Wenn auf `true` gesetzt, werden die Verbindungen getestet, ohne dass Daten geladen werden
  - Wenn auf `false` gesetzt, werden Daten geladen

- (optional) `schemas` oder `tables`: Ein Array von Schemata oder Tabellen, die geladen werden sollen. Sie können einen der folgenden Werte angeben:
  - Eine Liste von Tabellen im Format `tables => ARRAY['schema1.table1', 'schema1.table2', 'schema2.table1', ...]`
  - Eine Liste von Schemata im Format `schemas => ARRAY['schema1', 'schema2', ...]`

  Wenn Sie diesen Parameter nicht einschließen, wird die gesamte angegebene Quelldatenbank migriert.

Der Ausgabeparameter ist die Auftrags-ID mit einer Nachricht.

Im folgenden Beispiel wird veranschaulicht, wie die gespeicherte Prozedur `rds_aurora.limitless_data_load_start` zum Laden von Daten aus einem DB-Cluster von Aurora PostgreSQL verwendet wird.

```
CALL rds_aurora.limitless_data_load_start('aurora_postgresql',
    'my-db-cluster',
    'postgres',
    'full_load_and_cdc',
    'arn:aws:iam::123456789012:role/aurora-data-loader-8f2c66',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-source-8f2c66-EWrr0V',
    'arn:aws:secretsmanager:us-east-1:123456789012:secret:secret-destination-8f2c66-
d04fbD',
    'true',
    'false',
    tables => ARRAY['public.customer', 'public.order', 'public.orderdetails']);

INFO: limitless data load job id 1688761223647 is starting.
```

Überwachen des Ladens von Daten

Aurora PostgreSQL Limitless Database bietet mehrere Möglichkeiten, Aufträge zum Laden von Daten überwachen:

- Auflisten von Aufträgen zum Laden von Daten
- Anzeigen der Details von Aufträgen zum Laden von Daten mithilfe der Auftrags-ID
- Überwachen der Amazon-CloudWatch-Protokollgruppe
- Überwachen von RDS-Ereignissen

Auflisten von Aufträgen zum Laden von Daten

Sie können eine Verbindung zum Cluster-Endpunkt herstellen und die Ansicht `rds_aurora.limitless_data_load_jobs` verwenden, um Aufträge zum Laden von Daten aufzulisten.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_jobs LIMIT 6;

    job_id     |  status   | message |     source_db_identifier      | source_db_name |
 full_load_complete_time |
   progress_details                                                                 |
     start_time       |   last_updated_time    |  streaming_mode   | source_engine_type
 | ignore_primary_key_conflict | is_dryrun
---------------+-----------+---------+-------------------------------+---------------
+-------------------------
+-------------------------------------------------------------------------------------------------------------------------
+------------------------+-----------------------+------------------
+--------------------+-----------------------------+-----------
 1725697520693 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:48:15+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "9
 of 9 tables loaded", "COMPLETED_AT": "2024/09/07 08:48:15+00", "RECORDS_MIGRATED":
 600003}} | 2024-09-07 08:47:13+00 | 2024-09-07 08:48:15+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725696114225 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:24:20+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "3
 of 3 tables loaded", "COMPLETED_AT": "2024/09/07 08:24:20+00", "RECORDS_MIGRATED":
 200001}} | 2024-09-07 08:23:56+00 | 2024-09-07 08:24:20+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725696067630 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 08:23:45+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "6
 of 6 tables loaded", "COMPLETED_AT": "2024/09/07 08:23:45+00", "RECORDS_MIGRATED":
```

```
 400002}} | 2024-09-07 08:23:10+00 | 2024-09-07 08:23:45+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725694221753 | CANCELED  |         | persistent-kdm-auto-source-01 | postgres
    |                         | {}

           | 2024-09-07 07:31:18+00 | 2024-09-07 07:51:49+00 | full_load_and_cdc |
 aurora_postgresql  | t                           | f
 1725691698210 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:51+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:51+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:42+00 | 2024-09-07 07:10:52+00 | full_load         |
 aurora_postgresql  | t                           | f
 1725691695049 | COMPLETED |         | persistent-kdm-auto-source-01 | postgres
    | 2024-09-07 07:10:48+00  | {"FULL_LOAD": {"STATUS": "COMPLETED", "DETAILS": "1
 of 1 tables loaded", "COMPLETED_AT": "2024/09/07 07:10:48+00", "RECORDS_MIGRATED":
 100000}} | 2024-09-07 07:10:41+00 | 2024-09-07 07:10:48+00 | full_load         |
 aurora_postgresql  | t                           | f
(6 rows)
```

Datensätze von Aufträgen werden nach 90 Tagen gelöscht.

Anzeigen der Details von Aufträgen zum Laden von Daten mithilfe der Auftrags-ID

Wenn Sie die Auftrags-ID kennen, können Sie eine Verbindung zum Cluster-Endpunkt herstellen und die Ansicht `rds_aurora.limitless_data_load_job_details` verwenden, um die Details dieses Auftrags zum Laden von Daten anzuzeigen, einschließlich des Tabellennamens, des Auftragsstatus und der Anzahl der geladenen Zeilen. Sie können die Auftrags-ID in den Antworten auf die Startfunktionen für das Laden von Daten oder in der Ansicht `rds_aurora.limitless_data_load_jobs` abrufen.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_data_load_job_details WHERE
 job_id='1725696114225';

job_id        | destination_table_name | destination_schema_name | start_time
      | status    | full_load_rows | full_load_total_rows | full_load_complete_time |
 cdc_insert | cdc_update | cdc_delete
--------------+------------------------+-------------------------
+-------------------------+-----------+----------------+---------------------
+-------------------------+------------+------------+------------
1725696114225 | standard_1             | public                  | 2024-09-07
 08:23:57+00 | COMPLETED | 100000         | 100000              | 2024-09-07
 08:24:08+00  | 0          | 0          | 0
```

```
1725696114225 | standard_2             | public                  | 2024-09-07
 08:24:08+00 | COMPLETED | 100000        | 100000               | 2024-09-07
 08:24:17+00  | 0          | 0          | 0
1725696114225 | standard_3             | public                  | 2024-09-07
 08:24:18+00 | COMPLETED | 1             | 1                    | 2024-09-07
 08:24:20+00  | 0          | 0          | 0
1725696114225 | standard_4             | public                  | 2024-09-07
 08:23:58+00 | PENDING   | 0             | 0                    |
    | 0          | 0          | 0
(4 rows)
```

Datensätze von Aufträgen werden nach 90 Tagen gelöscht.

Überwachen der Amazon-CloudWatch-Protokollgruppe

Nachdem sich der Status des Auftrags zum Laden von Daten in `RUNNING` geändert hat, können Sie den Fortschritt der Laufzeit mithilfe von Amazon CloudWatch Logs überprüfen.

So überwachen Sie CloudWatch-Protokollstreams

Melden Sie sich bei AWS-Managementkonsole an und öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.

1. Navigieren Sie zu Protokolle und dann zu Protokollgruppen.
2. Wählen Sie die Protokollgruppe /aws/rds/aurora-limitless-database aus.
3. Suchen Sie per job_id nach dem Protokollstream Ihres Auftrags zum Laden von Daten.

   Der Protokollstream hat das Muster Data-Load-Job-***job_id***.
4. Wählen Sie den Protokollstream aus, um die Protokollereignisse anzusehen.

Jeder Protokollstream zeigt Ereignisse, die den Auftragsstatus und die Anzahl der Zeilen enthalten, die in die Zieltabellen von Aurora PostgreSQL Limitless Database geladen wurden. Wenn ein Auftrag zum Laden von Daten fehlschlägt, wird auch ein Fehlerprotokoll erstellt, das den Fehlerstatus und die Ursache anzeigt.

Datensätze von Aufträgen werden nach 90 Tagen gelöscht.

Überwachen von RDS-Ereignissen

Der Auftrag zum Laden von Daten veröffentlicht auch RDS-Ereignisse, z. B. wenn ein Auftrag erfolgreich ist, fehlschlägt oder abgebrochen wird. Sie können die Ereignisse in der Zieldatenbank ansehen.

Weitere Informationen finden Sie unter DB-Shard-Gruppenereignisse.

Abbrechen eines Auftrags zum Laden von Daten

Um einen Auftrag zum Laden von Daten abzubrechen, rufen Sie die gespeicherte Prozedur `rds_aurora.limitless_data_load_cancel` mit der Auftrags-ID als Eingabeparameter auf. Sie rufen diese gespeicherte Prozedur aus derselben Datenbank in der DB-Shard-Gruppe auf, in der der spezifische Auftrag zum Laden von Daten gestartet wurde. Zum Beispiel:

```
CALL rds_aurora.limitless_data_load_cancel(12345);

INFO: limitless data load job with id 12345 is canceling without rollback.
```

Sie können einen Auftrag zum Laden von Daten, der nicht existiert oder nicht in derselben DB-Shard-Gruppe ausgeführt wird, nicht abbrechen.

Das Dienstprogramm zum Laden von Daten in Aurora PostgreSQL Limitless Database belässt geladene Daten ohne Rollback in den Zieltabellen, wie die Antwort zeigt. Wenn Sie die geladenen Daten nicht behalten möchten, empfehlen wir, die Zieltabellen zu kürzen.

Abfragen einer Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database ist mit der PostgreSQL-Syntax für Abfragen kompatibel. Sie können Ihre Limitless Database mit `psql` oder einem anderen Verbindungsdienstprogramm abfragen, das mit PostgreSQL funktioniert. Um Abfragen auszuführen, stellen Sie eine Verbindung zum Limitless-Endpunkt her, wie unter Herstellen einer Verbindung zu Ihrem Aurora PostgreSQL Limitless Database-DB-Cluster gezeigt.

Alle PostgreSQL-`SELECT`-Abfragen werden in Aurora PostgreSQL Limitless Database unterstützt. Abfragen werden jedoch auf zwei Ebenen ausgeführt:

1. Router, an den der Client die Anfrage sendet
2. Shards, auf denen sich die Daten befinden

Die Leistung hängt davon ab, ob die Datenbank so abgefragt wird, dass ein hohes Maß an gleichzeitiger Verarbeitung verschiedener Abfragen auf verschiedenen Shards erreicht werden kann. Abfragen werden zunächst auf der Ebene der verteilten Transaktion (Router) analysiert. Vor der Planung der Abfrage findet eine Analysephase statt, in der der Ort aller an der Abfrage beteiligten Beziehungen ermittelt wird. Wenn es sich bei allen Beziehungen um Shard-Tabellen mit einem gefilterten Shard-Schlüssel auf demselben Shard oder um Referenztabellen handelt, wird die Abfrageplanung auf der Router-Ebene übersprungen und zur Planung und Ausführung vollständig auf den Shard übertragen. Dieser Prozess reduziert die Anzahl der Roundtrips zwischen verschiedenen Knoten (Router und Shard) und führt in den meisten Fällen zu einer besseren Leistung. Weitere Informationen finden Sie unter Single-Shard-Abfragen in Aurora PostgreSQL Limitless Database.

> **Note**
>
> In bestimmten Fällen, z. B. bei einem kartesischen Produkt (Cross-Join), kann die Leistung der Abfrage optimiert werden, indem Daten separat vom Shard abgerufen werden.

Weitere Informationen über Ausführungspläne zu Abfragen finden Sie in der Referenz zu Aurora PostgreSQL Limitless Database unter EXPLAIN. Allgemeine Informationen zu Abfragen finden Sie im Abschnitt Abfragen der PostgreSQL-Dokumentation.

Themen

- Single-Shard-Abfragen in Aurora PostgreSQL Limitless Database

- Verteilte Abfragen in Aurora PostgreSQL Limitless Database
- Verteiltes Abfrage-Tracing in PostgreSQL-Protokollen in Aurora PostgreSQL Limitless Database
- Verteilte Deadlocks in Aurora PostgreSQL Limitless Database

Single-Shard-Abfragen in Aurora PostgreSQL Limitless Database

Eine Single-Shard-Abfrage ist eine Abfrage, die direkt auf einem Shard ausgeführt werden kann, wobei die SQL-ACID-Semantik beibehalten wird. Wenn der Abfrageplaner auf dem Router auf eine solche Abfrage trifft, erkennt der Planer sie und leitet die gesamte SQL-Abfrage an den entsprechenden Shard weiter.

Diese Optimierung reduziert die Anzahl der Netzwerk-Roundtrips vom Router zum Shard und verbessert die Leistung. Derzeit wird diese Optimierung für die Abfragen `INSERT`, `SELECT`, `UPDATE` und `DELETE` durchgeführt.

Themen

- Beispiele für Single-Shard-Abfragen
- Einschränkungen für Single-Shard-Abfragen
- Vollständig qualifizierte (explizite) Joins
- Festlegen eines aktiven Shard-Schlüssels

Beispiele für Single-Shard-Abfragen

Die folgenden Beispiele umfassen die Sharded-Tabelle `customers` mit dem Shard-Schlüssel `customer_id` und die Referenztabelle `zipcodes`.

SELECT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM customers WHERE
 customer_id = 100;

                        QUERY PLAN
-----------------------------------------------------------
 Foreign Scan
   Output: customer_id, other_id, customer_name, balance
   Remote SQL:  SELECT customer_id,
     other_id,
     customer_name,
```

```
     balance
    FROM public.customers
   WHERE (customer_id = 100)
 Single Shard Optimized
(9 rows)
```

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM orders
    LEFT JOIN zipcodes ON orders.zipcode_id = zipcodes.zipcode_id
    WHERE customer_id = 11;

                                                QUERY PLAN

-----------------------------------------------------------------------------------------------
 Foreign Scan
   Output: customer_id, order_id, zipcode_id, customer_name, balance,
 zipcodes.zipcode_id, zipcodes.city
   Remote SQL:  SELECT orders.customer_id,
     orders.order_id,
     orders.zipcode_id,
     orders.customer_name,
     orders.balance,
     zipcodes.zipcode_id,
     zipcodes.city
    FROM (public.orders
      LEFT JOIN public.zipcodes ON ((orders.zipcode_id = zipcodes.zipcode_id)))
   WHERE (orders.customer_id = 11)
 Single Shard Optimized
(13 rows)
```

INSERT

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) INSERT INTO customers
    (customer_id, other_id, customer_name, balance)
    VALUES (1, 10, 'saikiran', 1000);

                      QUERY PLAN
------------------------------------------------------
 Insert on public.customers
   ->  Result
         Output: 1, 10, 'saikiran'::text, '1000'::real
 Single Shard Optimized
(4 rows)
```

AKTUALISIERUNG

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) UPDATE orders SET balance = balance +
 100
    WHERE customer_id = 100;

                                        QUERY PLAN

-----------------------------------------------------------------------------------------
 Update on public.orders
   Foreign Update on public.orders_fs00002 orders_1
   ->  Foreign Update
         Remote SQL:  UPDATE public.orders SET balance = (balance + (100)::double
 precision)
   WHERE (customer_id = 100)
 Single Shard Optimized
(6 rows)
```

DELETE

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) DELETE FROM orders
    WHERE customer_id = 100 and balance = 0;

                             QUERY PLAN
---------------------------------------------------------------------
 Delete on public.orders
   Foreign Delete on public.orders_fs00002 orders_1
   ->  Foreign Delete
         Remote SQL:  DELETE FROM public.orders
   WHERE ((customer_id = 100) AND (balance = (0)::double precision))
 Single Shard Optimized
(6 rows)
```

Einschränkungen für Single-Shard-Abfragen

Für Single-Shard-Abfragen gelten die folgenden Einschränkungen:

Funktionen

Wenn eine Single-Shard-Abfrage eine Funktion enthält, kommt die Abfrage nur dann für die Single-Shard-Optimierung in Frage, wenn eine der folgenden Bedingungen zutrifft:

- Die Funktion ist unveränderlich. Weitere Informationen finden Sie unter Funktionsvolatilität.
- Die Funktion ist veränderbar, aber in der Ansicht `rds_aurora.limitless_distributed_functions` registriert. Weitere Informationen finden Sie unter Verteilung der Funktionen.

Ansichten

Wenn eine Abfrage eine oder mehrere Ansichten enthält, ist die Single-Shard-Optimierung für die Abfrage deaktiviert, wenn sie eine der folgenden Bedingungen erfüllt:

- Jede Ansicht hat das Attribut `security_barrier`.
- Für die in der Abfrage verwendeten Objekte sind mehrere Benutzerberechtigungen erforderlich. Eine Abfrage enthält beispielsweise zwei Ansichten, und die Ansichten werden unter zwei verschiedenen Benutzern ausgeführt.

```
CREATE VIEW v1 AS SELECT customer_name FROM customers c WHERE c.customer_id =  1;
CREATE VIEW v2 WITH (security_barrier) AS SELECT customer_name FROM customers c
 WHERE c.customer_id =  1;

postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v1;
                                      QUERY PLAN
------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Output: customer_name
   Remote Plans from Shard postgres_s3:
         Seq Scan on public.customers_ts00001 c  (cost=0.00..24.12 rows=6 width=32)
           Output: c.customer_name
           Filter: (c.customer_id = 1)
         Query Identifier: -6005737533846718506
   Remote SQL:  SELECT customer_name
    FROM ( SELECT c.customer_name
             FROM public.customers c
            WHERE (c.customer_id = 1)) v1
 Query Identifier: -5754424854414896228
(12 rows)


postgres_limitless=> EXPLAIN VERBOSE SELECT * FROM v2;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Foreign Scan on public.customers_fs00001 c  (cost=100.00..128.41 rows=7 width=32)
   Output: c.customer_name
   Remote Plans from Shard postgres_s3:
```

```
         Seq Scan on public.customers_ts00001 customers  (cost=0.00..24.12 rows=6
 width=32)
           Output: customers.customer_name
           Filter: (customers.customer_id = 1)
         Query Identifier: 4136563775490008117
   Remote SQL: SELECT customer_name FROM public.customers WHERE ((customer_id = 1))
 Query Identifier: 5056054318010163757
(9 rows)
```

Anweisungen PREPARE und EXECUTE

Aurora PostgreSQL Limitless Database unterstützt die Single-Shard-Optimierung für vorbereitete `SELECT`-, `UPDATE`- und `DELETE`-Anweisungen.

Wenn Sie jedoch vorbereitete Anweisungen für `PREPARE` und `EXECUTE` verwenden und `plan_cache_mode` dabei auf `'force_generic_plan'` festgelegt haben, lehnt der Abfrageplaner die Single-Shard-Optimierung für diese Abfrage ab.

PL/pgSQL

Abfragen mit PL/pgSQL-Variablen werden als implizit vorbereitete Anweisungen ausgeführt. Wenn eine Abfrage PL/pgSQL-Variablen enthält, lehnt der Abfrageplaner die Single-Shard-Optimierung ab.

Die Optimierung wird im PL/pgSQL-Block unterstützt, wenn die Anweisung keine PL/pgSQL-Variablen enthält.

Vollständig qualifizierte (explizite) Joins

Die Single-Shard-Optimierung basiert auf der Eliminierung von Partitionen. Der PostgreSQL-Optimierer eliminiert Partitionen auf der Grundlage konstanter Bedingungen. Wenn Aurora PostgreSQL Limitless Database feststellt, dass sich alle verbleibenden Partitionen und Tabellen auf demselben Shard befinden, markiert sie die Abfrage, die für die Single-Shard-Optimierung in Frage kommt. Alle Filterbedingungen müssen explizit sein, damit das Eliminieren von Partitionen funktioniert. Aurora PostgreSQL Limitless Database kann Partitionen ohne ein oder mehrere Join- oder Filterprädikate in den Shard-Schlüsseln jeder Sharded-Tabelle in der Anweisung nicht eliminieren.

Gehen Sie davon aus, dass wir die Tabellen `customers`, `orders` und `order_details` auf Grundlage der Spalte `customer_id` partitioniert haben. In diesem Schema versucht die Anwendung, alle Daten für einen Kunden auf einem einzigen Shard zu speichern.

Betrachten Sie folgende Abfrage:

```
SELECT * FROM
    customers c, orders o, order_details od
WHERE c.customer_id = o.customer_id
    AND od.order_id = o.order_id
    AND c.customer_id = 1;
```

Diese Abfrage ruft alle Daten für einen Kunden (`c.customer_id = 1`) ab. Die Daten für diesen Kunden befinden sich auf einem einzigen Shard, aber Aurora PostgreSQL Limitless Database qualifiziert diese Abfrage nicht als Single-Shard-Abfrage. Der Optimierungsprozess für die Abfrage sieht wie folgt aus:

1. Der Optimierer kann Partitionen für `customers` und `orders` auf Grundlage der folgenden Bedingung entfernen:

   ```
   c.customer_id = 1
   c.customer_id = o.customer_id
   o.customer_id =  1 (transitive implicit condition)
   ```

2. Der Optimierer kann keine Partitionen für `order_details` eliminieren, da es in der Tabelle keine konstante Bedingung gibt.
3. Der Optimierer kommt zu dem Schluss, dass er alle Partitionen aus `order_details` gelesen hat. Daher kann die Abfrage nicht für die Single-Shard-Optimierung qualifiziert werden.

Um sie zu einer Single-Shard-Abfrage zu machen, fügen wir die folgende explizite Join-Bedingung hinzu:

```
o.customer_id = od.customer_id
```

Die geänderte Abfrage sieht wie folgt aus:

```
SELECT * FROM
    customers c, orders o,  order_details od
WHERE c.customer_id = o.customer_id
     AND o.customer_id = od.customer_id
     AND od. order_id = o. order_id
 AND c.customer_id =  1;
```

Jetzt kann der Optimierer Partitionen für `order_details` eliminieren. Die neue Abfrage wird zu einer Single-Shard-Abfrage und ist für die Optimierung geeignet.

Festlegen eines aktiven Shard-Schlüssels

Mit diesem Feature können Sie bei der Abfrage der Datenbank einen einzelnen Shard-Schlüssel festlegen, wodurch an alle `SELECT`- und DML-Abfragen der Shard-Schlüssel als konstantes Prädikat angehängt wird. Diese Funktion ist nützlich, wenn Sie eine Migration zu Aurora PostgreSQL Limitless Database durchgeführt und das Schema durch Hinzufügen von Shard-Schlüsseln zu Tabellen denormalisiert haben.

Sie können ein Shard-Schlüsselprädikat automatisch an die bestehende SQL-Logik anhängen, ohne die Semantik der Abfragen zu ändern. Das Anhängen eines aktiven Shard-Schlüsselprädikats erfolgt nur für kompatible Tabellen.

Das Feature Aktiver Shard-Schlüssel verwendet die Variable `rds_aurora.limitless_active_shard_key`, die die folgende Syntax hat:

```
SET [session | local] rds_aurora.limitless_active_shard_key = '{"col1_value",
 "col2_value", ...}';
```

Einige Überlegungen zu aktiven Shard-Schlüsseln und Fremdschlüsseln:

- Für eine Sharded-Tabelle kann eine Fremdschlüsselbeschränkung gelten, wenn die übergeordneten und untergeordneten Tabellen nebeneinander angeordnet sind und der Fremdschlüssel eine Obermenge des Shard-Schlüssels bildet.
- Für eine Sharded-Tabelle kann eine Fremdschlüsselbeschränkung hinsichtlich einer Referenztabelle gelten.
- Für eine Referenztabelle kann eine Fremdschlüsselbeschränkung hinsichtlich einer anderen Referenztabelle gelten.

Nehmen wir an, wir haben eine `customers`-Tabelle mit angewendetem Sharding in der Spalte `customer_id`.

```
BEGIN;
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE customers(customer_id int PRIMARY KEY, name text , email text);
```

```
COMMIT;
```

Bei einem aktiven Shard-Schlüsselsatz weisen Abfragen die folgenden Transformationen auf.

SELECT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
SELECT * FROM customers;

-- This statement is changed to:
SELECT * FROM customers WHERE customer_id = '123'::int;
```

INSERT

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
INSERT INTO customers(name, email) VALUES('Alex', 'alex@example.com');

-- This statement is changed to:
INSERT INTO customers(customer_id, name, email) VALUES('123'::int, 'Alex',
 'alex@example.com');
```

AKTUALISIERUNG

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
UPDATE customers SET email = 'alex_new_email@example.com';

-- This statement is changed to:
UPDATE customers SET email = 'alex_new_email@example.com' WHERE customer_id =
 '123'::int;
```

DELETE

```
SET rds_aurora.limitless_active_shard_key = '{"123"}';
DELETE FROM customers;

-- This statement is changed to:
DELETE FROM customers WHERE customer_id = '123'::int;
```

Joins

Wenn an Tabellen mit einem aktiven Shard-Schlüssel Join-Operationen ausgeführt werden, wird das Shard-Schlüssel-Prädikat automatisch allen an der Verknüpfung beteiligten Tabellen

hinzugefügt. Dieses automatische Hinzufügen des Shard-Schlüssel-Prädikats erfolgt nur, wenn alle Tabellen in der Abfrage derselben Kollokationsgruppe angehören. Wenn die Abfrage Tabellen aus verschiedenen Kollokationsgruppen umfasst, wird stattdessen ein Fehler ausgelöst.

Gehen Sie davon aus, dass wir auch über `orders`- und `order_details`-Tabellen verfügen, die mit der `customers`-Tabelle zusammengefasst sind.

```
SET local rds_aurora.limitless_create_table_mode='sharded';
SET local rds_aurora.limitless_create_table_collocate_with='customers';
SET local rds_aurora.limitless_create_table_shard_key='{"customer_id"}';
CREATE TABLE orders (id int , customer_id int, total_amount int, date date);
CREATE TABLE order_details (id int , order_id int, customer_id int, product_name
 VARCHAR(100), price int);
COMMIT;
```

Hier rufen wir die letzten 10 Bestellrechnungen für einen Kunden ab, dessen Kundennummer 10 ist.

```
SET rds_aurora.limitless_active_shard_key = '{"10"}';
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    order_details.order_id = orders.order_id AND
    customers.customer_id = 10
    order by order_date limit 10;
```

Diese Abfrage wird in Folgende transformiert:

```
SELECT * FROM customers, orders, order_details WHERE
    orders.customer_id = customers.customer_id AND
    orders.order_id = order_details.order_id AND
    customers.customer_id = 10 AND
    order_details.customer_id = 10 AND
    orders.customer_id = 10 AND
    ORDER BY "order_date" LIMIT 10;
```

Tabellen, die mit aktiven Shard-Schlüsseln kompatibel sind

Das Shard-Schlüssel-Prädikat wird nur zu Tabellen hinzugefügt, die mit dem aktiven Shard-Schlüssel kompatibel sind. Eine Tabelle gilt als kompatibel, wenn ihr Shard-Schlüssel genau die Anzahl von Spalten enthält, die in der Variablen `rds_aurora.limitless_active_shard_key` angegeben ist. Wenn die Abfrage Tabellen

umfasst, die nicht mit dem aktiven Shard-Schlüssel kompatibel sind, gibt das System einen Fehler aus, anstatt mit der Abfrage fortzufahren.

Zum Beispiel:

```
-- Compatible table
SET rds_aurora.limitless_active_shard_key = '{"10"}';

-- The following query works because the customers table is sharded on one column.
SELECT * FROM customers;

-- Incompatible table
SET rds_aurora.limitless_active_shard_key = '{"10","20"}';

-- The following query raises a error because the customers table isn't sharded on
 two columns.
 SELECT * FROM customers;
```

Verteilte Abfragen in Aurora PostgreSQL Limitless Database

Verteilte Abfragen werden auf einem Router und mehr als einem Shard ausgeführt. Die Anfrage wird von einem der Router empfangen. Der Router erstellt und verwaltet die verteilte Transaktion, die dann an die teilnehmenden Shards gesendet wird. Die Shards erstellen eine lokale Transaktion mit dem vom Router bereitgestellten Kontext, und die Abfrage wird ausgeführt.

Wenn die Transaktion übermittelt wird, verwendet der Router bei Bedarf ein optimiertes zweiphasiges Commit-Protokoll und einen zeitbasierten MVCC-Mechanismus (Multi Version Concurrency Control), um die ACID-Semantik in einem verteilten Datenbanksystem bereitzustellen.

Zeitbasiertes MVCC zeichnet den Commit-Zeitpunkt für jede Transaktion auf und verwendet die Startzeit der Transaktion, um den Daten-Snapshot-Zeitpunkt zu generieren. Um anhand eines Reader-Snapshots festzustellen, ob eine Transaktion übermittelt (sichtbar) ist, vergleicht die Datenbank den Commit-Zeitpunkt der Transaktion mit dem Snapshot-Zeitpunkt. Wenn der Commit-Zeitpunkt vor dem Reader-Snapshot-Zeitpunkt liegt, ist die Transaktion sichtbar, andernfalls ist sie unsichtbar. Im Rahmen dieses Protokolls können Sie in der Aurora PostgreSQL Limitless Database immer hoch konsistente Daten erwarten.

Verteiltes Abfrage-Tracing in PostgreSQL-Protokollen in Aurora PostgreSQL Limitless Database

Verteiltes Abfrage-Tracing ist ein Tool zum Verfolgen und Korrelieren von Abfragen in PostgreSQL-Protokollen in Aurora PostgreSQL Limitless Database. In Aurora PostgreSQL verwenden Sie die Transaktions-ID, um eine Transaktion zu identifizieren. In Aurora PostgreSQL Limitless Database kann sich eine Transaktions-ID jedoch auf verschiedenen Routern wiederholen. Daher empfehlen wir, dass Sie in Limitless Database stattdessen die Tracing-ID verwenden.

Wichtige Anwendungsfälle:

- Verwenden Sie die Funktion `rds_aurora.limitless_get_last_trace_id()`, um die eindeutige Tracing-ID der letzten Abfrage in der aktuellen Sitzung zu ermitteln. Durchsuchen Sie dann die DB-Cluster-Protokollgruppe in Amazon CloudWatch Logs mit dieser Tracing-ID, um alle zugehörigen Protokolle zu finden.

  Sie können die Parameter `log_min_messages` und `log_min_error_statement` zusammen verwenden, um die Menge der abgedruckten Protokolle zu steuern und eine Anweisung abzudrucken, die die Tracing-ID enthält.
- Verwenden Sie den Parameter `log_min_duration_statement`, um die Laufzeit zu bestimmen, bei deren Überschreitung alle Abfragen ihre Ausführungsdauer und die Tracing-ID abdrucken. Diese Laufzeit kann dann in der Protokollgruppe des DB-Clusters in CloudWatch-Protokollen gesucht werden, um Knoten mit Engpässen zu ermitteln und Optimierungsmaßnahmen zu planen.

  Der Parameter `log_min_duration_statement` aktiviert die Tracing-ID für alle Knoten, unabhängig von den Werten der Parameter `log_min_messages` und `log_min_error_statement`.

Themen

- Tracing-ID
- Verwenden des Abfrage-Tracings
- Beispiele für Protokolle

Tracing-ID

Im Mittelpunkt dieser Funktion steht eine eindeutige Kennung, die sogenannte Tracing-ID. Die Tracing-ID ist eine 31-stellige Zeichenfolge, die an die STATEMENT-Protokollzeilen von PostgreSQL-

Protokollen angehängt wird und als genaue Kennung für korrelierende Protokolle dient, die sich auf eine bestimmte Abfrage beziehen. Beispiele sind `1126253375719408504000000000011` und `1126253375719408495000000000090`.

Die Tracing-ID besteht aus den folgenden Elementen:

- Transaktions-ID: Die ersten 20 Zeichen, die die Transaktion eindeutig identifizieren.
- Befehls-ID: Die ersten 30 Zeichen, die auf eine einzelne Abfrage innerhalb einer Transaktion hinweisen.

  Wenn innerhalb eines expliziten Transaktionsblocks mehr als 4 294 967 294 Abfragen ausgeführt werden, wird die Befehls-Id zu 1. In diesem Fall werden Sie durch die folgende `LOG`-Meldung im PostgreSQL-Protokoll benachrichtigt:

  ```
  wrapping around the tracing ID back to 1 after running 4294967294 (4.2 billion or
   2^32-2) queries inside of an explicit transaction block
  ```

- Knotentyp-ID: Die letzte Ziffer, die angibt, ob der Knoten als Koordinator-Router (1) oder als Teilnehmerknoten (`0`) fungiert.

Die folgenden Beispiele veranschaulichen die Komponenten der Tracing-ID:

- `1126253375719408504000000000011`:
  - Transaktions-ID: `11262533757194085 04`
  - Befehls-ID: `112625337571940850400000000001` gibt den ersten Befehl im Transaktionsblock an.
  - Knotentyp-ID: 1 gibt einen Koordinator-Router an.
- `1126253375719408495000000000090`:
  - Transaktions-ID: `11262533757194084 95`
  - Befehls-ID: `112625337571940849500000000009` gibt den neunten Befehl im Transaktionsblock an.
  - Knotentyp-ID: `0` gibt einen Teilnehmerknoten an.

Verwenden des Abfrage-Tracings

Führen Sie die folgenden Aufgaben aus, um das Abfrage-Tracing zu verwenden:

1. Stellen Sie sicher, dass Tracing aktiviert ist.

   Sie können zur Überprüfung den folgenden Befehl verwenden:

   ```
   SHOW rds_aurora.limitless_log_distributed_trace_id;
   ```

   Die Funktion ist standardmäßig aktiviert (on). Wenn sie nicht aktiviert ist, aktivieren Sie sie mit dem folgenden Befehl:

   ```
   SET rds_aurora.limitless_log_distributed_trace_id = on;
   ```

2. Steuern Sie die Menge der abgedruckten Protokolle, indem Sie den Schweregrad des Protokolls konfigurieren.

   Die Menge der Protokolle wird durch den Parameter `log_min_messages` gesteuert. Der Parameter `log_min_error_statement` wird verwendet, um die `STATEMENT`-Zeile mit der Tracing-ID abzudrucken. Beide sind standardmäßig auf `ERROR` eingestellt. Sie können zur Überprüfung folgende Befehle verwenden:

   ```
   SHOW log_min_messages;
   SHOW log_min_error_statement;
   ```

   Um den Schweregrad zu aktualisieren und die Zeile `STATEMENT` für die aktuelle Sitzung abzudrucken, verwenden Sie die folgenden Befehle mit einem dieser Schweregrade:

   ```
   SET log_min_messages = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO | NOTICE
    | WARNING | ERROR | LOG | FATAL | PANIC';
   SET log_min_error_statement = 'DEBUG5 | DEBUG4 | DEBUG3 | DEBUG2 | DEBUG1 | INFO
    | NOTICE | WARNING | ERROR | LOG | FATAL | PANIC';
   ```

   Zum Beispiel:

   ```
   SET log_min_messages = 'WARNING';
   SET log_min_error_statement = 'WARNING';
   ```

3. Aktivieren Sie das Abdrucken der Tracing-ID in den Protokollen ab einer bestimmten Laufzeit.

   Der Parameter `log_min_duration_statement` kann in die minimale Abfragelaufzeit geändert werden, bei deren Überschreitung die Abfrage für den gesamten DB-Cluster eine Protokollzeile

mit der Ausführungsdauer zusammen mit den Tracing-IDs ausgibt. Dieser Parameter ist standardmäßig auf `-1` gesetzt, was bedeutet, dass er deaktiviert ist. Sie können zur Überprüfung den folgenden Befehl verwenden:

```
SHOW log_min_duration_statement;
```

Wenn Sie ihn in `0` ändern, werden die Dauer und die Tracing-ID in den Protokollen für jede Abfrage im DB-Cluster abgedruckt. Sie können ihn mit dem folgenden Befehl für die aktuelle Sitzung auf `0` festlegen:

```
SET log_min_duration_statement = 0;
```

4. Rufen Sie die Tracing-ID ab.

   Rufen Sie nach dem Ausführen einer Abfrage (auch innerhalb eines expliziten Transaktionsblocks) die Funktion `rds_aurora.limitless_get_last_trace_id` auf, um die Tracing-ID des letzten Abfragelaufs abzurufen:

   ```
   SELECT * FROM rds_aurora.limitless_get_last_trace_id();
   ```

   Diese Funktion gibt die Transaktions-ID und die Befehls-ID zurück. Sie gibt die Knotentyp-ID nicht zurück.

   ```
   => SELECT * FROM customers;
    customer_id | fname | lname
   -------------+-------+-------
   (0 rows)

   => SELECT * FROM rds_aurora.limitless_get_last_trace_id();
    transaction_identifier |       command_identifier
   ------------------------+--------------------------------
    10104661421959001813   | 10104661421959001813000000001
   (1 row)
   ```

   Die Funktion gibt für nicht verteilte Abfragen eine Leerzeile zurück, da sie über keine Tracing-ID verfügen.

   ```
   => SET search_path = public;
   SET
   ```

```
=> SELECT * FROM rds_aurora.limitless_get_last_trace_id();
 transaction_identifier | command_identifier
------------------------+--------------------
                        |
(1 row)
```

> **Note**
>
> Bei VACUUM- und ANALYZE-Abfragen wird die Anweisung zur Dauer nicht mit der Tracing-ID protokolliert. Daher gibt `limitless_get_last_trace_id()` die Tracing-ID nicht zurück. Wenn es sich bei VACUUM oder ANALYZE um einen Vorgang mit langer Laufzeit handelt, können Sie die folgende Abfrage verwenden, um die Tracing-ID für diesen Vorgang abzurufen:
>
> ```
> SELECT * FROM rds_aurora.limitless_stat_activity
> WHERE distributed_tracing_id IS NOT NULL;
> ```
>
> Wenn der Server stoppt, bevor Sie die letzte Tracing-ID ausfindig machen können, müssen Sie die PostgreSQL-Protokolle manuell durchsuchen, um die Tracing-IDs in den Protokollen vom Zeitpunkt unmittelbar vor dem Ausfall zu finden.

5. Suchen Sie mit CloudWatch in den DB-Cluster-Protokollen nach der Tracing-ID.

   Verwenden Sie CloudWatch Insights, um die Protokollgruppe des DB-Clusters abzufragen, wie in den folgenden Beispielen gezeigt.

   - Abfrage nach einer bestimmten Transaktions-ID und alle Befehle werden darin ausgeführt:

     ```
     fields @timestamp, @message
     | filter @message like /10104661421959001813/
     | sort @timestamp desc
     ```

   - Abfrage nach einer bestimmten Befehls-ID:

     ```
     fields @timestamp, @message
     | filter @message like /10104661421959001813000000001/
     | sort @timestamp desc
     ```

6. Untersuchen Sie alle Protokolle im DB-Cluster, die durch die verteilte Abfrage erstellt wurden.

Beispiele für Protokolle

Die folgenden Beispiele zeigen die Verwendung von Abfrage-Tracing.

Korrelierung von Protokollen für fehleranfällige Abfragen

In diesem Beispiel wird der Befehl `TRUNCATE` an der Tabelle `customers` ausgeführt, wenn diese Tabelle nicht existiert.

Ohne Abfrage-Tracing

PostgreSQL-Protokolldatei auf dem Koordinator-Router:

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:CONTEXT: remote
 SQL command: truncate public.customers;
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[27503]:STATEMENT:
 truncate customers;
```

PostgreSQL-Protokolldatei auf einem Teilnehmer-Shard:

```
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]: postgres@postgres_limitless:[ 27503]:STATEMENT:
 truncate customers;
```

Diese Protokolle sind typisch. Ihnen fehlen die exakten IDs, die für die einfache Korrelation von Abfragen im gesamten DB-Cluster erforderlich sind.

Mit Abfrage-Tracing

PostgreSQL-Protokolldatei auf dem Koordinator-Router:

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR: failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:DETAIL: relation
 "public.customers" does not exist
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:CONTEXT: remote SQL
 command: truncate public.customers;
```

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT: /* tid =
 11262533757194085027000000000011 */ truncate customers;
```

PostgreSQL-Protokolldatei auf einem Teilnehmer-Shard:

```
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:ERROR:  failed to
 execute remote query
2023-09-26 04:03:19 UTC:[local]:postgres@postgres_limitless:[27503]:STATEMENT:  /* tid
 = 11262533757194085027000000000010 */ truncate customers;
```

Beim Abfrage-Tracing wird an jede Protokollzeile eine 31-stellige eindeutige Kennung angehängt. Hier stellen `11262533757194085027000000000011` und `11262533757194085027000000000010` jeweils die Tracing-IDs für den Koordinator- und die Teilnehmerknoten dar.

- Transaktions-ID: `11262533757194085027`
- Befehls-ID: `1126253375719408502700000000001`
- Knotentyp-ID: Die letzte Ziffer, `1` oder `0`, steht für einen Koordinator-Router bzw. einen Teilnehmerknoten.

Korrelieren von Protokollen, um die Abfragelaufzeit auf verschiedenen Knoten zu ermitteln

In diesem Beispiel wurde der Parameter `log_min_duration_statement` in `0` geändert, damit für alle Abfragen die Dauer abgedruckt wird.

Ohne Abfrage-Tracing

```
2024-01-15 07:28:46 UTC:[local]:postgres@postgres_limitless:[178322]:LOG: duration:
 12.779 ms statement: select * from customers;
```

Mit Abfrage-Tracing

PostgreSQL-Protokolldatei auf dem Koordinator-Router:

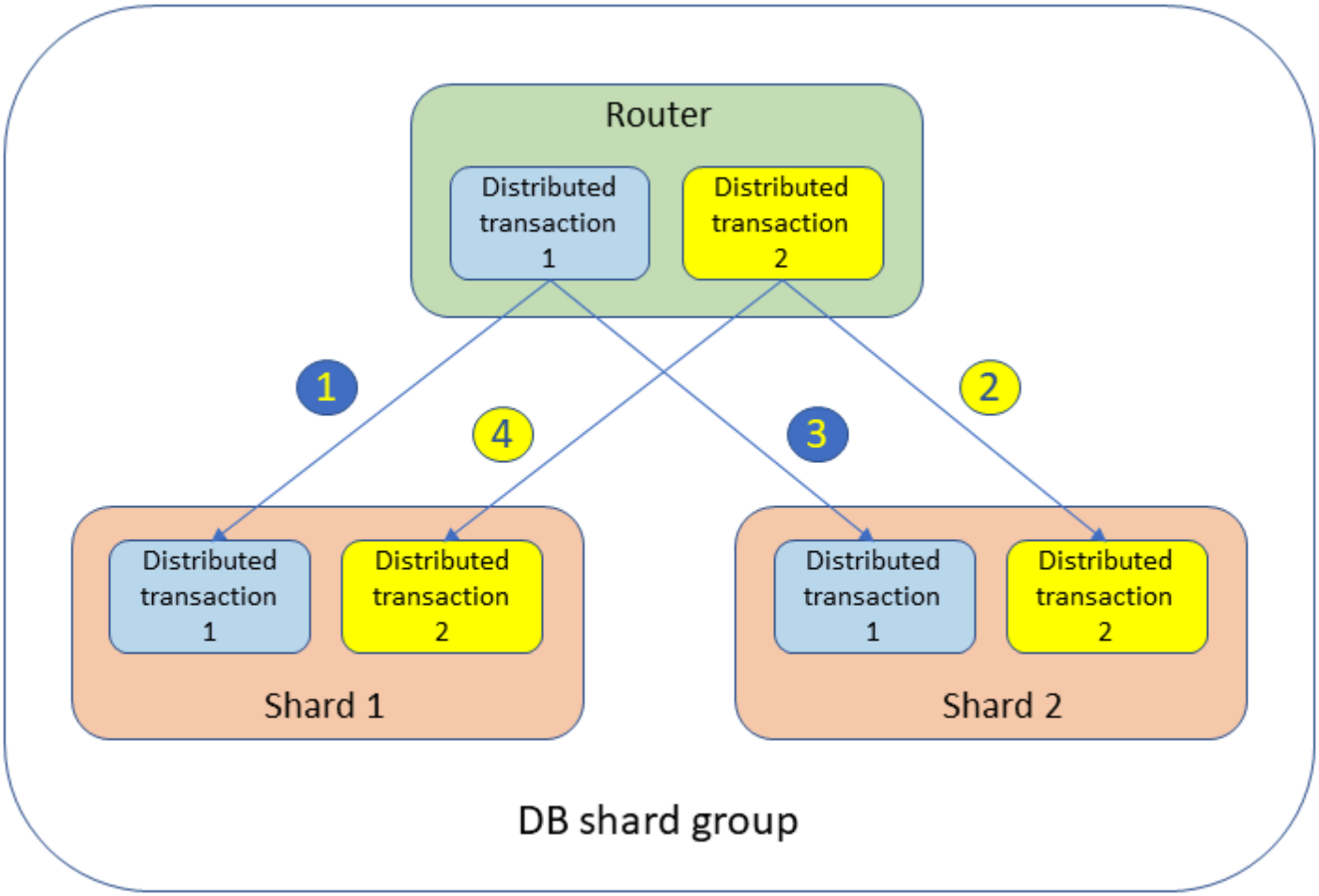
```
2024-01-15 07:32:08 UTC:[local]:postgres@postgres_limitless:[183877]:LOG: duration:
 12.618 ms statement: /* tid = 04576695662404970884000000000011 */ select * from
 customers;
```

PostgreSQL-Protokolldatei auf einem Teilnehmer-Shard:

```
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.279 ms statement: /* tid = 045766956624049708840000000010 */ START
 TRANSACTION ISOLATION LEVEL READ COMMITTED
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.249 ms parse <unnamed>: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.398 ms bind <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.019 ms execute <unnamed>/c1: SELECT customer_id, fname, lname FROM
 public.customers
2024-01-15 07:32:08 UTC:localhost(46358):postgres@postgres_limitless:[183944]:LOG:
 duration: 0.073 ms statement: /* tid = 045766956624049708840000000010 */ COMMIT
 TRANSACTION
```

Verteilte Deadlocks in Aurora PostgreSQL Limitless Database

In einer DB-Shard-Gruppe können Deadlocks zwischen Transaktionen auftreten, die auf verschiedene Router und Shards verteilt sind. Beispielsweise werden zwei gleichzeitige verteilte Transaktionen mit zwei Shards ausgeführt, wie in der folgenden Abbildung dargestellt.



Die Transaktionen sperren Tabellen und erzeugen Warteereignisse in den beiden Shards wie folgt:

1. Verteilte Transaktion 1:

   ```
   UPDATE table SET value = 1 WHERE key = 'shard1_key';
   ```

   Shard 1 bleibt gesperrt.

2. Verteilte Transaktion 2:

   ```
   UPDATE table SET value = 2 WHERE key = 'shard2_key';
   ```

Shard 2 bleibt gesperrt.

3. Verteilte Transaktion 1:

```
UPDATE table SET value = 3 WHERE key = 'shard2_key';
```

Die verteilte Transaktion 1 wartet auf Shard 2.

4. Verteilte Transaktion 2:

```
UPDATE table SET value = 4 WHERE key = 'shard1_key';
```

Die verteilte Transaktion 2 wartet auf Shard 1.

In diesem Szenario erkennen weder Shard 1 noch Shard 2 das Problem: Transaktion 1 wartet auf Shard 2 auf Transaktion 2 und Transaktion 2 wartet auf Shard 1 auf Transaktion 1. Aus globaler Sicht wartet Transaktion 1 auf Transaktion 2 und Transaktion 2 wartet auf Transaktion 1. Diese Situation, in der zwei Transaktionen auf zwei verschiedenen Shards aufeinander warten, wird als verteilter Deadlock bezeichnet.

Aurora PostgreSQL Limitless Database kann verteilte Deadlocks automatisch erkennen und lösen. Ein Router in der DB-Shard-Gruppe wird benachrichtigt, wenn eine Transaktion zu lange auf den Erwerb einer Ressource wartet. Der Router, der die Benachrichtigung empfängt, beginnt, die erforderlichen Informationen von allen Routern und Shards innerhalb der DB-Shard-Gruppe zu sammeln. Der Router beendet daraufhin Transaktionen, die an einem verteilten Deadlock beteiligt sind, bis die restlichen Transaktionen in der DB-Shard-Gruppe weiter ausgeführt werden können, ohne sich gegenseitig zu blockieren.

Sie erhalten die folgende Fehlermeldung, wenn Ihre Transaktion Teil eines verteilten Deadlocks war und dann vom Router beendet wurde:

```
ERROR: aborting transaction participating in a distributed deadlock
```

Der DB-Cluster-Parameter `rds_aurora.limitless_distributed_deadlock_timeout` legt fest, wie lange jede Transaktion auf einer Ressource wartet, bevor der Router aufgefordert wird, nach einem verteilten Deadlock zu suchen. Sie können den Parameterwert erhöhen, wenn Ihre Workload weniger anfällig für Deadlock-Situationen ist. Der Standardwert ist `1000` Millisekunden (1 Sekunde).

Der Zyklus eines verteilten Deadlocks wird in den PostgreSQL-Protokollen veröffentlicht, wenn ein knotenübergreifender Deadlock gefunden und behoben wurde. Zu den Informationen über jeden Prozess, der Teil des Deadlocks ist, gehören die folgenden:

- Koordinatorknoten, der die Transaktion gestartet hat
- ID der virtuellen Transaktions (xid) auf dem Koordinatorknoten im Format *backend_id*/*backend_local_xid*
- ID der verteilten Sitzung der Transaktion

Verwalten einer Aurora PostgreSQL Limitless Database

In den folgenden Themen wird beschrieben, wie Sie die DB-Cluster Ihrer Aurora PostgreSQL Limitless Database verwalten.

Themen

- Überlegungen zur Datenbank- und Tabellengröße
- Zurückgewinnen von Speicherplatz durch Bereinigung

Überlegungen zur Datenbank- und Tabellengröße

Für Aurora PostgreSQL Limitless Database ist auf jedem Shard eine Sharded-Tabelle in eine Anzahl von Tabellensegmenten unterteilt. Diese Anzahl variiert, je nachdem, wie viele Shards in der DB-Shard-Gruppe verfügbar sind. Jeder Tabellenbereich kann bis zu 32 TiB groß werden, aber jeder Shard hat eine maximale Kapazität von 128 TiB. Referenztabellen haben eine Größenbeschränkung von 32 TiB für die gesamte DB-Shard-Gruppe.

> **Note**
>
> Die maximale Kapazität jedes Knotens (Router oder Shard) beträgt 128 TiB, da dies die maximale Kapazität für einen DB-Cluster von Aurora PostgreSQL ist.

Die maximale Anzahl von Beziehungen pro Datenbank (einschließlich Tabellen, Ansichten und Indizes) in Aurora PostgreSQL und Aurora PostgreSQL Limitless Database beträgt 1 431 650 303.

Weitere Informationen finden Sie in Appendix K. – PostgreSQL-Limits in der PostgreSQL-Dokumentation und in Amazon Aurora-Größenbeschränkungen.

Zurückgewinnen von Speicherplatz durch Bereinigung

PostgreSQL Multiversion Concurrency Control (MVCC) trägt zur Wahrung der Datenintegrität bei, indem es eine interne Kopie aktualisierter oder gelöschter Zeilen speichert, bis eine Transaktion entweder festgeschrieben oder zurückgesetzt wird. Diese Kopien, auch Tupel genannt, können zu einer Überlastung der Tabelle führen, wenn sie nicht regelmäßig bereinigt werden. PostgreSQL-Instances ordnen Transaktionen nach ihren Transaktions-IDs, und PostgreSQL verwendet MVCC, das auf Transaktions-IDs basiert, um die Sichtbarkeit von Tupeln zu kontrollieren und die Transaktionsisolierung zu gewährleisten. Jede Transaktion erstellt einen Daten-Snapshot, und jedes Tupel hat eine Version. Sowohl der Snapshot als auch die Version basieren auf der Transaktions-ID.

Um Daten zu bereinigen, führt das Dienstprogramm `VACUUM` vier wichtige Funktionen in PostgreSQL aus:

- `VACUUM` – Entfernt abgelaufene Zeilenversionen, wodurch Speicherplatz für die Wiederverwendung verfügbar wird.
- `VACUUM FULL` – Ermöglicht eine vollständige Defragmentierung, indem veraltete Versionen entfernt und die Tabellen komprimiert werden, wodurch die Größe reduziert und die Effizienz gesteigert wird.
- `VACUUM FREEZE` – Schützt vor Wraparound-Problemen bei Transaktions-IDs, indem ältere Zeilenversionen als gesperrt markiert werden.
- `VACUUM ANALYZE` – Entfernt veraltete Zeilenversionen und aktualisiert die Abfrageplanungsstatistiken der Datenbank. Es ist eine Kombination aus `VACUUM`- und `ANALYZE`-Funktionen. Weitere Informationen zur Funktionsweise von `ANALYZE` in Aurora PostgreSQL Limitless Database finden Sie unter ANALYZE.

Wie bei MVCC basiert die Bereinigung in Aurora PostgreSQL auf der Transaktions-ID. Wenn zu Beginn der Bereinigung eine Transaktion läuft, werden Zeilen, die für diese Transaktion noch sichtbar sind, nicht entfernt.

Weitere Informationen zum Dienstprogramm `VACUUM` finden Sie unter VACUUM in der PostgreSQL-Dokumentation. Weitere Informationen zur Unterstützung von `VACUUM` in Aurora PostgreSQL Limitless Database finden Sie unter VACUUM.

Themen

- AUTOVACUUM
- Zeitbasiertes Bereinigen in Aurora PostgreSQL Limitless Database

- Verwenden von Datenbankstatistiken für die Bereinigung
- Unterschiede im Bereinigungsverhalten zwischen Aurora PostgreSQL und Aurora PostgreSQL Limitless Database

AUTOVACUUM

Aurora PostgreSQL verwendet die Dienstprogramme `VACUUM` und `AUTOVACUUM`, um nicht benötigte Tupel zu entfernen. Der zugrundeliegende Mechanismus für `AUTOVACUUM` und manuelles `VACUUM` ist derselbe, der einzige Unterschied besteht in der Automatisierung.

`AUTOVACUUM` in Aurora PostgreSQL und Aurora PostgreSQL Limitless Database ist eine Kombination der Dienstprogramme `VACUUM` und `ANALYZE`. `AUTOVACUUM` legt gemäß einer vordefinierten Regel fest, welche Datenbanken und Tabellen bereinigt werden sollen, z. B. anhand des Prozentsatzes inaktiver Tupel und der Anzahl der Einfügungen.

`AUTOVACUUM` wird beispielsweise regelmäßig aktiviert, um eine Bereinigung durchzuführen. Das Intervall wird durch den Parameter `autovacuum_naptime` gesteuert. Der Standardwert beträgt 1 Minute. Die Standardwerte für die Konfigurationsparameter `AUTOVACUUM` und `VACUUM` sind für Aurora PostgreSQL Limitless Database dieselben wie für Aurora PostgreSQL.

Wenn der Daemon für `AUTOVACUUM` aktiviert ist, gibt er automatisch `ANALYZE`-Befehle aus, wenn sich der Inhalt einer Tabelle ausreichend geändert hat. In Aurora PostgreSQL Limitless Database gibt `AUTOVACUUM ANALYZE` auf Routern und auf Shards aus.

Weitere Informationen zum Daemon für `AUTOVACUUM` und zu den mit `AUTOVACUUM` verbundenen Tabellenspeicherparametern finden Sie unter Der Bereinigungsdaemon und die Speicherparameter in der PostgreSQL-Dokumentation.

Zeitbasiertes Bereinigen in Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database ist ein verteiltes System, was bedeutet, dass mehrere Instances an einer Transaktion beteiligt sein können. Daher gilt die auf der Transaktions-ID basierende Transparenz nicht. Stattdessen verwendet Aurora PostgreSQL Limitless Database zeitbasierte Sichtbarkeit, da Transaktions-IDs nicht Instance-übergreifend „vereinheitlicht“ werden, die Zeit hingegen schon. Jeder Transaktions-Snapshot und jede Tupel-Version folgen der Zeit und nicht der Transaktions-ID. Genauer gesagt hat ein Transaktions-Snapshot eine Snapshot-Startzeit, und ein Tupel hat eine Erstellungszeit (bei einem `INSERT`- oder `UPDATE`-Vorgang) und eine Löschzeit (bei einem `DELETE`-Vorgang).

Um die Datenkonsistenz zwischen den Instances in der DB-Shard-Gruppe aufrechtzuerhalten, muss Aurora PostgreSQL Limitless Database sicherstellen, dass beim Bereinigen keine Tupel entfernt werden, die für aktive Transaktionen in der DB-Shard-Gruppe noch sichtbar sind. Daher ist die Bereinigung in Aurora PostgreSQL Limitless Database auch zeitbasiert. Andere Aspekte von `VACUUM` bleiben gleich, einschließlich der Tatsache, dass ein Benutzer Zugriff auf diese Tabelle haben muss, um `VACUUM` auf einer bestimmten Tabelle auszuführen.

> **Note**
>
> Es wird dringend davon abgeraten, Transaktionen für längere Zeit offen zu lassen. Die zeitbasierte Bereinigung verbraucht mehr Speicher als die auf der Transaktions-ID basierte Bereinigung.

Das folgende Beispiel veranschaulicht die Funktionsweise von zeitbasierter Bereinigung.

1. Eine Kundentabelle ist auf vier Shards verteilt.
2. Transaktion 1 beginnt mit einem wiederholbaren Lesevorgang und zielt nur auf einen Shard (Shard 1) ab. Diese Transaktion bleibt offen.

   Transaktion 1 ist älter als jede andere Transaktion, die danach gestartet wurde.
3. Transaktion 2 beginnt später, löscht alle Tupel aus einer Tabelle und wird anschließend festgeschrieben.
4. Wenn `AUTOVACUUM` oder manuelles `VACUUM` versucht, inaktive Tupel zu bereinigen (die aufgrund von Transaktion 2 inaktiv sind), wird nichts entfernt.

   Dies gilt nicht nur für Shard 1, sondern auch für die Shards 2 bis 4, da Transaktion 1 möglicherweise immer noch auf diese Tupel zugreifen muss. Sie sind aufgrund von MVCC immer noch für Transaktion 1 sichtbar.

Der letzte Schritt wird durch Synchronisierung erreicht, sodass alle Shards über Transaktion 1 informiert sind, auch wenn Transaktion 1 nicht alle Shards betrifft.

Verwenden von Datenbankstatistiken für die Bereinigung

Um Informationen über Tupel zu erhalten, die Sie möglicherweise bereinigen müssen, verwenden Sie die Ansicht limitless_stat_all_tables, die ähnlich wie pg_stat_all_tables funktioniert. Im folgenden Beispiel wird die Ansicht abgefragt.

```
SELECT * FROM rds_aurora.limitless_stat_all_tables WHERE relname LIKE '%customer%';
```

Entsprechend verwenden Sie für Datenbankstatistiken limitless_stat_database anstelle von pg_stat_database und limitless_stat_activity anstelle von pg_stat_activity.

Um die Festplattennutzung zu überprüfen, verwenden Sie die Funktion limitless_stat_relation_sizes, die ähnlich wie pg_relation_size funktioniert. Im folgenden Beispiel wird die Funktion abgefragt.

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customer');
```

Um den Fortschritt eines VACUUM-Vorgangs in der Aurora PostgreSQL Limitless Database zu verfolgen, verwenden Sie die Ansicht limitless_stat_progress_vacuum anstelle von pg_stat_progress_vacuum. Im folgenden Beispiel wird die Ansicht abgefragt.

```
SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;
```

Weitere Informationen erhalten Sie unter Ansichten für Aurora PostgreSQL Limitless Database und Funktionen für Aurora PostgreSQL Limitless Database.

Unterschiede im Bereinigungsverhalten zwischen Aurora PostgreSQL und Aurora PostgreSQL Limitless Database

Einige weitere Unterschiede zwischen Aurora PostgreSQL und Aurora PostgreSQL Limitless Database in Bezug auf die Funktionsweise der Bereinigung sind die folgenden:

- Aurora PostgreSQL führt VACUUM-Vorgänge mit Transaktions-IDs bis zur ältesten laufenden Transaktion durch. Wenn in der Datenbank keine laufende Transaktion vorhanden ist, führt VACUUM den Vorgang bis zur letzten Transaktion ausgeführt.
- Aurora PostgreSQL Limitless Database synchronisiert den ältesten Zeit-Snapshot alle 10 Sekunden. Daher wird VACUUM den Vorgang möglicherweise nicht für Transaktionen durch, die innerhalb der letzten 10 Sekunden ausgeführt wurden.

Weitere Informationen zur Unterstützung für VACUUM in Aurora PostgreSQL Limitless Database finden Sie unter VACUUM in der Referenz zu Aurora PostgreSQL Limitless Database.

Überwachen einer Aurora PostgreSQL Limitless Database

Sie können Amazon CloudWatch, Verbesserte Überwachung und Performance Insights verwenden, um Aurora PostgreSQL Limitless Database zu überwachen. Es gibt auch neue Statistikfunktionen und -ansichten sowie Warteereignisse für Aurora PostgreSQL Limitless Database, die Sie für die Überwachung und Diagnose verwenden können.

Themen

- Überwachen von Aurora PostgreSQL Limitless Database mit Amazon CloudWatch
- Überwachen von Aurora PostgreSQL Limitless Database mit CloudWatch Database Insights
- Überwachen von Aurora PostgreSQL Limitless-Datenbank mit Amazon CloudWatch Logs
- Überwachen einer Aurora PostgreSQL Limitless Database mithilfe der erweiterten Überwachung
- Überwachen einer Aurora PostgreSQL Limitless Database mithilfe von Performance Insights
- Überwachen einer Aurora PostgreSQL Limitless Database mithilfe des RDS-Schutzes von Amazon GuardDuty
- Funktionen und Ansichten für Aurora PostgreSQL Limitless Database
- Warteereignisse für Aurora PostgreSQL Limitless Database

Überwachen von Aurora PostgreSQL Limitless Database mit Amazon CloudWatch

CloudWatch-Metriken für Aurora PostgreSQL Limitless Database werden unter den folgenden Dimensionen gemeldet:

- DBShardGroup
- DBShardGroupRouterAggregation
- DBShardGroupInstance
- DBClusterIdentifier

Weitere Informationen zu CloudWatch-Metriken finden Sie unter Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch.

DBShardGroup-Metriken

Wenn Sie `DBShardGroup`-Metriken für Aurora PostgreSQL Limitless Database in der CloudWatch-Konsole anzeigen möchten, wählen Sie RDS und dann DBShardGroup aus.

Sie können die folgenden CloudWatch-Metriken verfolgen:

- `DBShardGroupACUUtilization`: Auslastung der Aurora-Kapazitätseinheiten (ACU) in Prozent, berechnet aus `DBShardGroupCapacity` geteilt durch `DBShardGroupMaxACU`.
- `DBShardGroupCapacity`: Anzahl der ACUs, die von den Writer-Instances der DB-Shard-Gruppe konsumiert werden.
- `DBShardGroupComputeRedundancyCapacity`: Anzahl der ACUs, die von den Standby-Instances der DB-Shard-Gruppe konsumiert werden.
- `DBShardGroupMaxACU`: Maximale Anzahl der ACUs, die für die DB-Shard-Gruppe konfiguriert sind.
- `DBShardGroupMinACU`: Mindestanzahl der ACUs, die von der DB-Shard-Gruppe benötigt wird.

Der Dimensionsschlüssel `DBShardGroupIdentifier` ist für die Aggregation der `DBShardGroup`-Metriken verfügbar.

DBShardGroupRouterAggregation-Metriken

Wenn Sie `DBShardGroupRouterAggregation`-Metriken für Aurora PostgreSQL Limitless Database in der CloudWatch-Konsole anzeigen möchten, wählen Sie RDS und dann DBShardGroupRouterAggregation aus.

Sie können die folgenden CloudWatch-Metriken verfolgen:

- `CommitThroughput`: Die durchschnittliche Anzahl von Commit-Vorgängen pro Sekunde auf allen Router-Knoten in der DB-Shard-Gruppe.
- `DatabaseConnections`: Die Summe aller Verbindungen zwischen allen Router-Knoten in der DB-Shard-Gruppe.

DBShardGroupInstance-Metriken

Eine DBShardGroupInstance ist die einzelne DB-Instance innerhalb jedes Shard- oder Router-Subclusters.

Wenn Sie `DBShardGroupInstance`-Metriken für Aurora PostgreSQL Limitless Database in der CloudWatch-Konsole anzeigen möchten, wählen Sie RDS und dann DBShardGroupInstance aus.

Sie können die folgenden CloudWatch-Metriken verfolgen:

- `ACUUtilization`: Der als `ServerlessDatabaseCapacity`-Metrik berechnete Prozentsatz geteilt durch den zugewiesenen maximalen ACU-Wert des Subclusters.
- `AuroraReplicaLag`: In Limitless-Clustern mit aktivierter Compute-Redundanz ist dies die Dauer der Verzögerung, wenn Updates aus einer primären Instance im Subcluster repliziert werden.
- `AuroraReplicaLagMaximum`: In Limitless-Clustern mit aktivierter Compute-Redundanz ist dies die maximale Verzögerung, wenn Updates aus einer primären Instance im Subcluster repliziert werden. Wenn Lesereplikate gelöscht oder umbenannt werden, könnte es zu einem vorübergehenden Anstieg der Replikationsverzögerung kommen, da die alte Ressource einem Recycling unterzogen wird. Verwenden Sie diese Metrik, um herauszufinden, ob auf einem der Lesegeräte ein Failover aufgrund einer hohen Replikationsverzögerung aufgetreten ist.
- `AuroraReplicaLagMinimum`: In Limitless-Clustern mit aktivierter Compute-Redundanz ist dies die minimale Verzögerung, wenn Updates aus einer primären Instance im Subcluster repliziert werden.
- `BufferCacheHitRatio`: Der Prozentsatz der Daten und Indizes, die aus dem Speichercache einer Instance bedient werden (im Gegensatz zum Speichervolume).

- `CommitLatency`: Die durchschnittliche Dauer, die Engine und Speicher benötigen, um die Commit-Operationen für einen bestimmten Knoten (Router oder Shard) abzuschließen.
- `CommitThroughput`: Die durchschnittliche Anzahl der Commit-Operationen pro Sekunde
- `CPUUtilization`: Die CPU-Auslastung in Prozent des maximal zugewiesenen ACU-Werts des Subclusters.
- `FreeableMemory`: Die Menge des nicht belegten Speichers, die verfügbar ist, wenn die Shard-Gruppe auf ihre maximale Kapazität skaliert wird. Dies wird durch die zugewiesenen ACUs der Shard-Gruppe bestimmt. Für jede ACU, bei der die aktuelle Kapazität unter der maximalen Kapazität liegt, erhöht sich dieser Wert ungefähr um 2 GiB. Daher nähert sich diese Metrik erst Null, wenn die DB-Shard-Gruppe bis zum Maximum hochskaliert ist.
- `MaximumUsedTransactionIDs`: Das Alter der ältesten nicht bereinigten Transaktions-ID in Transaktionen. Erreicht dieser Wert 2.146.483.648 (2^31 - 1 000 000), wird die Datenbank in den Nur-Lese-Modus gezwungen, um den Wraparound der Transaktions-ID zu vermeiden. Weitere Informationen finden Sie unter Transaktions-ID-Wraparound-Fehler vermeiden in der PostgreSQL-Dokumentation.
- `NetworkReceiveThroughput`: Die Menge des von Clients erhaltenen Netzwerkdurchsatzes für jede Instance in der DB-Shard-Gruppe. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances in der DB-Shard-Gruppe und dem Cluster-Volume.
- `NetworkThroughput`: Der aggregierte Netzwerkdurchsatz (sowohl übertragen als auch empfangen) zwischen Clients und Routern sowie Routern und Shards in der DB-Shard-Gruppe. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances in der DB-Shard-Gruppe und dem Cluster-Volume.
- `NetworkTransmitThroughput`: Die Menge des an Clients gesendeten Netzwerkdurchsatzes für jede Instance in der DB-Shard-Gruppe. Dieser Durchsatz beinhaltet nicht den Netzwerkdatenverkehr zwischen den Instances in der DB-Shard-Gruppe und dem Cluster-Volume.
- `ReadIOPS`: Die durchschnittliche Anzahl der Datenträger-Eingabe-/Ausgabe-Lesevorgänge pro Sekunde (IOPS).
- `ReadLatency`: Die durchschnittliche Dauer für einen Datenträger-Ein-/Ausgabe-(I/O)-Lesevorgang.
- `ReadThroughput`: Die durchschnittliche Anzahl Byte, die pro Sekunde vom Datenträger gelesen werden.
- `ServerlessDatabaseCapacity`: Die aktuelle Kapazität des DB-Shards oder Router-Subclusters innerhalb der DB-Shard-Gruppe.

- `StorageNetworkReceiveThroughput`: Der Umfang des vom Aurora-Speicheruntersystem empfangenen Netzwerkdurchsatzes für jede Instance in der DB-Shard-Gruppe.
- `StorageNetworkThroughput`: Der aggregierte Netzwerkdurchsatz, den jede Instance in der DB-Shard-Gruppe an das Aurora-Speichersubsystem sendet und vom Aurora-Speichersubsystem empfängt.
- `StorageNetworkTransmitThroughput`: Der Umfang des von jeder Instance in der DB-Shard-Gruppe an das Aurora-Speicheruntersystem gesendeten Netzwerkdurchsatzes.
- `SwapUsage`: Die Menge des von der DB-Shard-Gruppe verwendeten Auslagerungsbereichs.
- `TempStorageIOPS`: Die durchschnittliche Anzahl von I/O-Operationen, die im lokalen Speicher durchgeführt werden, der der DB-Instance angefügt ist. Sie umfasst sowohl Lese- als auch Schreib-I/O-Vorgänge.

  `TempStorageIOPS` kann zusammen mit `TempStorageThroughput` verwendet werden, um die seltenen Fälle zu diagnostizieren, in denen die Netzwerkaktivität für Übertragungen zwischen Ihren DB-Instances und lokalen Speichergeräten für unerwartete Kapazitätssteigerungen verantwortlich ist.
- `TempStorageThroughput`: Die Menge der mit einem Router oder einem Shard verknüpften Daten, die an den und aus dem lokalen Speicher übertragen wurden.
- `WriteIOPS`: Die durchschnittliche Anzahl der Datenträger-Schreib-IOPS.
- `WriteLatency`: Die durchschnittliche Dauer für einen Datenträger-Schreib-I/O-Vorgang.
- `WriteThroughput`. Die durchschnittliche Anzahl Byte, die pro Sekunde auf den Datenträger geschrieben werden.

Die folgenden Dimensionsschlüssel sind zum Aggregieren der `DBShardGroupInstance`-Metrik verfügbar.

- `DBClusterIdentifier`: Der DB-Cluster von Aurora PostgreSQL.
- `DBShardGroupIdentifier`: Die DB-Shard-Gruppe, zu der die Instance gehört.
- `DBShardGroupSubClusterType`: Der Knotentyp, entweder `Distributed Transaction Router` (Router) oder `Data Access Shard` (Shard).
- `DBShardGroupSubClusterIdentifier`: Der Name des Routers oder Shards, zu dem die Instance gehört.

Im Folgenden finden Sie Beispiele für das Aggregieren von CloudWatch-Metriken:

- Gesamt-`CPUUtilization` aller Instances, die zu einem bestimmten Shard oder Router in einer DB-Shard-Gruppe gehören.
- Gesamt-`CPUUtilization` aller Instances in einer DB-Shard-Gruppe.

DBClusterIdentifier-Metriken

Wenn Sie `DBClusterIdentifier`-Metriken für Aurora PostgreSQL Limitless Database in der CloudWatch-Konsole anzeigen möchten, wählen Sie RDS und dann DBClusterIdentifier aus.

Wenn Sie Aurora PostgreSQL Limitless Database verwenden, haben Sie möglicherweise mehr I/O-Vorgänge (Ein-/Ausgabe) als bei einem Aurora-DB-Cluster. Sie können die folgenden CloudWatch-Metriken für Ihren Limitless-Database-Cluster verfolgen:

- `VolumeReadIops` Die Anzahl von in Rechnung gestellten I/O-Vorgängen aus einem Cluster-Volume, gemeldet in 5-Minuten-Intervallen.
- `VolumeWriteIops`: Die Anzahl der Datenträger-I/O-Schreibvorgänge im Cluster-Volume, gemeldet in 5-Minuten-Intervallen.

Aurora PostgreSQL Limitless Database verwendet die Cluster-Speicherkonfiguration Aurora I/O-Optimized. Mit Aurora I/O-Optimized zahlen Sie einen einzigen monatlichen Preis für alle I/O-Vorgänge, anstatt für jede Million I/O-Anfragen. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.

Möglicherweise verwenden Sie auch mehr Speicher als für einen Aurora-DB-Cluster. Sie können die folgenden CloudWatch-Metriken hinsichtlich der Speichernutzung verfolgen:

- `BackupRetentionPeriodStorageUsed`: Die gesamte kontinuierliche Backup-Speicherauslastung Ihres Clusters von Aurora PostgreSQL Limitless Database, die in Rechnung gestellt wurde.
- `SnapshotStorageUsed`: Die gesamte Snapshot-Speicherauslastung Ihres Clusters von Aurora PostgreSQL Limitless Database, die in Rechnung gestellt wurde.
- `TotalBackupStorageBilled`: Die Summe Ihrer Kosten für automatische Backup-Aufbewahrung und DB-Cluster-Snapshots.

  Weitere Information zu Backup-Speicherkosten finden Sie unter Grundlegendes zur Backup-Speicher-Nutzung in Amazon Aurora.

- `VolumeBytesUsed`: Die vom Cluster Ihrer Aurora PostgreSQL Limitless Database verwendeten Speichermenge, gemeldet in 5-Minuten-Intervallen.

Überwachen von Aurora PostgreSQL Limitless Database mit CloudWatch Database Insights

Der Standardmodus von Database Insights ist für die Aktivierung von Aurora PostgreSQL Limitless Database erforderlich. Sie können damit die Datenbanklast (DB-Last) Ihrer DB-Instances von Limitless Database in Echtzeit überwachen. Die DB-Last misst den Sitzungsaktivitätsgrad in einer Datenbank. Sie können Database Insights verwenden, um die Leistung Ihrer DB-Instances von Aurora PostgreSQL Limitless Database in großem Umfang zu analysieren und Fehler zu beheben.

Weitere Informationen zu CloudWatch-Database-Metriken finden Sie im folgenden Abschnitt.

- Überwachen von Amazon-Aurora-Datenbanken mit CloudWatch Database Insights
- CloudWatch Database Insights im Amazon-CloudWatch-Benutzerhandbuch
- Erste Schritte mit CloudWatch Database Insights im Amazon-CloudWatch-Benutzerhandbuch
- Konfigurieren Ihrer Datenbank zur Überwachung langsamer SQL-Abfragen mit Database Insights für Amazon Aurora

Informationen zum Aktivieren des erweiterten Modus oder des Standardmodus von Database Insights finden Sie in den folgenden Themen.

Themen

- Aktivieren des erweiterten Modus von Database Insights für Aurora PostgreSQL Limitless Database
- Aktivieren des Standardmodus von Database Insights für Aurora PostgreSQL Limitless Database

Aktivieren des erweiterten Modus von Database Insights für Aurora PostgreSQL Limitless Database

Gehen Sie wie folgt vor, um den erweiterten Modus von Database Insights für Ihre Aurora PostgreSQL Limitless Database zu aktivieren.

Aktivieren des erweiterten Modus von Database Insights beim Erstellen eines DB-Clusters für Aurora PostgreSQL Limitless Database

Aktivieren Sie den erweiterten Modus von Database Insights, wenn Sie eine Datenbank für Aurora PostgreSQL Limitless Database erstellen.

Console

In der Konsole können Sie den erweiterten Modus von Database Insights aktivieren, wenn Sie einen DB-Cluster erstellen. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den erweiterten Modus von Database Insights, wenn Sie einen DB-Cluster über die Konsole erstellen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie Create database (Datenbank erstellen) aus.
4. Wählen Sie im Abschnitt Database Insights Erweiterter Modus aus. Wählen Sie dann eine der folgenden Optionen aus:
   - Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Der Aufbewahrungszeitraum für den erweiterten Modus von Database Insights muss 15-24 Monate betragen.
   - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
5. Wählen Sie Create database (Datenbank erstellen) aus.

AWS CLI

Um den erweiterten Modus von Database Insights beim Erstellen eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl create-db-cluster auf und geben Sie die folgenden Werte ein:

- `--db-cluster-identifier`: Die Kennung für den DB-Cluster.
- `--database-insights-mode advanced`: Um den erweiterten Modus von Database Insights zu aktivieren.
- `--engine`: Der DB-Cluster muss die `aurora-postgresql`-DB-Engine verwenden.
- `--engine-version`: Der DB-Cluster muss eine der DB-Engine-Versionen verwenden:
  - `16.4-limitless`

  - `16.6-limitless`
- `--storage-type`: Der DB-Cluster muss die Speicherkonfiguration des `aurora-iopt1`-DB-Clusters verwenden.
- `--cluster-scalability-type`: Gibt den Skalierbarkeitsmodus des Aurora-DB-Clusters an. Wenn dieser Parameter auf `limitless` gesetzt ist, arbeitet der Cluster als Aurora PostgreSQL Limitless Database. Wenn er auf `standard` (Standard) gesetzt ist, verwendet der Cluster die normale DB-Instance-Erstellung.

  > **Note**
  >
  > Sie können diese Einstellung nicht mehr ändern, nachdem Sie den DB-Cluster erstellt haben.

- `--master-username`: Der Name des Masterbenutzers für diesen DB-Cluster.
- `--master-user-password`: Das Passwort für den Masterbenutzer.
- `--enable-performance-insights`: Um Performance Insights für Database Insights zu aktivieren.
- `--performance-insights-retention-period`: Der Aufbewahrungszeitraum für Daten für Ihren DB-Cluster. Um Database Insights zu aktivieren, muss der Aufbewahrungszeitraum mindestens 465 Tage betragen.
- `--monitoring-interval`: Das Intervall in Sekunden zwischen den Punkten, an denen erweiterte Überwachungsmetriken für den DB-Cluster erfasst werden. Dieser Wert darf nicht `0` sein.
- `--monitoring-role-arn`: Der Amazon-Ressourcenname (ARN) für die IAM-Rolle, die es RDS erlaubt, erweiterte Überwachungsmetriken an Amazon CloudWatch Logs zu senden.
- `--enable-cloudwatch-logs-exports`: Sie müssen `postgresql`-Protokolle zu CloudWatch-Protokollen exportieren.

Im folgenden Beispiel wird der erweiterte Modus von Database Insights aktiviert, wenn ein DB-Cluster erstellt wird.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
```

```
--database-insights-mode advanced \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
--enable-performance-insights \
--performance-insights-retention-period 465 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

Für Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode advanced ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 465 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

Um den erweiterten Modus von Database Insights zu aktivieren, wenn Sie DB-Cluster erstellen, geben Sie die folgenden Parameter für Ihren Vorgang mit der Amazon-RDS-API CreateDBCluster an.

- `DatabaseInsightsMode` auf `advanced`
- `Engine` auf `aurora-postgresql`
- `EngineVersion` auf einer verfügbaren Engine-Version für Limitless Database
- `StorageType` auf `aurora-iopt1`
- `ClusterScalabilityType` auf `limitless`

- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` auf `True`
- `PerformanceInsightsRetentionPeriod` auf mindestens `465` Tage
- `MonitoringInterval` auf einen Wert, der nicht `0` ist
- `MonitoringRoleArn` auf den Amazon-Ressourcennamen (ARN) für die IAM-Rolle, die es RDS erlaubt, erweiterte Überwachungsmetriken an Amazon CloudWatch Logs zu senden.

Aktivieren des erweiterten Modus von Database Insights beim Ändern eines DB-Clusters für Aurora PostgreSQL Limitless Database

Aktivieren Sie Database Insights, wenn Sie eine Datenbank für Aurora PostgreSQL Limitless Database ändern.

> **Note**
>
> Um Database Insights zu aktivieren, muss jede DB-Instance in einem DB-Cluster dieselben Einstellungen für Performance Insights und Enhanced Monitoring aufweisen.

Console

In der Konsole können Sie den erweiterten Modus von Database Insights aktivieren, wenn Sie einen DB-Cluster ändern. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den erweiterten Modus von Database Insights, wenn Sie einen DB-Cluster über die Konsole ändern

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie einen DB-Cluster aus und klicken Sie dann auf Ändern.
4. Wählen Sie im Abschnitt Database Insights Erweiterter Modus aus. Wählen Sie dann eine der folgenden Optionen aus:

- Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Der Aufbewahrungszeitraum für den erweiterten Modus von Database Insights muss 15-24 Monate betragen.
- AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.

5. Klicken Sie auf Weiter.
6. Wählen Sie für Einplanung von Änderungen die Option Sofort anwenden aus. Wenn Sie Anwendung während des nächsten geplanten Wartungsfensters auswählen, ignoriert Ihre Datenbank diese Einstellung und aktiviert den erweiterten Modus von Database Insights sofort.
7. Wählen Sie Cluster bearbeiten aus.

AWS CLI

Um den erweiterten Modus von Database Insights beim Ändern eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl modify-db-cluster auf und geben Sie die folgenden Werte ein:

- `--database-insights-mode advanced`, um den erweiterten Modus von Database Insights zu aktivieren.
- `--db-cluster-identifier`: Die Kennung für den DB-Cluster.
- `--enable-performance-insights`, um Performance Insights für Database Insights zu aktivieren.
- `--performance-insights-retention-period`: Der Aufbewahrungszeitraum für Daten für Ihren DB-Cluster. Um den erweiterten Modus von Database Insights zu aktivieren, muss der Aufbewahrungszeitraum mindestens 465 Tage betragen.

Im folgenden Beispiel wird der erweiterte Modus von Database Insights aktiviert, wenn ein DB-Cluster geändert wird.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode advanced \
```

```
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

Für Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode advanced ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

Um den erweiterten Modus von Database Insights zu aktivieren, wenn Sie einen DB-Cluster ändern, geben Sie die folgenden Parameter für Ihren Vorgang der Amazon-RDS-API ModifyDBCluster an.

- `DatabaseInsightsMode` auf `advanced`
- `EnablePerformanceInsights` auf `True`
- `PerformanceInsightsRetentionPeriod` auf mindestens `465` Tage

Aktivieren des Standardmodus von Database Insights für Aurora PostgreSQL Limitless Database

Gehen Sie wie folgt vor, um den Standardmodus von Database Insights für Ihre Aurora PostgreSQL Limitless Database zu aktivieren.

Aktivieren des Standardmodus von Database Insights beim Erstellen eines DB-Clusters für Aurora PostgreSQL Limitless Database

Aktivieren Sie den Standardmodus von Database Insights, wenn Sie eine Datenbank für Aurora PostgreSQL Limitless Database erstellen.

Console

In der Konsole können Sie den Standardmodus von Database Insights aktivieren, wenn Sie einen DB-Cluster erstellen. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den Standardmodus von Database Insights, wenn Sie einen DB-Cluster über die Konsole erstellen

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie Create database (Datenbank erstellen) aus.
4. Wählen Sie im Abschnitt Database Insights Standardmodus aus. Wählen Sie dann eine der folgenden Optionen aus:
   - Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Um einen DB-Cluster für Aurora PostgreSQL Limitless Database zu erstellen, muss der Aufbewahrungszeitraum mindestens 31 Tage betragen.
   - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
5. Wählen Sie Create database (Datenbank erstellen) aus.

AWS CLI

Um den Standardmodus von Database Insights beim Erstellen eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl create-db-cluster auf und geben Sie die folgenden Werte ein:

- `--db-cluster-identifier`: Die Kennung für den DB-Cluster.
- `--database-insights-mode standard`: Um den Standardmodus von Database Insights zu aktivieren.
- `--engine`: Der DB-Cluster muss die `aurora-postgresql`-DB-Engine verwenden.
- `--engine-version`: Der DB-Cluster muss eine der DB-Engine-Versionen verwenden:
  - `16.4-limitless`
  - `16.6-limitless`
- `--storage-type`: Der DB-Cluster muss die Speicherkonfiguration des `aurora-iopt1`-DB-Clusters verwenden.
- `--cluster-scalability-type`: Gibt den Skalierbarkeitsmodus des Aurora-DB-Clusters an. Wenn dieser Parameter auf `limitless` gesetzt ist, arbeitet der Cluster als Aurora PostgreSQL

Limitless Database. Wenn er auf `standard` (Standard) gesetzt ist, verwendet der Cluster die normale DB-Instance-Erstellung.

> **Note**
>
> Sie können diese Einstellung nicht mehr ändern, nachdem Sie den DB-Cluster erstellt haben.

- `--master-username`: Der Name des Masterbenutzers für diesen DB-Cluster.
- `--master-user-password`: Das Passwort für den Masterbenutzer.
- `--enable-performance-insights`, um Performance Insights für Database Insights zu aktivieren.
- `--performance-insights-retention-period`: Der Aufbewahrungszeitraum für Daten für Ihren DB-Cluster. Um einen DB-Cluster für Aurora PostgreSQL Limitless Database zu erstellen, muss der Aufbewahrungszeitraum mindestens 31 Tage betragen.
- `--monitoring-interval`: Das Intervall in Sekunden zwischen den Punkten, an denen erweiterte Überwachungsmetriken für den DB-Cluster erfasst werden. Dieser Wert darf nicht `0` sein.
- `--monitoring-role-arn`: Der Amazon-Ressourcenname (ARN) für die IAM-Rolle, die es RDS erlaubt, erweiterte Überwachungsmetriken an Amazon CloudWatch Logs zu senden.
- `--enable-cloudwatch-logs-exports`: Sie müssen `postgresql`-Protokolle zu CloudWatch-Protokollen exportieren.

Im folgenden Beispiel wird der Standardmodus von Database Insights aktiviert, wenn ein DB-Cluster erstellt wird.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier my-limitless-cluster \
--database-insights-mode standard \
--engine aurora-postgresql \
--engine-version 16.6-limitless \
--storage-type aurora-iopt1 \
--cluster-scalability-type limitless \
--master-username myuser \
--master-user-password mypassword \
```

```
--enable-performance-insights \
--performance-insights-retention-period 31 \
--monitoring-interval 5 \
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole \
--enable-cloudwatch-logs-exports postgresql
```

Für Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier my-limitless-cluster ^
--database-insights-mode standard ^
--engine aurora-postgresql ^
--engine-version 16.6-limitless ^
--storage-type aurora-iopt1 ^
--cluster-scalability-type limitless ^
--master-username myuser ^
--master-user-password mypassword ^
--enable-performance-insights ^
--performance-insights-retention-period 31 ^
--monitoring-interval 5 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/EMrole ^
--enable-cloudwatch-logs-exports postgresql
```

RDS API

Um den Standardmodus von Database Insights zu aktivieren, wenn Sie DB-Cluster erstellen, geben Sie die folgenden Parameter für Ihren Vorgang mit der Amazon-RDS-API CreateDBCluster an.

- `DatabaseInsightsMode` auf `standard`
- `Engine` auf `aurora-postgresql`
- `EngineVersion` auf einer verfügbaren Engine-Version für Limitless Database
- `StorageType` auf `aurora-iopt1`
- `ClusterScalabilityType` auf `limitless`
- `MasterUsername`
- `MasterUserPassword`
- `EnablePerformanceInsights` auf `True`
- `PerformanceInsightsRetentionPeriod` auf mindestens `31` Tage
- `MonitoringInterval` auf einen Wert, der nicht `0` ist

- `MonitoringRoleArn` auf den Amazon-Ressourcennamen (ARN) für die IAM-Rolle, die es RDS erlaubt, erweiterte Überwachungsmetriken an Amazon CloudWatch Logs zu senden.

Aktivieren des Standardmodus von Database Insights beim Ändern eines DB-Clusters für Aurora PostgreSQL Limitless Database

Aktivieren Sie Database Insights, wenn Sie eine Datenbank für Aurora PostgreSQL Limitless Database ändern.

> **Note**
>
> Um Database Insights zu aktivieren, muss jede DB-Instance in einem DB-Cluster dieselben Einstellungen für Performance Insights und Enhanced Monitoring aufweisen.

Console

In der Konsole können Sie den Standardmodus von Database Insights aktivieren, wenn Sie einen DB-Cluster erstellen. Die Einstellungen für Database Insights gelten für alle DB-Instances in Ihrem DB-Cluster.

So aktivieren Sie den Standardmodus von Database Insights, wenn Sie einen DB-Cluster über die Konsole ändern

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie einen DB-Cluster aus und klicken Sie dann auf Ändern.
4. Wählen Sie im Abschnitt Database Insights Standardmodus aus. Wählen Sie dann eine der folgenden Optionen aus:

   - Aufbewahrung – Die Zeitspanne, für die die Performance-Insights-Daten aufbewahrt werden sollen. Um einen DB-Cluster für Aurora PostgreSQL Limitless Database zu erstellen, muss der Aufbewahrungszeitraum mindestens 31 Tage betragen.
   - AWS KMS key – Geben Sie Ihren KMS-Schlüssel an. Performance Insights verschlüsselt alle potenziell sensiblen Daten mit Ihrem KMS-Schlüssel. Die Daten werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.

5. Klicken Sie auf Weiter.
6. Wählen Sie für Einplanung von Änderungen die Option Sofort anwenden aus. Wenn Sie Anwendung während des nächsten geplanten Wartungsfensters auswählen, ignoriert Ihre Datenbank diese Einstellung und aktiviert den Standardmodus von Database Insights sofort.
7. Wählen Sie Cluster bearbeiten aus.

AWS CLI

Um den Standardmodus von Database Insights beim Ändern eines DB-Clusters zu aktivieren, rufen Sie den AWS CLI-Befehl modify-db-cluster auf und geben Sie die folgenden Werte ein:

- `--db-cluster-identifier`: Die Kennung für den DB-Cluster.
- `--database-insights-mode standard`, um den Standardmodus von Database Insights zu aktivieren.
- `--enable-performance-insights`, um Performance Insights für Database Insights zu aktivieren.
- `--performance-insights-retention-period`: Der Aufbewahrungszeitraum für Daten für Ihren DB-Cluster. Um den Standardmodus von Database Insights zu aktivieren, muss der Aufbewahrungszeitraum mindestens 31 Tage betragen.

Im folgenden Beispiel wird der Standardmodus von Database Insights aktiviert, wenn ein DB-Cluster geändert wird.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --database-insights-mode standard \
    --db-cluster-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 31
```

Für Windows:

```
aws rds modify-db-cluster ^
    --database-insights-mode standard ^
    --db-cluster-identifier sample-db-identifier ^
    --enable-performance-insights ^
```

```
    --performance-insights-retention-period 31
```

RDS API

Um den Standardmodus von Database Insights zu aktivieren, wenn Sie einen DB-Cluster ändern, geben Sie die folgenden Parameter für Ihren Vorgang der Amazon-RDS-API ModifyDBCluster an.

- `DatabaseInsightsMode` auf `standard`
- `EnablePerformanceInsights` auf `True`
- `PerformanceInsightsRetentionPeriod` auf mindestens `31` Tage

Überwachen von Aurora PostgreSQL Limitless-Datenbank mit Amazon CloudWatch Logs

Der Export von PostgreSQL-Protokollen nach CloudWatch Logs ist im Rahmen der Aktivierung von Aurora PostgreSQL Limitless Database erforderlich. Sie können in CloudWatch Logs Insights auf diese Protokolle zugreifen und sie analysieren, ähnlich wie beim Zugriff auf PostgreSQL-Protokolle für einen standardmäßigen DB-Cluster von Aurora PostgreSQL. Weitere Informationen finden Sie unter Analysieren von Aurora-PostgreSQL-Protokollen mit CloudWatch Logs Insights.

Der Name der Protokollgruppe für den DB-Cluster ist derselbe wie in Aurora PostgreSQL:

```
/aws/rds/cluster/DB_cluster_ID/postgresql
```

Der Name der Protokollgruppe für die DB-Shard-Gruppe hat folgendes Format:

```
/aws/rds/cluster/DB_cluster_ID/DB_shard_group_ID/postgresql
```

Für jeden Knoten (Router oder Shard) gibt es Protokollstreams. Ihre Namen haben das folgende Format:

```
[DistributedTransactionRouter|
DataAccessShard]/node_cluster_serial_ID-node_instance_serial_ID/n
```

Zum Beispiel:

- Router – `DistributedTransactionRouter/6-6.2`
- Shard – `DataAccessShard/22-22.0`

> **Note**
>
> Sie können PostgreSQL-Protokolldateien für die DB-Shard-Gruppe nicht direkt in der RDS-Konsole, in der AWS CLI oder in der RDS-API anzeigen, wie es für den DB-Cluster möglich ist. Um sie anzuzeigen, müssen Sie CloudWatch Logs Insights verwenden.

Überwachen einer Aurora PostgreSQL Limitless Database mithilfe der erweiterten Überwachung

Im Rahmen der Aktivierung von Aurora PostgreSQL Limitless Database ist eine erweiterte Überwachung erforderlich. Sie können damit das Betriebssystem Ihrer DB-Instances von Limitless Database in Echtzeit überwachen.

Aurora veröffentlicht die Metriken für erweiterte Überwachung in CloudWatch-Protokollen. Zu den wichtigsten verfügbaren Metriken gehören Datenbankverbindungen, Speichernutzung und Abfragelatenz. Diese können helfen, Leistungsengpässe zu identifizieren.

Weitere Informationen zu Metriken für erweiterte Überwachung finden Sie unter Betriebssystemmetriken für Aurora.

Überwachen einer Aurora PostgreSQL Limitless Database mithilfe von Performance Insights

Verwenden Sie Performance Insights, um Ihren Cluster von Aurora PostgreSQL Limitless Database zu überwachen. Performance Insights funktioniert für Aurora PostgreSQL Limitless Database ähnlich wie für standardmäßige Aurora-DB-Cluster. Sie verfolgen jedoch Metriken für Aurora PostgreSQL Limitless Database auf Shard-Gruppenebene.

Die beiden wichtigsten Performance-Insights-Metriken, die es zu verfolgen gilt, sind die folgenden:

- Datenbanklast: Misst den Aktivitätsgrad in Ihrer Datenbank. Die wichtigste Metrik für Performance Insights ist `DBLoad`, die jede Sekunde erfasst wird.

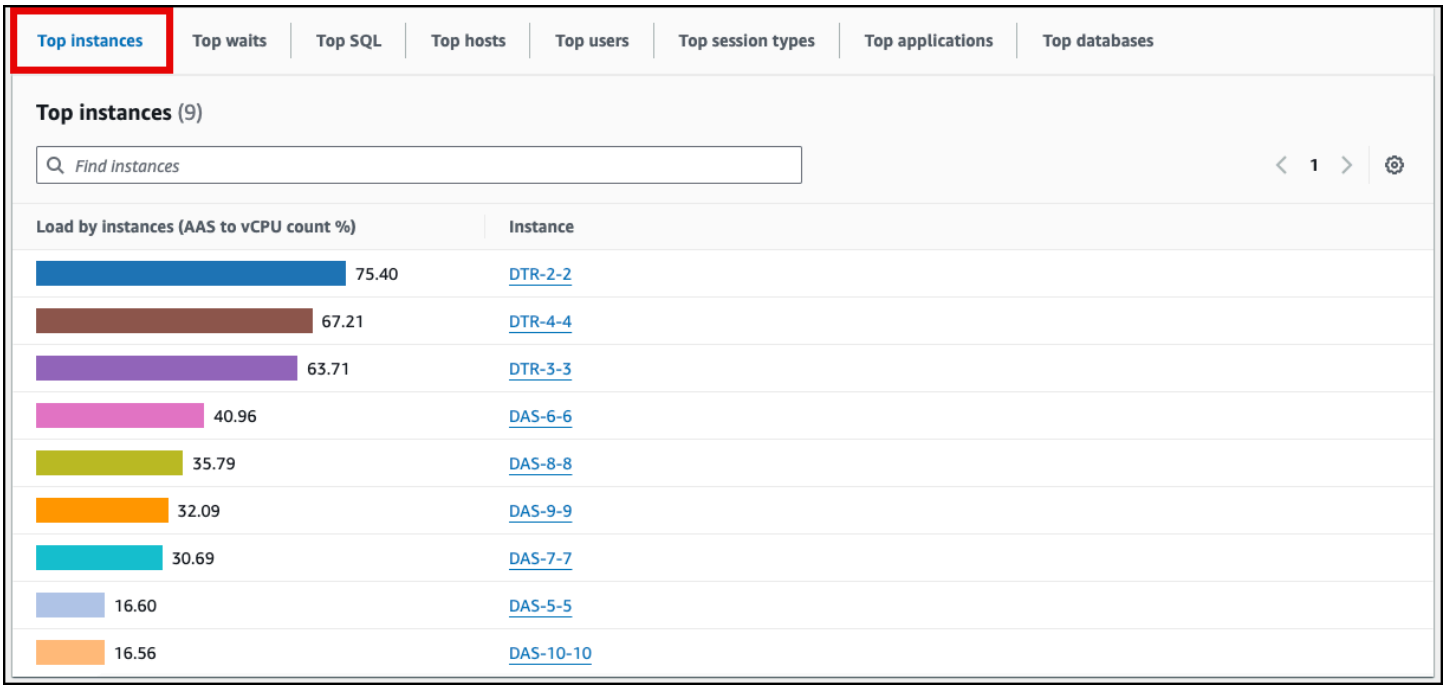
  Die Einheit für die Metrik `DBLoad` in Performance Insights ist AAS (Durchschnittliche aktive Sitzungen) Um die Anzahl der durchschnittlich aktiven Sitzungen AAS zu erhalten, ruft Performance Insights die Anzahl der Sitzungen ab, die gleichzeitig eine Abfrage ausführen. Die AAS ist die Gesamtzahl der Sitzungen geteilt durch die Gesamtzahl der Beispiele. Weitere Informationen zu `DBLoad` und AAS finden Sie unter Datenbanklast.
- Maximale CPU: Die maximale Rechenleistung, die Ihrer Datenbank zur Verfügung steht. Um zu sehen, ob aktive Sitzungen die maximale CPU überschreiten, sehen Sie sich ihre Beziehung zur `Max vCPU`-Linie an. Der `Max vCPU`-Wert wird anhand der Anzahl der vCPU (virtuellen CPU)-Cores für Ihre DB-Instance bestimmt. Weitere Informationen zu `Max vCPU` finden Sie unter Maximale CPU.

Darüber hinaus können Sie die Metrik `DBLoad` in Dimensionen aufteilen, also in „Unterkategorien“ der Metrik. Die am häufigsten verwendeten Dimensionen sind folgende:

- Top-Instances: Zeigt die relative DB-Auslastung für Ihre Instances (Shards und Router) in absteigender Reihenfolge an.
- Warteereignis: Bewirkt, dass SQL-Anweisungen warten, bis ein bestimmtes Ereignis eintritt, bevor sie mit der Ausführung fortfahren können. Warteereignisse geben an, wo Arbeit behindert wird.
- Top-SQL: Zeigt, welche Abfragen am meisten zur Datenbanklast beitragen.

Weitere Informationen zu Performance-Insights-Dimensionen finden Sie unter Dimensionen.

Die folgende Abbildung zeigt die Dimension Top-Instances für eine DB-Shard-Gruppe.



Themen

- [Analysieren der DB-Last nach Wartevorgängen für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights]

Analysieren der DB-Last nach Wartevorgängen für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights

Mit Performance Insights können Sie Metriken auf Shard-Gruppenebene und Instance-Ebene für eine Aurora PostgreSQL Limitless-Datenbank verfolgen. Bei der Analyse der DB-Auslastung für eine Aurora PostgreSQL Limitless Database können Sie die DB-Last für jeden Shard und Router mit der maximalen vCPU vergleichen.

> **Note**
>
> Bei Aurora PostgreSQL Limitless Database sind Performance Insights und Enhanced Monitoring immer aktiviert. Die Mindestaufbewahrungsdauer für Performance-Insights-Daten für Limitless Database beträgt 31 Tage (1 Monat).

In der Ansicht Absolut werden die Anzahl der durchschnittlichen aktiven Sitzungen (AAS) und die geschätzte vCPU angezeigt. Die Ansicht Relativ zeigt das Verhältnis von AAS zur geschätzten vCPU.

Themen

- Analysieren der relativen DB-Last für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights
- Analysieren der DB-Last nach Wartevorgängen für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights
- Analysieren der Lastverteilung für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights

Analysieren der relativen DB-Last für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights

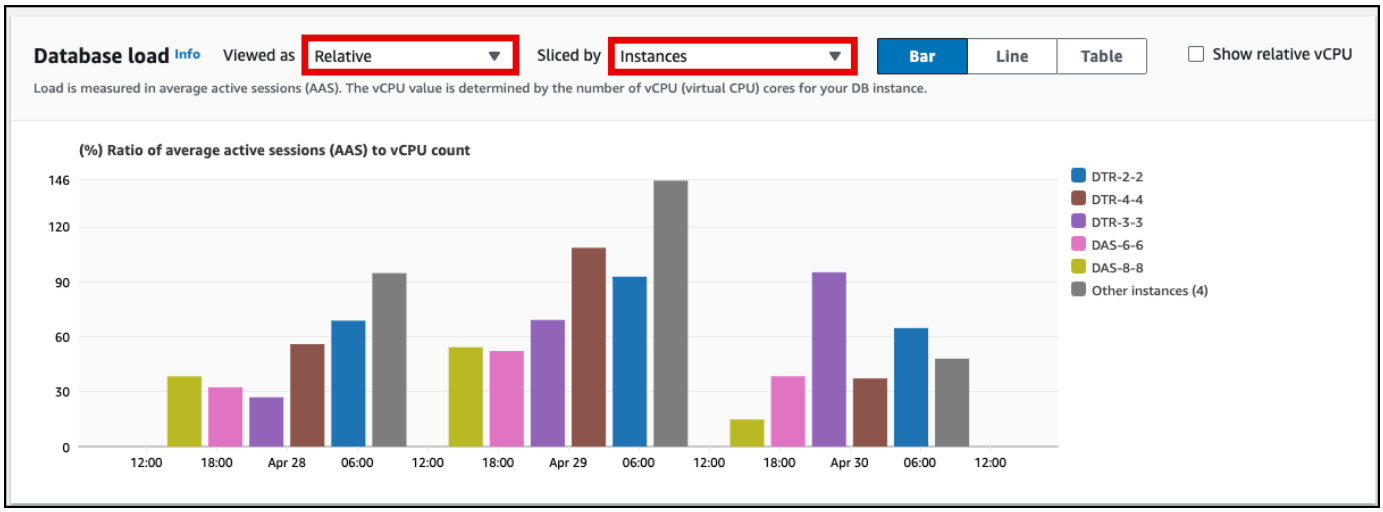
Sie können die Leistung Ihrer Aurora PostgreSQL Limitless Database verbessern, indem Sie die relative DB-Last verfolgen. Gehen Sie wie folgt vor, um die relative DB-Last nach Instance für Ihre Aurora PostgreSQL Limitless Database zu analysieren.

So analysieren Sie die relative DB-Last für Aurora PostgreSQL Limitless Database über die Konsole

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.

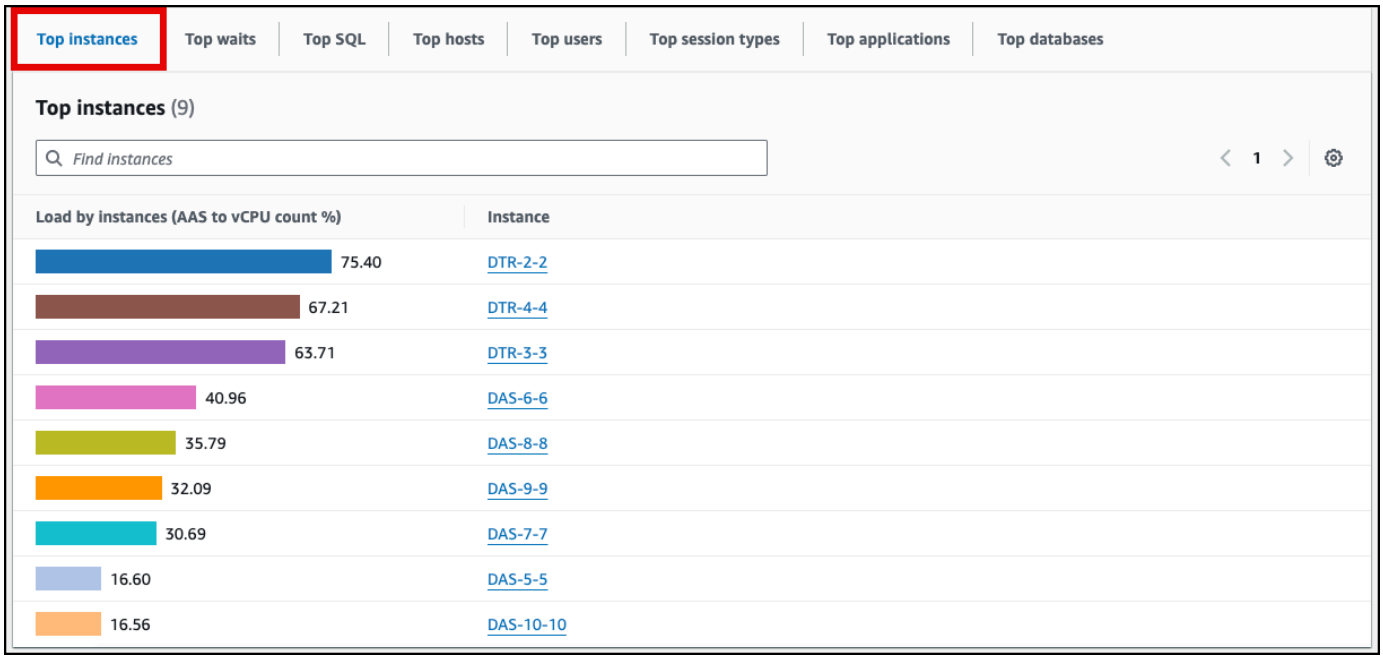
3. Wählen Sie eine Aurora PostgreSQL Limitless Database aus. Das Dashbard von Performance Insights wird für diese Aurora PostgreSQL Limitless Database angezeigt.
4. Wählen Sie im Abschnitt Datenbanklast (DB-Last) die Option Instances für Aufgeteilt nach aus. Um das Verhältnis von durchschnittlichen aktiven Sitzungen (AAS) zu vCPU-Kernen für alle Instances in Ihrer Aurora PostgreSQL Limitless Database anzuzeigen, wählen Sie Relativ für Angezeigt als aus.

   Im Diagramm „Durchschnittliche aktive Sitzungen“ wird die DB-Last für Instances in Ihrer Aurora PostgreSQL Limitless Database angezeigt.



5. Um die Top-Instances anzuzeigen, wählen Sie die Registerkarte Top-Instances aus.

   Im folgenden Beispiel ist `DTR-2-2` die Instance mit der höchsten DB-Last



6. (Optional) Um die DB-Last für eine Instance in Ihrer Aurora PostgreSQL Limitless Database zu analysieren, wählen Sie den Instance-Namen in der Spalte Instances aus. Um die DB-Last für `DTR-2-2` anzuzeigen, wählen Sie `DTR-2-2` in der Spalte Instances aus.

> **Note**
>
> Sie können Performance-Insights-Metriken nur für Instances in einer Aurora PostgreSQL Limitless Database anzeigen.

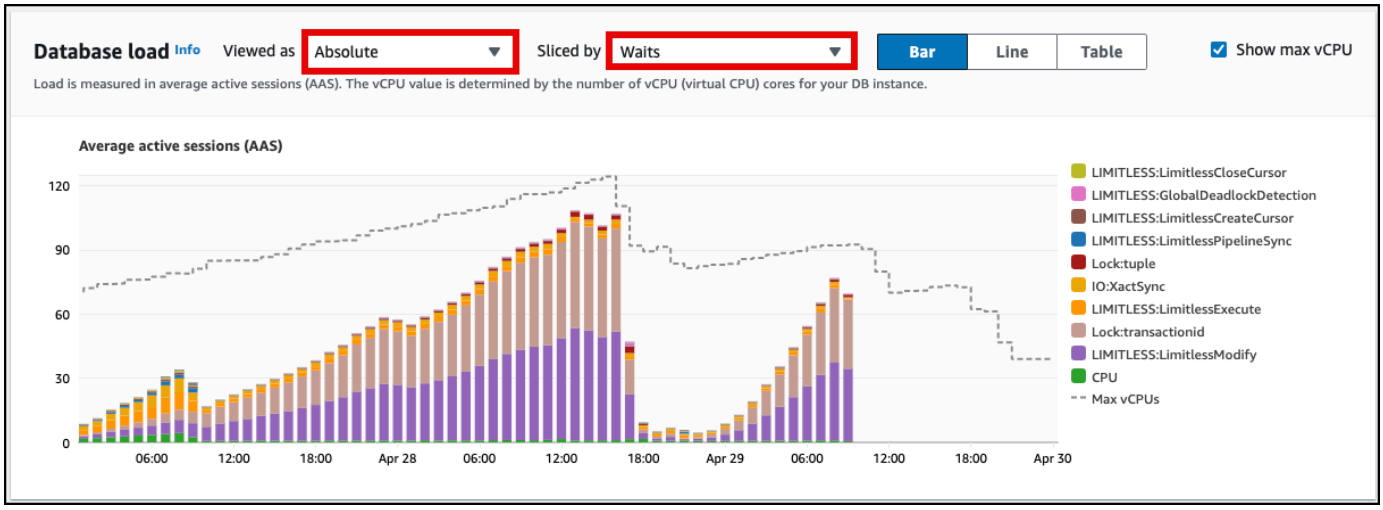
Analysieren der DB-Last nach Wartevorgängen für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights

Sie können die Leistung Ihrer Aurora PostgreSQL Limitless Database verbessern, indem Sie Warteereignisse verfolgen. Gehen Sie wie folgt vor, um die DB-Last nach Warteereignissen für Ihre Aurora PostgreSQL Limitless Database zu analysieren.

So analysieren Sie die Datenbanklast nach Wartevorgängen für Aurora PostgreSQL Limitless Database über die Konsole

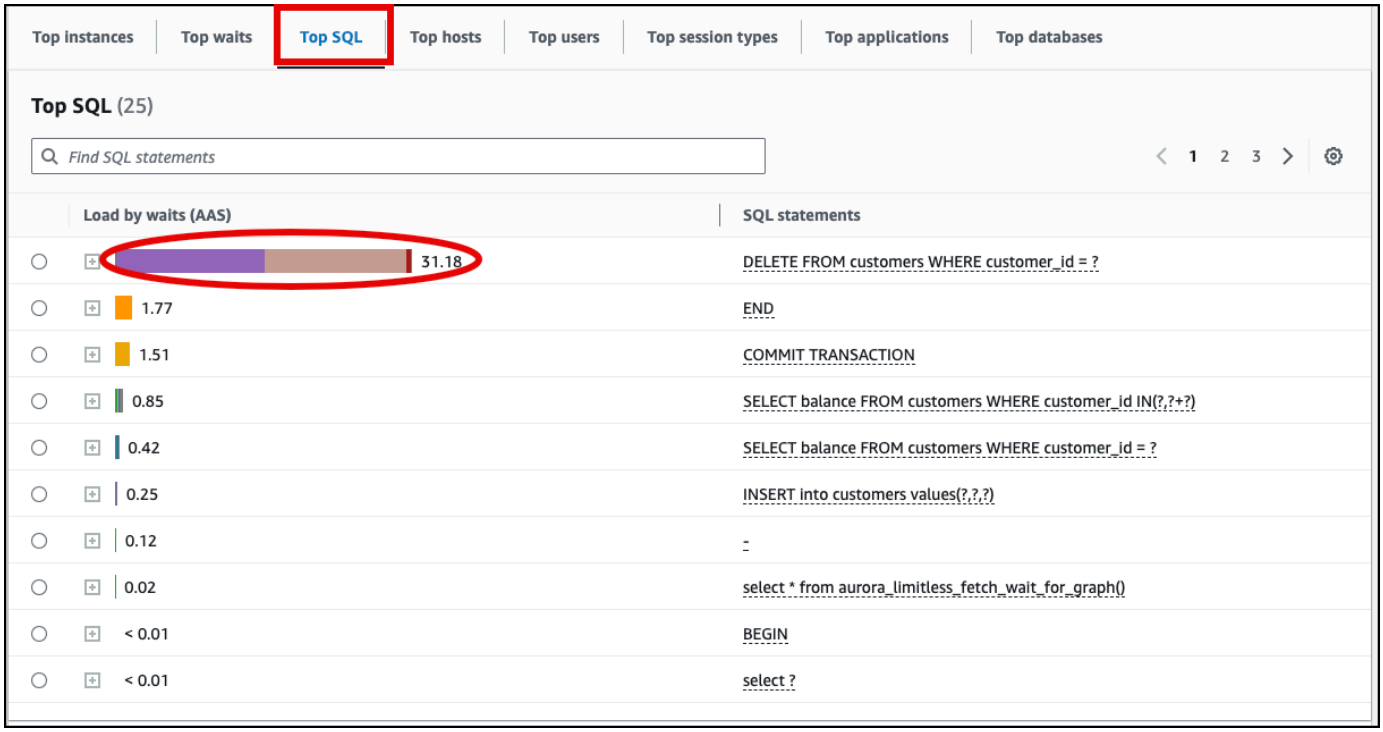
1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine Aurora PostgreSQL Limitless Database aus. Das Dashbard von Performance Insights wird für diese Aurora PostgreSQL Limitless Database angezeigt.
4. Wählen Sie im Abschnitt Datenbanklast (DB-Last) die Option Wartevorgänge für Aufgeteilt nach aus. Um die Anzahl der AAS und die geschätzte vCPU anzuzeigen, wählen Sie Absolut für Angezeigt als aus.

   Im Diagramm „Durchschnittliche aktive Sitzungen“ wird die DB-Last für Instances in Ihrer Aurora PostgreSQL Limitless Database angezeigt.



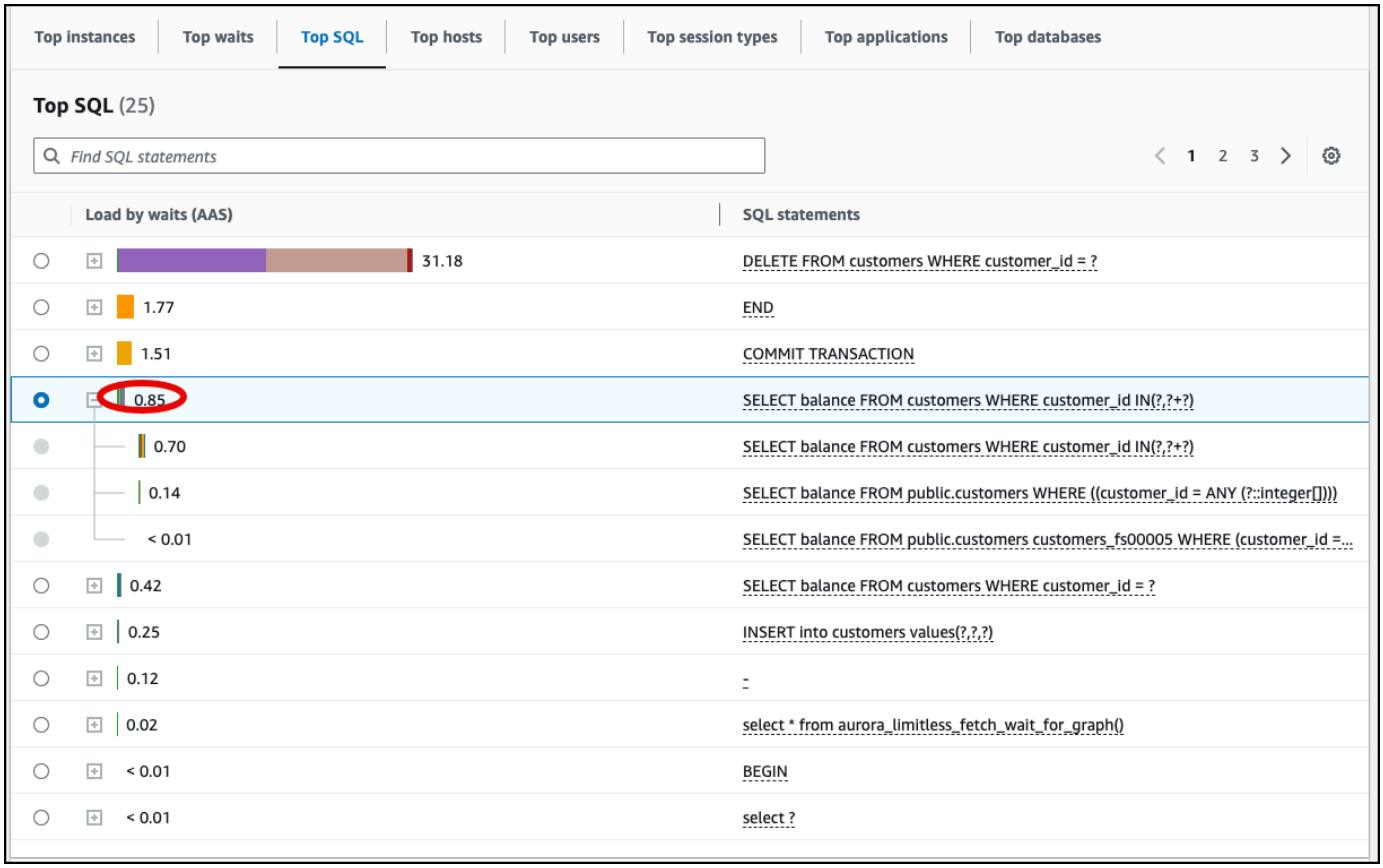
5. Scrollen Sie nach unten zur Registerkarte Top SQL (Top-SQL).

   Im folgenden Beispiel ist die SQL-Anweisung mit der höchsten Last nach Wartevorgängen die Anweisung `DELETE`.



6. Wählen Sie die SQL-Anweisung aus, um sie in auf ihre Komponentenanweisungen zu erweitern.

   Im folgenden Beispiel umfasst die `SELECT`-Anweisung drei Komponentenanweisungen.



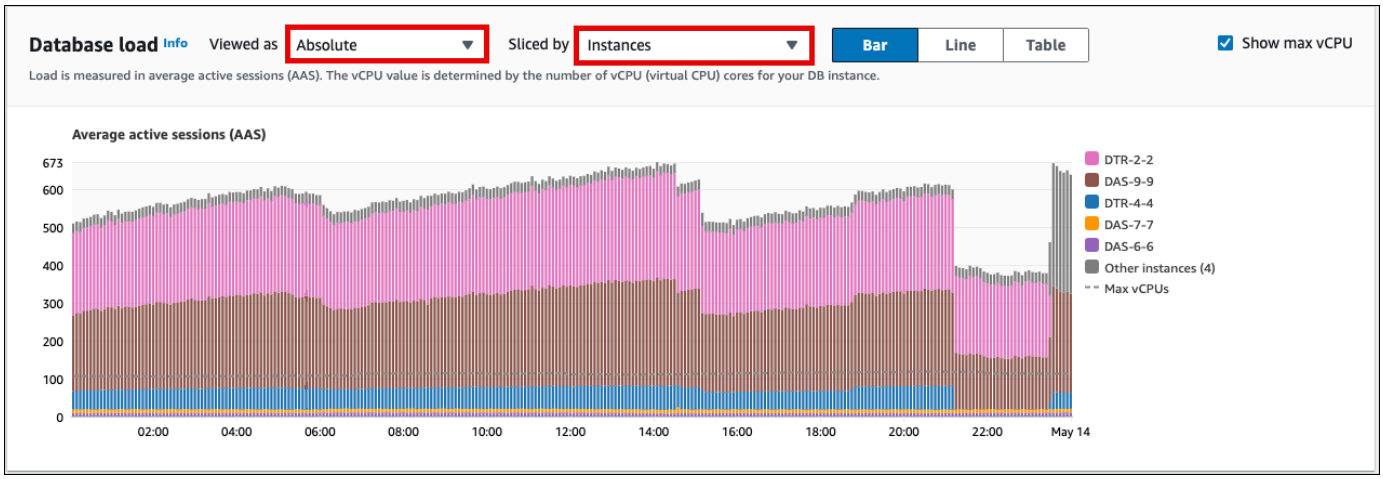
Analysieren der Lastverteilung für Aurora PostgreSQL Limitless Database über das Dashboard von Performance Insights

Möglicherweise möchten Sie die Lastverteilung für Instances auf Ihrer Aurora PostgreSQL Limitless Database ausgleichen. Gehen Sie wie folgt vor, um die Lastverteilung der Instances auf einer Aurora PostgreSQL Limitless Database zu analysieren.

So analysieren Sie die Lastverteilung der Instances auf einer Aurora PostgreSQL Limitless Database über die Konsole

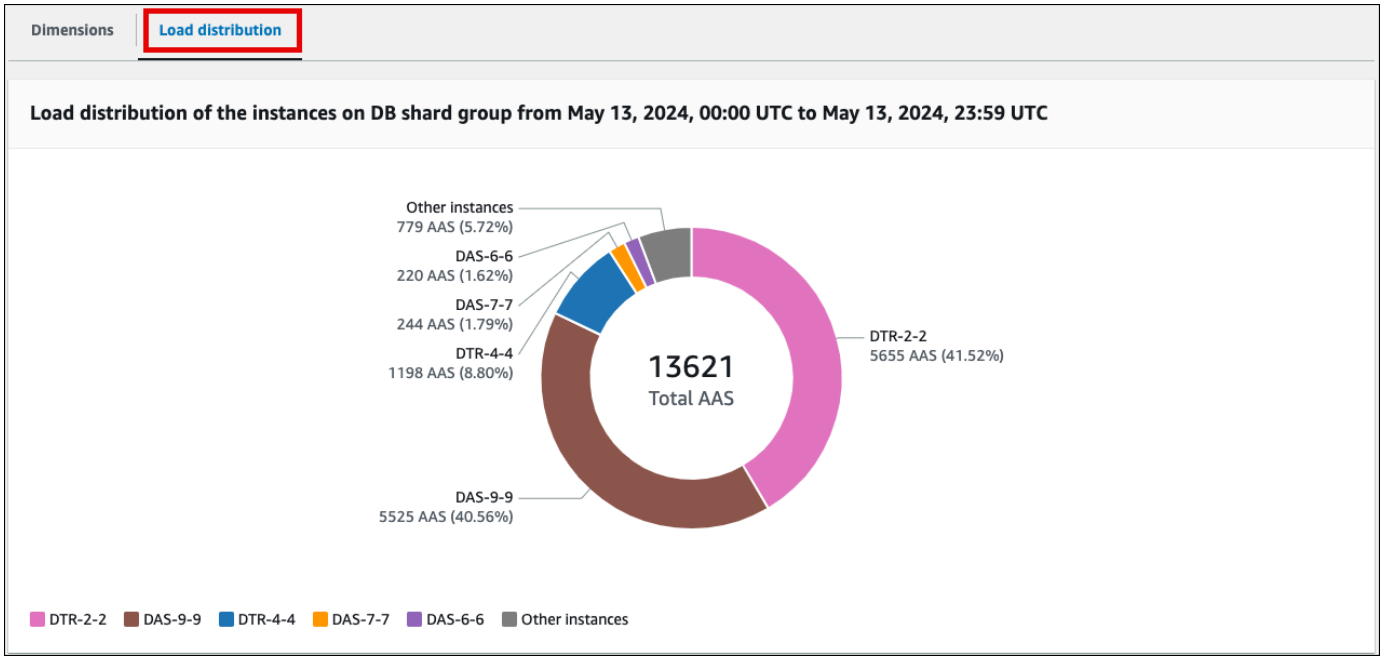
1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Performance-Insights aus.
3. Wählen Sie eine Aurora PostgreSQL Limitless Database aus. Das Dashbard von Performance Insights wird für diese Aurora PostgreSQL Limitless Database angezeigt.
4. Wählen Sie im Abschnitt Datenbanklast (DB-Last) die Option Instances für Aufgeteilt nach aus. Um die Anzahl der AAS und die geschätzte vCPU für alle Instances in Ihrer Aurora PostgreSQL Limitless Database anzuzeigen, wählen Sie Absolut für Angezeigt als aus.

Im Diagramm „Durchschnittliche aktive Sitzungen“ wird die DB-Last für Instances in Ihrer Aurora PostgreSQL Limitless Database angezeigt.



5. Um ein Diagramm der Lastverteilung der Instances in Ihrer Aurora PostgreSQL Limitless Database anzuzeigen, wählen Sie die Registerkarte Lastverteilung aus.

   Im folgenden Beispiel ist `DTR-2-2` die Instance mit der höchsten DB-Last



Überwachen einer Aurora PostgreSQL Limitless Database mithilfe des RDS-Schutzes von Amazon GuardDuty

Amazon GuardDuty ist ein Service zur Erkennung von Bedrohungen, der Sie beim Schutz Ihrer Konten, Container, Workloads und Daten in Ihrer AWS-Umgebung unterstützt. Mithilfe von Machine Learning (ML)-Modellen und Funktionen zur Erkennung von Anomalien und Bedrohungen überwacht GuardDuty kontinuierlich verschiedene Protokollquellen und Laufzeitaktivitäten, um potenzielle Sicherheitsrisiken und böswillige Aktivitäten in Ihrer Umgebung zu identifizieren und zu priorisieren.

GuardDuty RDS Protection analysiert und profiliert Anmeldeereignisse auf potenzielle Zugriffsbedrohungen, die für Ihre Amazon-Aurora-Datenbanken bestehen könnten. Wenn Sie RDS Protection aktivieren, verarbeitet GuardDuty RDS-Anmeldeereignisse aus Ihren Aurora-Datenbanken. RDS Protection überwacht diese Ereignisse und erstellt ein Profil für potenzielle interne Bedrohungen oder externe Akteure.

Weitere Informationen zum RDS-Schutz von GuardDuty in Aurora finden Sie unter Überwachen von Bedrohungen mit Amazon GuardDuty RDS Protection für Amazon Aurora.

Weitere Informationen zum Aktivieren des RDS-Schutzes von GuardDuty finden Sie unter GuardDuty-RDS-Schutz im Amazon-GuardDuty-Benutzerhandbuch.

Funktionen und Ansichten für Aurora PostgreSQL Limitless Database

Aurora PostgreSQL Limitless Database verfügt über hinzugefügte Funktionen und Ansichten. Sie basieren auf den entsprechenden Funktionen und Ansichten von Aurora PostgreSQL.

> **Note**
>
> Einige Statistiken können uneinheitliche Ergebnisse zurückgeben, wenn Transaktionen im Gange sind.

Themen

- Funktionen für Aurora PostgreSQL Limitless Database
- Ansichten für Aurora PostgreSQL Limitless Database

Funktionen für Aurora PostgreSQL Limitless Database

In der folgenden Tabelle sehen Sie die neuen Funktionen für Aurora PostgreSQL Limitless Database.

> **Note**
>
> Die in dieser Tabelle aufgeführten Funktionen befinden sich im Schema `rds_aurora`. Wenn Sie eine Limitless-Database-Funktion verwenden, stellen Sie sicher, dass Sie den vollqualifizierten Objektnamen angeben: `rds_aurora.`*`object_name`*.

Funktion für Aurora PostgreSQL Limitless Database	Entsprechende Funktion für Aurora PostgreSQL
limitless_backend_dsid	pg_backend_pid
limitless_cancel_session	pg_cancel_backend
limitless_stat_clear_snapshot	pg_stat_clear_snapshot
limitless_stat_database_size	pg_database_size

Funktion für Aurora PostgreSQL Limitless Database	Entsprechende Funktion für Aurora PostgreSQL
limitless_stat_get_snapshot_timestamp	pg_stat_get_snapshot_timestamp
limitless_stat_prepared_xacts	pg_prepared_xacts
limitless_stat_relation_sizes	pg_indexes_size, pg_relation_size, pg_table_size, pg_total_relation_size
limitless_stat_reset	pg_stat_reset
limitless_stat_statements_reset	pg_stat_statements_reset
limitless_stat_system_waits	aurora_stat_system_waits
limitless_terminate_session	pg_terminate_backend
limitless_wait_report	aurora_wait_report

Die folgenden Beispiele enthalten Einzelheiten zu den Funktionen für Aurora PostgreSQL Limitless Database. Weitere Informationen zu PostgreSQL-Funktionen finden Sie unter Funktionen und Operatoren in der PostgreSQL-Dokumentation.

limitless_backend_dsid

Die Funktion `limitless_backend_dsid` gibt die ID der verteilten Sitzung zurück. Eine verteilte Sitzung wird auf einem Router in einer DB-Shard-Gruppe ausgeführt und umfasst Backend-Prozesse auf einem oder mehreren Shards in der DB-Shard-Gruppe.

Das folgende Beispiel zeigt, wie die Funktion `limitless_backend_dsid` verwendet werden kann.

```
SELECT rds_aurora.limitless_backend_dsid();

limitless_backend_dsid
------------------------
8CACD7B04D0FC2A5
(1 row)
```

limitless_cancel_session

Die Funktion `limitless_cancel_session` arbeitet ähnlich wie `pg_cancel_backend`, versucht aber, alle Backend-Prozesse im Zusammenhang mit der angegebenen ID einer verteilten Sitzung abzubrechen, indem sie ein `SIGINT` (Unterbrechungssignal) sendet.

Es wird der folgende Eingabeparameter verwendet:

- `distributed_session_id` (Text): Die ID der verteilten Sitzung, die abgebrochen werden soll.

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `pid` (Text): Die ID des Backend-Prozesses.
- `success` (Boolean): Ob der Abbruch erfolgreich war.

Das folgende Beispiel zeigt, wie die Funktion `limitless_cancel_session` verwendet werden kann.

```
SELECT * FROM rds_aurora.limitless_cancel_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_stat_clear_snapshot

Die Funktion `limitless_stat_clear_snapshot` verwirft den aktuellen Statistik-Snapshot oder die zwischengespeicherten Informationen auf allen Knoten.

Das folgende Beispiel zeigt, wie die Funktion `limitless_stat_clear_snapshot` verwendet werden kann.

```
SELECT rds_aurora.limitless_stat_clear_snapshot();
```

limitless_stat_database_size

Die Funktion `limitless_stat_database_size` gibt die Größen einer Datenbank in der DB-Shard-Gruppe zurück.

Es wird der folgende Eingabeparameter verwendet:

- dbname (Name): Die Datenbank, für die die Größen abgerufen werden sollen.

Es werden folgende Ausgabeparameter verwendet:

- subcluster_id (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- subcluster_type (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: router oder shard.
- db_size: Die Größe der Datenbank in diesem Subcluster in Byte.

Das folgende Beispiel zeigt, wie die Funktion limitless_stat_database_size verwendet werden kann.

```
SELECT * FROM rds_aurora.limitless_stat_database_size('postgres_limitless');

 subcluster_id | subcluster_type | db_size
---------------+-----------------+----------
             1 | router          |  8895919
             2 | router          |  8904111
             3 | shard           | 21929391
             4 | shard           | 21913007
             5 | shard           | 21831087
(5 rows)
```

limitless_stat_get_snapshot_timestamp

Die Funktion limitless_stat_get_snapshot_timestamp gibt den Zeitstempel des aktuellen Statistik-Snapshots zurück, oder NULL, wenn kein Statistik-Snapshot erstellt wurde. Ein Snapshot wird erstellt, wenn in einer Transaktion zum ersten Mal auf kumulative Statistiken zugegriffen wird, sofern stats_fetch_consistency auf snapshot gesetzt ist. Gibt eine konsolidierte Ansicht der Snapshot-Zeitstempel von allen Knoten zurück. Die Spalten subcluster_id und subcluster_type zeigen, von welchem Knoten die Daten stammen.

Das folgende Beispiel zeigt, wie die Funktion limitless_stat_get_snapshot_timestamp verwendet werden kann.

```
SELECT * FROM rds_aurora.limitless_stat_get_snapshot_timestamp();

 subcluster_id | subcluster_type | snapshot_timestamp
---------------+-----------------+--------------------
             1 | router          |
             2 | router          |
```

```
             3 | shard           |
             4 | shard           |
             5 | shard           |
(5 rows)
```

limitless_stat_prepared_xacts

Die Funktion `limitless_stat_prepared_xacts` gibt Informationen über Transaktionen auf allen Knoten zurück, die derzeit für ein zweiphasiges Commit vorbereitet sind. Weitere Informationen finden Sie unter pg_prepared_xacts in der PostgreSQL-Dokumentation.

Das folgende Beispiel zeigt, wie die Funktion `limitless_stat_prepared_xacts` verwendet werden kann.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_prepared_xacts;

 subcluster_id | subcluster_type | transaction_id |             gid              |
         prepared            |  owner_id  |    database_id
---------------+-----------------+----------------+------------------------------
+-------------------------------+------------+--------------------
 8             | shard           |        5815978 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659603+00 | auroraperf | postgres_limitless
 12            | shard           |        4599138 | 7_4599899_postgres_limitless |
 2024-09-03 15:51:17.659637+00 | auroraperf | postgres_limitless
(2 rows)
```

limitless_stat_relation_sizes

Die Funktion `limitless_stat_relation_sizes` gibt die verschiedenen Größen einer Tabelle in der DB-Shard-Gruppe zurück.

Es werden folgende Eingabeparameter verwendet:

- `relnspname` (Name): Der Name des Schemas, das die Tabelle enthält.
- `relname` (Name): Der Name der Tabelle.

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `subcluster_type` (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: `router` oder `shard`.
- `main_size`: Die Größe der Haupt-Datenzweige in diesem Knoten in Byte.

- `fsm_size`: Die Größe der Free-Space-Map für die Tabelle in diesem Knoten in Byte.
- `vm_size`: Die Größe der Sichtbarkeits-Map für die Tabelle in diesem Knoten in Byte.
- `init_size`: Die Größe der Initialisierung der Tabelle in diesem Knoten in Byte.
- `toast_size`: Die Größe der Toast-Tabelle in Byte, die der Tabelle in diesem Zweig zugeordnet ist.
- `index_size`: Die Größe aller Indizes für die Tabelle in diesem Knoten in Byte.
- `total_size`: Die Größe aller Tabellensegmente in diesem Knoten in Byte.

Das folgende Beispiel zeigt, wie die Funktion `limitless_stat_relation_sizes` verwendet werden kann (einige Spalten wurden weggelassen).

```
SELECT * FROM rds_aurora.limitless_stat_relation_sizes('public','customers');

 subcluster_id | subcluster_type | main_size | fsm_size | vm_size | toast_size |
 table_size | total_size
---------------+-----------------+-----------+----------+---------+------------
+------------+------------
             1 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             2 | router          |         0 |        0 |       0 |          0 |
   0 |          0
             3 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             4 | shard           |   4169728 |  4177920 | 1392640 |    1392640 |
 11132928 |   11132928
             5 | shard           |   3981312 |  4227072 | 1409024 |    1409024 |
 11026432 |   11026432
(5 rows)
```

limitless_stat_reset

Die Funktion `limitless_stat_reset` setzt alle Statistikzähler für die aktuelle Datenbank auf Null (0) zurück. Wenn `track_functions` aktiviert ist, zeigt die Spalte `stats_reset` in `limitless_stat_database` an, wann die Statistiken für die Datenbank zuletzt zurückgesetzt wurden. `limitless_stat_reset` kann standardmäßig nur von einem Superuser ausgeführt werden. Anderen Benutzern kann mithilfe des Privilegs `EXECUTE` eine entsprechende Berechtigung erteilt werden.

Das folgende Beispiel zeigt, wie die Funktion `limitless_stat_reset` verwendet werden kann.

```
SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

 tup_inserted | tup_deleted
--------------+-------------
          896 |           0
(1 row)

SELECT rds_aurora.limitless_stat_reset();

limitless_stat_reset
---------------------
(1 row)

SELECT tup_inserted, tup_deleted FROM pg_stat_database
WHERE datname = 'postgres_limitless';

tup_inserted | tup_deleted
-------------+-------------
           0 |           0
(1 row)
```

limitless_stat_statements_reset

Die Funktion `limitless_stat_statements_reset` verwirft Statistiken, die inzwischen von `limitless_stat_statements` erfasst wurden, für die festgelegten Parameter `username`, `dbname`, `distributed_query_id` und `queryid`. Wenn einer der Parameter nicht angegeben ist, wird für jeden Parameter der Standardwert `""` oder `0` (ungültig) verwendet, und die Statistiken, die mit anderen Parametern übereinstimmen, werden zurückgesetzt. Wenn kein Parameter angegeben ist oder alle angegebenen Parameter `""` oder `0` (ungültig) lauten, verwirft die Funktion alle Statistiken. Wenn alle Statistiken in der Ansicht `limitless_stat_statements` verworfen werden, setzt die Funktion auch die Statistiken in der Ansicht `limitless_stat_statements_info` zurück.

Es werden folgende Eingabeparameter verwendet:

- `username` (Name): Der Benutzer, der die Anweisung abgefragt hat.
- `dbname` (Name): Die Datenbank, in der die Abfrage ausgeführt wurde.
- `distributed_query_id` (bigint): Die Abfrage-ID der übergeordneten Abfrage vom Koordinatorknoten. Diese Spalte lautet `NULL`, wenn es sich um die übergeordnete Abfrage handelt. Der Koordinatorknoten gibt die ID der verteilten Abfrage an die teilnehmenden Knoten

weiter. Für die Teilnehmerknoten sind die Werte für die ID der verteilten Abfrage und die ID der Abfrage also unterschiedlich.

- `queryid` (bigint): Die Abfrage-ID der Anweisung.

Das folgende Beispiel zeigt, wie die Funktion `limitless_stat_statements_reset` verwendet wird, um alle von `limitless_stat_statements` erfassten Statistiken zurückzusetzen.

```
SELECT rds_aurora.limitless_stat_statements_reset();
```

limitless_stat_system_waits

Die Funktion `limitless_stat_system_waits` gibt eine konsolidierte Ansicht der Warteereignisdaten aus `aurora_stat_system_waits` von allen Knoten zurück, die systemweite Warteaktivitäten in einer Instance meldet. Die Spalten `subcluster_id` und `subcluster_type` zeigen, von welchem Knoten die Daten stammen.

Das folgende Beispiel zeigt, wie die Funktion `limitless_stat_system_waits` verwendet werden kann.

```
postgres_limitless=> SELECT *
FROM rds_aurora.limitless_stat_system_waits() lssw, pg_catalog.aurora_stat_wait_event()
 aswe
WHERE lssw.event_id=aswe.event_id and aswe.event_name='LimitlessTaskScheduler';

 subcluster_id | subcluster_type | type_id | event_id  | waits  |  wait_time   |
 event_name
---------------+-----------------+---------+-----------+--------+--------------
+------------------------
             1 | router          |      12 | 201326607 | 677068 | 616942216307 |
 LimitlessTaskScheduler
             2 | router          |      12 | 201326607 | 678586 | 616939897111 |
 LimitlessTaskScheduler
             3 | shard           |      12 | 201326607 | 756640 | 616965545172 |
 LimitlessTaskScheduler
             4 | shard           |      12 | 201326607 | 755184 | 616958057620 |
 LimitlessTaskScheduler
             5 | shard           |      12 | 201326607 | 757522 | 616963183539 |
 LimitlessTaskScheduler
(5 rows)
```

limitless_terminate_session

Die Funktion `limitless_terminate_session` arbeitet ähnlich wie `pg_terminate_backend`, versucht aber, alle Backend-Prozesse im Zusammenhang mit der angegebenen ID einer verteilten Sitzung zu beenden, indem sie ein `SIGTERM` (Beendigungssignal) sendet.

Es wird der folgende Eingabeparameter verwendet:

- `distributed_session_id` (Text): Die ID der verteilten Sitzung, die beendet werden soll.

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `pid` (Text): Die ID des Backend-Prozesses.
- `success` (Boolean): Ob der Prozess erfolgreich beendet wurde.

Das folgende Beispiel zeigt, wie die Funktion `limitless_terminate_session` verwendet werden kann.

```
SELECT * FROM rds_aurora.limitless_terminate_session('940CD5C81E3C796B');

 subcluster_id |  pid  | success
---------------+-------+---------
             1 | 26920 | t
(1 row)
```

limitless_wait_report

Die Funktion `limitless_wait_report` gibt die Aktivität „Warteereignis“ über einen bestimmten Zeitraum von allen Knoten zurück. Die Spalten `subcluster_id` und `subcluster_type` zeigen, von welchem Knoten die Daten stammen.

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `subcluster_type` (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: `router` oder `shard`.

Die restlichen Spalten sind dieselben wie in `aurora_wait_report`.

Das folgende Beispiel zeigt, wie die Funktion `limitless_wait_report` verwendet werden kann.

```
postgres_limitless=> select * from rds_aurora.limitless_wait_report();

 subcluster_id | subcluster_type | type_name | event_name | waits | wait_time |
 ms_per_wait | waits_per_xact | ms_per_xact
---------------+-----------------+-----------+------------+-------+-----------
+-------------+--------------- +-------------
             1 | router          | Client    | ClientRead |    57 | 741550.14 |
 13009.652 |           0.19 |    2505.237
             5 | shard           | Client    | ClientRead |    54 | 738897.68 |
 13683.290 |           0.18 |    2496.276
             4 | shard           | Client    | ClientRead |    54 | 738859.53 |
 13682.584 |           0.18 |    2496.147
             2 | router          | Client    | ClientRead |    53 | 719223.64 |
 13570.257 |           0.18 |    2429.810
             3 | shard           | Client    | ClientRead |    54 | 461720.40 |
 8550.378 |           0.18 |    1559.86
```

Ansichten für Aurora PostgreSQL Limitless Database

In der folgenden Tabelle sehen Sie die neuen Ansichten für Aurora PostgreSQL Limitless Database.

> **Note**
>
> Die in dieser Tabelle aufgeführten Ansichten befinden sich im Schema `rds_aurora`. Wenn Sie eine Limitless-Database-Ansicht verwenden, stellen Sie sicher, dass Sie den vollqualifizierten Objektnamen angeben: `rds_aurora.object_name`.

Ansicht für Aurora PostgreSQL Limitless Database	Entsprechende Ansicht für Aurora PostgreSQL
limitless_database	pg_database
limitless_locks	pg_locks
limitless_stat_activity	pg_stat_activity
limitless_stat_all_indexes	pg_stat_all_indexes
limitless_stat_all_tables	pg_stat_all_tables
limitless_stat_database	pg_stat_database
limitless_stat_progress_vacuum	pg_stat_progress_vacuum
limitless_stat_statements	pg_stat_statements
limitless_stat_subclusters	Keine
limitless_stat_statements_info	pg_stat_statements_info
limitless_statio_all_indexes	pg_statio_all_indexes
limitless_statio_all_tables	pg_statio_all_tables
limitless_tables	pg_tables
limitless_table_collocations	Keine

Ansicht für Aurora PostgreSQL Limitless Database	Entsprechende Ansicht für Aurora PostgreSQL
limitless_table_collocation_distributions	Keine

Die folgenden Beispiele enthalten Einzelheiten zu den Ansichten für Aurora PostgreSQL Limitless Database. Weitere Informationen zu PostgreSQL-Ansichten finden Sie unter Anzeigen von Statistiken in der PostgreSQL-Dokumentation.

> **Note**
>
> Einige Statistikansichten können uneinheitliche Ergebnisse liefern, wenn Transaktionen im Gange sind.

limitless_database

Diese Ansicht enthält Informationen über die verfügbaren Datenbanken in der DB-Shard-Gruppe. Zum Beispiel:

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, oid, datname, datacl
 FROM rds_aurora.limitless_database;

 subcluster_id | subcluster_type |  oid  |      datname       |
                                 datacl

---------------+-----------------+-------+--------------------
+-------------------------------------------------------------------------------------------
 2             | router          |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 2             | router          |     5 | postgres           |
 2             | router          | 16384 | rdsadmin           |
 {rdsadmin=CTc/rdsadmin,rds_aurora_limitless_metadata_admin=c/
rdsadmin,rds_aurora_limitless_heat_mgmt_admin=c/rdsadmin}
 2             | router          | 16477 | postgres_limitless |
 2             | router          |     1 | template1          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
 6             | shard           |     4 | template0          | {=c/
rdsadmin,rdsadmin=CTc/rdsadmin}
```

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters (Knoten)
- `subcluster_type` (Text): Der Typ des Subclusters (Knoten), Router oder Shard

Die restlichen Spalten sind dieselben wie in `pg_database`.

limitless_locks

Diese Ansicht enthält eine Zeile pro Prozess und Knoten. Sie bietet Zugriff auf Informationen über die Sperren, die aktive Prozesse im Datenbankserver aufrecht erhalten.

Example Beispiel für das Erstellen einer Sperre mit zwei Transaktionen

In diesem Beispiel führen wir zwei Transaktionen gleichzeitig auf zwei Routern aus.

```
# Transaction 1 (run on router 1)
BEGIN;
SET search_path = public;
SELECT * FROM customers;
INSERT INTO customers VALUES (400,'foo','bar');

# Transaction 2 (run on router 2)
BEGIN;
SET search_path = public;
ALTER TABLE customers ADD COLUMN phone VARCHAR;
```

Die erste Transaktion wird ausgeführt. Nachfolgende Transaktionen müssen warten, bis die erste Transaktion abgeschlossen ist. Daher wird die zweite Transaktion durch eine Sperre blockiert. Um die Ursache dafür zu ermitteln, führen wir einen Befehl aus, indem wir `limitless_locks` mit `limitless_stat_activity` vereinen.

```
# Run on router 2
SELECT distributed_session_id, state, usename, query, query_start
FROM rds_aurora.limitless_stat_activity
WHERE distributed_session_id in (
SELECT distributed_session_id
FROM rds_aurora.limitless_locks
WHERE relname = 'customers'
);

 distributed_session_id | state               | usename                | query
                                  | query_start
```

```
-----------------------+--------------------+-------------------------
+-------------------------------- -------------+-----------------------------
 47BDE66E9A5E8477       | idle in transaction | limitless_metadata_admin | INSERT INTO
 customers VALUES (400,'foo','bar'); | 2023-04-13 17:44:45.152244+00
 2AD7F370202D0FA9       | active              | limitless_metadata_admin | ALTER TABLE
 customers ADD COLUMN phone VARCHAR; | 2023-04-13 17:44:55.113388+00
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 47BDE66E9A5E8477       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
 2AD7F370202D0FA9       |                     | limitless_auth_admin     |
 <insufficient privilege>                       |
(6 rows)
```

Example Beispiel für das explizite Erstellen einer Sperre

In diesem Beispiel erstellen wir explizit eine Sperre und verwenden dann die Ansicht `limitless_locks`, um die Sperren zu sehen (einige Spalten werden weggelassen).

```
BEGIN;
SET search_path = public;
LOCK TABLE customers IN ACCESS SHARE MODE;
SELECT * FROM rds_aurora.limitless_locks WHERE relname = 'customers';

 subcluster_id | subcluster_type | distributed_session_id | locktype |      datname
   | relnspname |  relname  | virtualtransaction |  pid  |      mode
---------------+-----------------+------------------------+----------
+--------------------+------------+ ----------+--------------------+-------
+-----------------
             1 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600787          | 59564 |
 AccessShareLock
             2 | router          | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 28/600405          | 67130 |
 AccessShareLock
             3 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 15/473401          | 27735 |
 AccessShareLock
             4 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473524          | 27734 |
 AccessShareLock
```

```
             5 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/472935          | 27737 |
 AccessShareLock
             6 | shard           | 7207702F862FC937       | relation |
 postgres_limitless | public     | customers | 13/473015          | 48660 |
 AccessShareLock
(6 rows)
```

limitless_stat_activity

Diese Ansicht enthält eine Zeile pro Prozess und Knoten. Sie kann verwendet werden, um den Gesamtzustand des Systems zu verfolgen und lange dauernde Prozesse zu klassifizieren. Zum Beispiel:

```
postgres=# SELECT
    subcluster_id,
    subcluster_type,
    distributed_session_id,
    distributed_session_state,
    datname,
    distributed_query_id,
    is_sso_query
FROM
    rds_aurora.limitless_stat_activity
WHERE
    distributed_session_id in ('D2470C97E3D07E06', '5A3CD7B8E5FD13FF')
    order by  distributed_session_id;

 subcluster_id | subcluster_type | distributed_session_id | distributed_session_state |
      datname       | distributed_query_id | is_sso_query
---------------+-----------------+------------------------+---------------------------
+--------------------+----------------------+-------------
 2             | router          | 5A3CD7B8E5FD13FF       | coordinator               |
 postgres_limitless |                      | f
 3             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 4             | shard           | 5A3CD7B8E5FD13FF       | participant               |
 postgres_limitless |  6808291725541680947 |
 2             | router          | D2470C97E3D07E06       | coordinator               |
 postgres_limitless |                      | t
 3             | shard           | D2470C97E3D07E06       | participant               |
 postgres_limitless |  4058400544464210222 |
(5 rows)
```

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `subcluster_type` (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: `router` oder `shard`.
- `distributed_session_id` (Text): Die ID der verteilten Sitzung, zu der dieser Prozess gehört.
- `distributed_session_state` (Text): Ob es sich um einen Koordinator, einen Teilnehmer oder einen eigenständigen/nicht verteilten Prozess handelt (angezeigt als `NULL`).
- `datname` (Text): Die Datenbank, mit der dieser Prozess verbunden ist.
- `distributed_query_id` (bigint): Die Abfrage-ID der übergeordneten Abfrage vom Koordinatorknoten. Diese Spalte lautet `NULL`, wenn es sich um die übergeordnete Abfrage handelt. Der Koordinatorknoten gibt die ID der verteilten Abfrage an die teilnehmenden Knoten weiter. Für die Teilnehmerknoten sind die Werte für die ID der verteilten Abfrage und die ID der Abfrage also unterschiedlich.
- `is_sso_query` (Text): Dadurch erfahren wir, ob die Abfrage für einen einzelnen Shard optimiert ist oder nicht.

Die restlichen Spalten sind dieselben wie in `pg_stat_activity`.

limitless_stat_all_indexes

Diese Ansicht enthält Auslastungsstatistiken für Indizes in der DB-Shard-Gruppe. Zum Beispiel:

```
postgres_limitless=> SELECT schemaname, relname, indexrelname, idx_scan
  FROM rds_aurora.limitless_stat_all_indexes
  WHERE relname LIKE 'orders_ts%' ORDER BY indexrelname LIMIT 10;

 schemaname |    relname     |    indexrelname     | idx_scan
------------+----------------+---------------------+----------
 ec_sample  | orders_ts00001 | orders_ts00001_pkey |   196801
 ec_sample  | orders_ts00002 | orders_ts00002_pkey |   196703
 ec_sample  | orders_ts00003 | orders_ts00003_pkey |   196376
 ec_sample  | orders_ts00004 | orders_ts00004_pkey |   197966
 ec_sample  | orders_ts00005 | orders_ts00005_pkey |   195301
 ec_sample  | orders_ts00006 | orders_ts00006_pkey |   195673
 ec_sample  | orders_ts00007 | orders_ts00007_pkey |   194475
 ec_sample  | orders_ts00008 | orders_ts00008_pkey |   191694
 ec_sample  | orders_ts00009 | orders_ts00009_pkey |   193744
 ec_sample  | orders_ts00010 | orders_ts00010_pkey |   195421
```

```
(10 rows)
```

limitless_stat_all_tables

Diese Ansicht enthält Statistiken über alle Tabellen in der aktuellen Datenbank in der DB-Shard-Gruppe. Dies ist nützlich, wenn Sie Bereinigungsvorgänge und DML-Operationen (Data Manipulation Language) verfolgen möchten. Zum Beispiel:

```
postgres_limitless=> SELECT subcluster_id, subcluster_type, relname,
 n_ins_since_vacuum, n_tup_ins, last_vacuum
  FROM rds_aurora.limitless_stat_all_tables
  WHERE relname LIKE 'orders_ts%' ORDER BY relname LIMIT 10;

 subcluster_id | subcluster_type |    relname     | n_ins_since_vacuum | n_tup_ins |
 last_vacuum
---------------+-----------------+----------------+--------------------+-----------
+-------------
 5             | shard           | orders_ts00001 |              34779 |    196083 |
 5             | shard           | orders_ts00002 |              34632 |    194721 |
 5             | shard           | orders_ts00003 |              34950 |    195965 |
 5             | shard           | orders_ts00004 |              34745 |    197283 |
 5             | shard           | orders_ts00005 |              34879 |    195754 |
 5             | shard           | orders_ts00006 |              34340 |    194605 |
 5             | shard           | orders_ts00007 |              33779 |    192203 |
 5             | shard           | orders_ts00008 |              33826 |    191293 |
 5             | shard           | orders_ts00009 |              34660 |    194117 |
 5             | shard           | orders_ts00010 |              34569 |    195560 |
(10 rows)
```

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `subcluster_type` (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: `router` oder `shard`.
- `relname` (Name): Der Name der Tabelle.

Die restlichen Spalten sind dieselben wie in `pg_stat_all_tables`.

limitless_stat_database

Diese Ansicht enthält Statistiken über alle Datenbanken in der DB-Shard-Gruppe. Gibt eine Zeile pro Datenbank und Knoten zurück. Zum Beispiel:

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    datname,
    blks_read,
    blks_hit
FROM
    rds_aurora.limitless_stat_database
WHERE
    datname='postgres_limitless';
 subcluster_id | subcluster_type |      datname       | blks_read | blks_hit
---------------+-----------------+--------------------+-----------+----------
             1 | router          | postgres_limitless |       484 | 34371314
             2 | router          | postgres_limitless |       673 | 33859317
             3 | shard           | postgres_limitless |      1299 | 17749550
             4 | shard           | postgres_limitless |      1094 | 17492849
             5 | shard           | postgres_limitless |      1036 | 17485098
             6 | shard           | postgres_limitless |      1040 | 17437257
(6 rows)
```

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `subcluster_type` (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: `router` oder `shard`.
- `datname` (Name): Der Name der Datenbank.

Die restlichen Spalten sind dieselben wie in `pg_stat_database`.

limitless_stat_progress_vacuum

Diese Ansicht enthält Informationen zu laufenden Bereinigungsvorgängen. Zum Beispiel:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_progress_vacuum;

-[ RECORD 1 ]----------+-----------------
subcluster_id          | 3
subcluster_type        | shard
distributed_session_id | A56D96E2A5C9F426
pid                    | 5270
datname                | postgres
nspname                | public
relname                | customer_ts2
```

```
phase                   | vacuuming heap
heap_blks_total         | 130500
heap_blks_scanned       | 100036
heap_blks_vacuumed      | 0
index_vacuum_count      | 0
max_dead_tuples         | 11184810
num_dead_tuples         | 0

-[ RECORD 2 ]----------+------------------
subcluster_id           | 3
subcluster_type         | shard
distributed_session_id | 56DF26A89EC23AB5
pid                     | 6854
datname                 | postgres
nspname                 | public
relname                 | sales_ts1
phase                   | vacuuming heap
heap_blks_total         | 43058
heap_blks_scanned       | 24868
heap_blks_vacuumed      | 0
index_vacuum_count      | 0
max_dead_tuples         | 8569523
num_dead_tuples         | 0
```

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `subcluster_type` (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: `router` oder `shard`.
- `distributed_session_id` (Text): Die ID der Sitzung, die den Bereinigungsvorgang initiiert hat.
- `datname` (Name): Die Datenbank, in der die Bereinigung durchgeführt wird.
- `nspname` (Name): Der Name des Schemas der Tabelle, die bereinigt wird. Der Wert lautet `null`, wenn sich die Tabelle, die bereinigt wird, nicht in derselben Datenbank befindet wie die, mit der der Benutzer verbunden ist.
- `relname` (Name): Der Name der Tabelle, die bereinigt wird. Der Wert lautet `null`, wenn sich die Tabelle, die bereinigt wird, nicht in derselben Datenbank befindet wie die, mit der der Benutzer verbunden ist.

Die restlichen Spalten sind dieselben wie in `pg_stat_progress_vacuum`.

limitless_stat_statements

Diese Ansicht bietet die Möglichkeit, Statistiken zur Planung und Ausführung aller SQL-Anweisungen zu verfolgen, die auf allen Knoten ausgeführt werden.

> **Note**
>
> Sie müssen die Erweiterung pg_stat_statements installieren, um die Ansicht `limitless_stat_statements` verwenden zu können.
>
> ```
> -- CREATE EXTENSION must be run by a superuser
> CREATE EXTENSION pg_stat_statements;
>
> -- Verify that the extension is created on all nodes in the DB shard group
> SELECT distinct node_id
>     FROM rds_aurora.limitless_stat_statements
>     LIMIT 10;
> ```

Das folgende Beispiel veranschaulicht die Verwendung der Ansicht `limitless_stat_statements`.

```
postgres_limitless=> SELECT
 subcluster_id,
 subcluster_type,
 distributedqueryid,
 username,
 dbname,
 sso_calls
FROM
 rds_aurora.limitless_stat_statements;

 subcluster_id | subcluster_type |  distributedqueryid  |             username
       |       dbname       | sso_calls
---------------+-----------------+----------------------
+-------------------------------------+--------------------+-----------
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
```

```
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         0
 2             | router          |                      | postgres
       | postgres_limitless |         1
 3             | shard           | -7975178695405682176 | postgres
       | postgres_limitless |
[...]
```

Es werden folgende Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.
- `subcluster_type` (Text): Der Typ des Subclusters, zu dem dieser Prozess gehört: `router` oder `shard`.
- `distributedqueryid` (bigint): Die Abfrage-ID der übergeordneten Abfrage vom Koordinatorknoten. Diese Spalte lautet `NULL`, wenn es sich um die übergeordnete Abfrage handelt. Der Koordinatorknoten gibt die ID der verteilten Abfrage an die teilnehmenden Knoten weiter. Für die Teilnehmerknoten sind die Werte für die ID der verteilten Abfrage und die ID der Abfrage also unterschiedlich.
- `username` (Name): Der Benutzer, der die Anweisung abgefragt hat.
- `dbname` (Name): Die Datenbank, in der die Abfrage ausgeführt wurde.
- `sso_calls` (Name): Gibt an, wie oft die Anweisung für einen einzelnen Shard optimiert wurde.

Die restlichen Spalten sind dieselben wie in pg_stat_statements.

limitless_stat_statements_info

Diese Ansicht enthält Statistiken für die Ansicht `limitless_stat_statements`. Jede Zeile enthält Daten für die Ansicht pg_stat_statements_info von jedem Knoten. Die Spalte `subcluster_id` identifiziert jeden Knoten.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_statements_info;

 subcluster_id | subcluster_type | dealloc |          stats_reset
---------------+-----------------+---------+-------------------------------
             1 | router          |       0 | 2023-06-30 21:22:09.524781+00
             2 | router          |       0 | 2023-06-30 21:21:40.834111+00
             3 | shard           |       0 | 2023-06-30 21:22:10.709942+00
             4 | shard           |       0 | 2023-06-30 21:22:10.740179+00
```

```
             5 | shard           |       0 | 2023-06-30 21:22:10.774282+00
             6 | shard           |       0 | 2023-06-30 21:22:10.808267+00
(6 rows)
```

Es wird folgender Ausgabeparameter verwendet:

- `subcluster_id` (Text): Die ID des Subclusters, zu dem dieser Prozess gehört.

Die restlichen Spalten sind dieselben wie in pg_stat_statements_info.

limitless_stat_subclusters

Diese Ansicht enthält Netzwerkstatistiken zwischen Routern und anderen Knoten. Sie enthält eine Zeile pro Paar aus Router und anderem Knoten, zum Beispiel:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_stat_subclusters;

 orig_subcluster | orig_instance_az | dest_subcluster | dest_instance_az |
 latency_us |       latest_collection       | failed_requests | received_bytes |
 sent_bytes | same_az_requests | cross_az_requests |     stat_reset_timestamp
-----------------+------------------+-----------------+------------------
+------------+-------------------------------+-----------------+---------------
+------------+------------------+-------------------+------------------------------
 3               | us-west-2b       | 2               | us-west-2a       |
 847 | 2024-10-07 17:25:39.518617+00 |               0 |       35668633 |   92090171
 |                0 |           302787 | 2024-10-05 12:39:55.239675+00
 3               | us-west-2b       | 4               | us-west-2b       |
 419 | 2024-10-07 17:25:39.546376+00 |               0 |      101190464 |  248795719
 |           883478 |                0 | 2024-10-05 12:39:55.231218+00
 3               | us-west-2b       | 5               | us-west-2c       |
    1396 | 2024-10-07 17:25:39.52122+00  |               0 |       72864849 |
 172086292 |               0 |           557726 | 2024-10-05 12:39:55.196412+00
 3               | us-west-2b       | 6               | us-west-2c       |
 729 | 2024-10-07 17:25:39.54828+00  |               0 |       35668584 |   92090171
 |                0 |           302787 | 2024-10-05 12:39:55.247334+00
 3               | us-west-2b       | 7               | us-west-2a       |
    1702 | 2024-10-07 17:25:39.545307+00 |               0 |       71699576 |
 171634844 |               0 |           556278 | 2024-10-05 12:39:52.715168+00
 2               | us-west-2a       | 3               | us-west-2b       |
 868 | 2024-10-07 17:25:40.293927+00 |               0 |       35659611 |   92011872
 |                0 |           302817 | 2024-10-05 12:39:54.420758+00
 2               | us-west-2a       | 4               | us-west-2b       |
 786 | 2024-10-07 17:25:40.296863+00 |               0 |      102437253 |  251838024
 |                0 |           895060 | 2024-10-05 12:39:54.404081+00
```

```
2               | us-west-2a       | 5               | us-west-2c       |
   1232 | 2024-10-07 17:25:40.292021+00 |               0 |       71990027 |
168828110 |                0 |            545453 | 2024-10-05 12:39:36.769549+00
```

Es werden folgende Ausgabeparameter verwendet:

- `orig_subcluster` (Text): Die ID des Routers, der Absender der Kommunikation ist
- `orig_subcluster_az` (Text): Die Availability Zone (AZ) des Absender-Routers
- `dest_subcluster` (Text): Die ID des Zielknotens
- `dest_subcluster_az` (Text): Die zuletzt erfasste AZ des Zielknotens
- `latency_us` (bigint): Die zuletzt erfasste Netzwerklatenz zwischen Knoten in Mikrosekunden. Der Wert lautet `0`, wenn der Knoten nicht erreichbar ist.
- `latest_collection` (Zeitstempel): Der Zeitstempel der letzten Erfassung von AZ und Latenz für den Zielknoten
- `failed_requests` (bigint): Die Gesamtzahl der fehlgeschlagenen internen Anfragen
- `received_bytes` (bigint): Die geschätzte kumulative Anzahl von Byte, die von diesem Knoten empfangen wurden
- `sent_bytes` (bigint): Die geschätzte kumulative Anzahl von Byte, die an diesen Knoten gesendet wurden
- `same_az_requests` (bigint): Die kumulative Anzahl interner DB-Anfragen an diesen Knoten, wenn er sich in derselben AZ wie der Absender-Router befindet
- `cross_az_requests` (bigint): Die kumulative Anzahl interner DB-Anfragen an diesen Knoten, wenn er sich in einer anderen AZ als der Absender-Router befindet
- `stat_reset_timestamp` (Zeitstempel): Der Zeitstempel des Zeitpunkts, zu dem die kumulativen Statistiken für diese Ansicht zuletzt zurückgesetzt wurden

limitless_statio_all_indexes

Diese Ansicht enthält Ein/Ausgabe (I/O)-Statistiken für alle Indizes in der DB-Shard-Gruppe. Zum Beispiel:

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_statio_all_indexes WHERE
 relname like'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      |
 indexrelname             | idx_blks_read | idx_blks_hit
---------------+-----------------+------------+-------------------
+------------------------------------+ --------------+--------------
```

```
             3 | shard           | public     | customers_ts00002 |
 customers_ts00002_customer_name_idx |             1 |            0
             3 | shard           | public     | customers_ts00001 |
 customers_ts00001_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00003 |
 customers_ts00003_customer_name_idx |             1 |            0
             4 | shard           | public     | customers_ts00004 |
 customers_ts00004_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00005 |
 customers_ts00005_customer_name_idx |             1 |            0
             5 | shard           | public     | customers_ts00006 |
 customers_ts00006_customer_name_idx |             1 |            0
             6 | shard           | public     | customers_ts00007 |
 customers_ts00007_customer_name_idx |             1 |            0
             6 | shard           | public     | customers_ts00008 |
 customers_ts00008_customer_name_idx |             1 |            0
(8 rows)
```

limitless_statio_all_tables

Diese Ansicht enthält Ein/Ausgabe (I/O)-Statistiken für alle Tabellen in der DB-Shard-Gruppe. Zum Beispiel:

```
postgres_limitless=> SELECT
    subcluster_id,
    subcluster_type,
    schemaname,
    relname,
    heap_blks_read,
    heap_blks_hit
FROM
    rds_aurora.limitless_statio_all_tables
WHERE
    relname LIKE 'customers_ts%';

 subcluster_id | subcluster_type | schemaname |      relname      | heap_blks_read |
 heap_blks_hit
---------------+-----------------+------------+-------------------+----------------
+---------------
             3 | shard           | public     | customers_ts00002 |            305 |
      57780
             3 | shard           | public     | customers_ts00001 |            300 |
      56972
```

```
             4 | shard           | public     | customers_ts00004 |            302 |
      57291
             4 | shard           | public     | customers_ts00003 |            302 |
      57178
             5 | shard           | public     | customers_ts00006 |            300 |
      56932
             5 | shard           | public     | customers_ts00005 |            302 |
      57386
             6 | shard           | public     | customers_ts00008 |            300 |
      56881
             6 | shard           | public     | customers_ts00007 |            304 |
      57635
(8 rows)
```

limitless_tables

Diese Ansicht enthält Informationen zu Tabellen in Aurora PostgreSQL Limitless Database.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_tables;

 table_gid | local_oid | schema_name | table_name  | table_status | table_type  |
 distribution_key
-----------+-----------+-------------+-------------+--------------+-------------
+------------------
         5 |     18635 | public      | placeholder | active       | placeholder |
         6 |     18641 | public      | ref         | active       | reference   |
         7 |     18797 | public      | orders      | active       | sharded     | HASH
 (order_id)
         2 |     18579 | public      | customer    | active       | sharded     | HASH
 (cust_id)
(4 rows)
```

limitless_table_collocations

Diese Ansicht enthält Informationen über zusammengefasste Sharded-Tabellen.

Im folgenden Beispiel sind die Tabellen `orders` und `customers` zusammengefasst, und die Tabellen `users` und `followers` ebenso. Zusammengefasste Tabellen haben die gleiche `collocation_id`.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_table_collocations ORDER BY
 collocation_id;
```

```
 collocation_id | schema_name | table_name
----------------+-------------+------------
              2 | public      | orders
              2 | public      | customers
              5 | public      | users
              5 | public      | followers
(4 rows)
```

limitless_table_collocation_distributions

Diese Ansicht zeigt die Schlüsselverteilung für jede Kollokation.

```
postgres_limitless=> SELECT * FROM
 rds_aurora.limitless_table_collocation_distributions ORDER BY collocation_id,
 lower_bound;

 collocation_id | subcluster_id |     lower_bound      |     upper_bound
----------------+---------------+----------------------+----------------------
              2 |             6 | -9223372036854775808 | -4611686018427387904
              2 |             5 | -4611686018427387904 |                    0
              2 |             4 |                    0 |  4611686018427387904
              2 |             3 |  4611686018427387904 |  9223372036854775807
              5 |             6 | -9223372036854775808 | -4611686018427387904
              5 |             5 | -4611686018427387904 |                    0
              5 |             4 |                    0 |  4611686018427387904
              5 |             3 |  4611686018427387904 |  9223372036854775807
(8 rows)
```

Warteereignisse für Aurora PostgreSQL Limitless Database

Ein Warteereignis in Aurora PostgreSQL weist auf eine Ressource hin, auf die eine Sitzung wartet, z. B. Ein-/Ausgabe (I/O) und Sperren. Warteereignisse sind hilfreich, um herauszufinden, warum Sitzungen auf Ressourcen warten, und um Engpässe zu identifizieren. Weitere Informationen finden Sie unter Aurora PostgreSQL-Warteereignisse.

Aurora PostgreSQL Limitless Database verfügt über eigene Warteereignisse, die sich auf Router und Shards beziehen. Viele davon beziehen sich auf Router, die darauf warten, dass Shards bestimmte Aufgaben erledigen. Shard-Warteereignisse enthalten Details zu Aufgaben, die gerade ausgeführt werden.

Themen

- Abfragen von Warteereignissen
- Warteereignisse für Limitless Database

Abfragen von Warteereignissen

Sie können die Ansicht limitless_stat_activity verwenden, um Warteereignisse abzufragen, wie im folgenden Beispiel gezeigt.

```
SELECT wait_event FROM rds_aurora.limitless_stat_activity WHERE
 wait_event_type='AuroraLimitless';

      wait_event
----------------------
 RemoteStatementSetup
 RemoteStatementSetup
(2 rows)
```

Sie können die Funktion `aurora_stat_system_waits` auch verwenden, um die Anzahl der Warteereignisse und die Gesamtzeit aufzulisten, die für jedes Warteereignis aufgewendet wurde, wie im folgenden Beispiel gezeigt.

```
postgres_limitless=> SELECT type_name,event_name,waits,wait_time
    FROM aurora_stat_system_waits()
    NATURAL JOIN aurora_stat_wait_event()
    NATURAL JOIN aurora_stat_wait_type()
    WHERE type_name='AuroraLimitless'
```

```
    ORDER BY wait_time DESC;

    type_name    |       event_name          |  waits  |  wait_time
-----------------+---------------------------+---------+-------------
 AuroraLimitless | RemoteStatementSetup      |    7518 | 75236507897
 AuroraLimitless | RemoteStatementExecution  |      40 |      132986
 AuroraLimitless | Connect                   |       5 |        1453
(3 rows)
```

Warteereignisse für Limitless Database

Die folgenden Wareereignisse gelten für Aurora PostgreSQL Limitless Database. Sie können diese Warteereignisse überwachen, um Engpässe bei der Verarbeitung von Aurora PostgreSQL Limitless Database zu identifizieren.

Themen

- Warteereignis „IO:TwophaseFilePoolWrite“
- Warteereignis „IO:TwophaseFilePoolRead“
- Warteereignis „AuroraLimitless:Connect“
- Warteereignis „AuroraLimitless:AsyncConnect“
- Warteereignis „AuroraLimitless:RemoteStatementSetup“
- Warteereignis „AuroraLimitless:RemoteDDLExecution“
- Warteereignis „AuroraLimitless:RemoteStatementExecution“
- Warteereignis „AuroraLimitless:FetchRemoteResults“
- Warteereignis „AuroraLimitless:AsyncGetInitialResponse“
- Warteereignis „AuroraLimitless:AsyncGetNextResponse“
- Warteereignis „AuroraLimitless:AbortedCommandCleanup“
- Warteereignis „AuroraLimitless:DistributedCommitPrepare“
- Warteereignis „AuroraLimitless:DistributedCommit“
- Warteereignis „AuroraLimitless:DistributedCommitPrepareThrottle“
- Warteereignis „AuroraLimitless:PreparedTransactionResolution“
- Warteereignis „AuroraLimitless:SendPreparedTransactionOutcome“
- Warteereignis „AuroraLimitless:CommitClockBarrier“
- Warteereignis „AuroraLimitless:SnapshotClockBarrier“

- Warteereignis „AuroraLimitless:ReaderSnapshotClockBarrier“
- Warteereignis „AuroraLimitless:GatherDistributedDeadlockGraph“
- Warteereignis „AuroraLimitless:DistributedDeadlockDetection“
- Warteereignis „AuroraLimitless:DistributedDeadlockAbort“
- Warteereignis „AuroraLimitless:GatherRemoteStats“
- Warteereignis „AuroraLimitless:GlobalSequenceRefresh“
- Warteereignis „AuroraLimitless:GlobalVacuumTimeExchange“
- Warteereignis „AuroraLimitless:DistributedTransactionMonitorGather“
- Warteereignis „AuroraLimitlessActivity:AdminTaskSchedulerMain“
- Warteereignis „AuroraLimitlessActivity:AdminTaskExecutorMain“
- Warteereignis „AuroraLimitlessActivity:AdminTaskMonitorMain“
- Warteereignis „AuroraLimitlessActivity:DatabaseCleanupMonitorMain“
- Warteereignis „AuroraLimitlessActivity:TopologyCleanupMonitorMain“
- Warteereignis „AuroraLimitlessActivity:ToplogyChangeMonitorMain“
- Warteereignis „AuroraLimitlessActivity:DistributedTransactionMonitorMain“
- Warteereignis „AuroraLimitlessActivity:GlobalVacuumMonitorMain“

Warteereignis „IO:TwophaseFilePoolWrite“

Warten auf das Schreiben einer zweiphasigen Statusdatei innerhalb des zweiphasigen Statusdateipools. Dies ist ein Aurora-spezifisches Ereignis.

Ursachen

Prozesse, die einen `PREPARED TRANSACTION`-Befehl ausführen, einschließlich Teilnehmern an einer verteilten Transaktion von Limitless Database, müssen den Transaktionsstatus in einer zweiphasigen Datei beibehalten. Aurora verwendet einen Dateipool, um die Leistung dieses Vorgangs zu verbessern.

Aktion

Da es sich um einen synchronen I/O-Schreibvorgang handelt, hat eine hohe Latenz in diesem Fall ähnliche Ursachen wie `IO:XactSync` und kann auf dieselbe Weise untersucht werden. Wenn Sie Limitless Database verwenden, müssen Sie möglicherweise die Anzahl der ausgeführten verteilten Transaktionen reduzieren.

Warteereignis „IO:TwophaseFilePoolRead“

Warten auf das Lesen einer zweiphasigen Statusdatei innerhalb des zweiphasigen Statusdateipools.

Ursachen

Prozesse, die einen `COMMIT PREPARED`-Befehl für eine zuvor vorbereitete Transaktion ausführen, einschließlich Teilnehmern an einer verteilten Transaktion von Limitless Database, müssen möglicherweise den zuvor gespeicherten Transaktionsstatus aus einer zweiphasigen Datei lesen. Aurora verwendet einen Dateipool, um die Leistung dieses Vorgangs zu verbessern.

Aktion

Dies ist ein I/O-Lesevorgang. Daher hat eine hohe Latenz bei diesem Ereignis ähnliche Ursachen wie `IO:DataFileRead` und kann auf die gleiche Weise untersucht werden. Wenn Sie Limitless Database verwenden, müssen Sie möglicherweise die Anzahl der ausgeführten verteilten Transaktionen reduzieren.

Warteereignis „AuroraLimitless:Connect“

Der Prozess wartet darauf, dass eine Verbindung zu einem anderen Knoten im Cluster hergestellt wird.

Ursachen

Verbindungen werden zwischen Prozessen und Remote-Knoten hergestellt, um Abfragen, verteilte Transaktionen und DDLs auszuführen.

Aktion

Reduzieren Sie die Anzahl der gleichzeitigen Verbindungen zum Cluster oder optimieren Sie die Verwendung von Cross-Shard-Abfragen.

Warteereignis „AuroraLimitless:AsyncConnect“

Dieses Ereignis ähnelt `Connect`, stellt aber einen Prozess dar, der darauf wartet, dass parallele Verbindungen zu einer Gruppe von Knoten hergestellt werden.

Ursachen

Der parallele Verbindungsaufbau erfolgt am häufigsten bei der Ausführung von DDL-Anweisungen.

Aktion

Reduzieren Sie die Anzahl der DDL-Anweisungen oder kombinieren Sie mehrere DDLs in derselben Sitzung, um die Wiederverwendung von Verbindungen zu verbessern.

Warteereignis „AuroraLimitless:RemoteStatementSetup“

Der Prozess wartet auf die Einrichtung der Remoteabfrageausführung, wie z. B. das Öffnen oder Schließen des Cursors oder die Erstellung einer vorbereiteten Anweisung.

Ursachen

Dieses Warteereignis nimmt mit der Anzahl der Scans in Sharde-Tabellen zu, bei denen die Anweisung nicht für ein einzelnes Shard optimiert werden konnte.

Aktion

Optimieren Sie Abfragen, um die Anzahl der Scanvorgänge zu reduzieren oder die Eignung für die Single-Shard-Optimierung zu erhöhen.

Warteereignis „AuroraLimitless:RemoteDDLExecution“

Der Prozess wartet darauf, dass ein Remote-DDL-Befehl abgeschlossen wird.

Ursachen

Wenn Sie einen DDL-Befehl für eine DB-Shard-Gruppe ausgeben, muss er an andere Router- und Shard-Knoten verteilt werden, bevor der Vorgang bestätigt wird. Einige DDL-Vorgänge können lange dauern, da Daten an Schemaänderungen angepasst werden müssen.

Aktion

Identifizieren Sie DDL-Befehle mit langer Laufzeit, damit Sie sie optimieren können.

Warteereignis „AuroraLimitless:RemoteStatementExecution“

Der Prozess wartet darauf, dass ein Remote-Befehl abgeschlossen wird.

Ursachen

Ein SQL-Befehl wird auf einem Remote-Knoten ausgeführt. Dieses Ereignis tritt häufig bei der internen Kommunikation auf, z. B. bei `auto_analyze` Heartbeat-Überprüfungen.

Aktion

Identifizieren Sie Befehle mit langer Laufzeit in der Ansicht „limitless_stat_statements“. In vielen Fällen handelt es sich hierbei um ein erwartetes Ereignis, insbesondere bei Hintergrundprozessen oder internen Prozessen, sodass keine Maßnahmen erforderlich sind.

Warteereignis „AuroraLimitless:FetchRemoteResults“

Ein Prozess wartet darauf, Zeilen von einem Remote-Knoten abzurufen.

Ursachen

Dieses Warteereignis kann zunehmen, wenn eine große Anzahl von Zeilen aus einer Remote-Tabelle abgerufen wird, z. B. aus einer Sharded- oder Referenztabelle.

Aktion

Identifizieren Sie nicht optimierte `SELECT`-Abfragen in der Ansicht `limitless_stat_statements`. Optimieren Sie Abfragen, um nur die erforderlichen Daten abzurufen. Sie können auch den Parameter `rds_aurora.limitless_maximum_adaptive_fetch_size` anpassen.

Warteereignis „AuroraLimitless:AsyncGetInitialResponse“

Der Prozess wartet auf eine erste Antwort, wenn der Pipeline-Modus bei der Abfrageausführung verwendet wird.

Ursachen

Dies tritt normalerweise bei der Router-zu-Shard-Ausführung bei Abfragen mit Single-Shard-Datenplatzierung auf und ist ein erwarteter Teil der normalen Ausführung.

Aktion

Es sind keine weiteren Maßnahmen erforderlich.

Warteereignis „AuroraLimitless:AsyncGetNextResponse“

Der Prozess wartet auf eine zusätzliche Antwort, wenn der Pipeline-Modus bei der Abfrageausführung verwendet wird.

Ursachen

Dies tritt normalerweise bei der Router-zu-Shard-Ausführung bei Abfragen mit Single-Shard-Datenplatzierung auf und ist ein erwarteter Teil der normalen Ausführung.

Aktion

Es sind keine weiteren Maßnahmen erforderlich.

Warteereignis „AuroraLimitless:AbortedCommandCleanup“

Der Prozess wartet auf das Ergebnis einer Remote-Bereinigungsabfrage. Bereinigungsabfragen werden an Shard-Knoten ausgegeben, um sie nach dem Beenden einer verteilten Transaktion wieder in einen geeigneten Zustand zu versetzen.

Ursachen

Eine Transaktionsbereinigung wird durchgeführt, wenn eine Transaktion abgebrochen wird, weil ein Fehler gefunden wurde oder weil ein Benutzer einen expliziten ABORT-Befehl ausgegeben oder die laufende Abfrage abgebrochen hat.

Aktion

Untersuchen Sie die Ursache für den Abbruch der Transaktion.

Warteereignis „AuroraLimitless:DistributedCommitPrepare“

Der Prozess schreibt eine verteilte Transaktion fest und wartet darauf, dass alle Teilnehmer den Prepare-Befehl bestätigen.

Ursachen

Transaktionen, die mehrere Knoten ändern, müssen eine verteilte Festschreibung durchführen. Eine lange Wartezeit in `DistributedCommitPrepare` kann durch lange Wartezeiten beim Ereignis `IO:TwophaseFilePoolWrite` auf den teilnehmenden Knoten verursacht werden.

Aktion

Reduzieren Sie die Anzahl der Transaktionen, die Daten auf mehreren Knoten ändern. Untersuchen Sie `IO:TwophaseFilePoolWrite`-Ereignisse in anderen Knoten des Clusters.

Warteereignis „AuroraLimitless:DistributedCommit“

Der Prozess schreibt eine verteilte Transaktion fest und wartet darauf, dass der führende Teilnehmer den Prepare-Befehl bestätigt.

Ursachen

Transaktionen, die mehrere Knoten ändern, müssen eine verteilte Festschreibung durchführen. Eine lange Wartezeit in `DistributedCommit` kann durch lange Wartezeiten beim `IO:XactSync`-Ereignis auf den führenden Teilnehmer verursacht werden.

Aktion

Reduzieren Sie die Anzahl der Transaktionen, die Daten auf mehreren Knoten ändern. Untersuchen Sie `IO:XactSync`-Ereignisse in anderen Knoten des Clusters.

Warteereignis „AuroraLimitless:DistributedCommitPrepareThrottle“

Der Prozess versucht, eine verteilte Transaktion vorzubereiten, und wird aufgrund vorhandener vorbereiteter Transaktionen auf die Dauer gedrosselt.

Ursachen

Transaktionen, die mehrere Knoten ändern, müssen eine verteilte Festschreibung durchführen. Die Teilnehmer an diesen Transaktionen müssen im Rahmen des Festschreibungsprotokolls einen Vorbereitungsvorgang durchführen. Aurora begrenzt die Anzahl der gleichzeitigen prepare-Vorgänge, und wenn dieses Limit überschritten wird, wartet der Prozess auf das `DistributedCommitPrepareThrottle`-Ereignis.

Aktion

Reduzieren Sie die Anzahl der Transaktionen, die Daten auf mehreren Knoten ändern. Untersuchen Sie `IO:TwophaseFilePoolWrite`-Ereignisse, da eine längere Dauer dieser Ereignisse dazu führen kann, dass sich bereits vorbereitete Transaktionen ansammeln, was zu einer Drosselung neuer Vorbereitungsversuche führt.

Warteereignis „AuroraLimitless:PreparedTransactionResolution“

Beim Prozess ist ein Tupel aufgetreten, das durch eine verteilte Transaktion verändert wurde, die sich im Status „Vorbereitet“ befindet. Der Prozess muss ermitteln, ob die verteilte Transaktion in ihrem Snapshot sichtbar wird.

Ursachen

Transaktionen, die mehrere Knoten ändern, müssen eine verteilte Festschreibung durchführen, die eine Vorbereitungsphase beinhaltet. Eine hohe Anzahl verteilter Transaktionen oder eine erhöhte

Latenz bei verteilten Festschreibungen können dazu führen, dass bei anderen Prozessen das Warteereignis `PreparedTransactionResolution` auftritt.

Aktion

Reduzieren Sie die Anzahl der Transaktionen, die Daten auf mehreren Knoten ändern. Untersuchen Sie Ereignisse im Zusammenhang mit verteilten Festschreibungen, da eine längere Dauer dieser Ereignisse die Latenz im Festschreibungspfad verteilter Transaktionen erhöhen könnte. Möglicherweise möchten Sie auch die Netzwerk- und CPU-Auslastung untersuchen.

Warteereignis „AuroraLimitless:SendPreparedTransactionOutcome“

Der Prozess wird auf einem Knoten ausgeführt, der eine verteilte Transaktion koordiniert, und ein anderer Prozess hat den Status dieser Transaktion abgefragt, oder der Prozess hat eine verteilte Transaktion festgeschrieben und sendet das Ergebnis an die Teilnehmer.

Ursachen

Prozesse, bei denen das Warteereignis `PreparedTransactionResolution` auftritt, fragen den Transaktionskoordinator ab. Bei der Antwort des Transaktionskoordinators tritt SendPreparedTransactionOutcome auf.

Aktion

Reduzieren Sie die Anzahl der Transaktionen, die Daten auf mehreren Knoten ändern. Untersuchen Sie Ereignisse im Zusammenhang mit verteilten Festschreibungen sowie `IO:TwophaseFilePoolWrite`- und `IO:TwophaseFilePoolRead`-Ereignissen, da eine längere Dauer dieser Ereignisse die Latenz im Festschreibungspfad verteilter Transaktionen erhöhen könnte. Möglicherweise möchten Sie auch die Netzwerk- und CPU-Auslastung untersuchen.

Warteereignis „AuroraLimitless:CommitClockBarrier“

Der Prozess schreibt eine Transaktion fest und muss warten, um sicherzustellen, dass die zugewiesene Festschreibungszeit für alle Knoten im Cluster garantiert in der Vergangenheit liegt.

Ursachen

Eine CPU- oder Netzwerkauslastung kann zu einer erhöhten Taktabweichung führen, was zu einer Verlängerung der Wartezeit bei diesem Warteereignis führt.

Aktion

Untersuchen Sie die CPU- oder Netzwerkauslastung in Ihrem Cluster.

Warteereignis „AuroraLimitless:SnapshotClockBarrier“

Der Prozess hat eine Snapshot-Zeit von einem anderen Knoten mit einem Takt in der Zukunft empfangen und wartet darauf, dass sein eigener Takt diese Zeit erreicht.

Ursachen

Dies tritt normalerweise auf, nachdem der Prozess Ergebnisse einer Funktion erhalten hat, die auf einen Shard heruntergeschoben wurde, und eine Taktverschiebung zwischen den Knoten vorliegt. Eine CPU- oder Netzwerkauslastung kann zu einer erhöhten Taktabweichung führen, was zu einer Verlängerung der Wartezeit bei diesem Warteereignis führt.

Aktion

Untersuchen Sie die CPU- oder Netzwerkauslastung in Ihrem Cluster.

Warteereignis „AuroraLimitless:ReaderSnapshotClockBarrier“

Dieses Ereignis tritt auf Leseknoten auf. Der Prozess wartet darauf, dass der Leseknoten den Schreibstream wiedergibt, sodass alle Schreibvorgänge, die vor dem Zeitpunkt des Prozess-Snapshots stattfanden, übernommen wurden.

Ursachen

Eine Zunahme der Aurora-Replikatverzögerung kann in diesem Fall zu längeren Wartezeiten führen.

Aktion

Untersuchen Sie die Aurora-Replikatverzögerung.

Warteereignis „AuroraLimitless:GatherDistributedDeadlockGraph“

Der Prozess kommuniziert mit anderen Knoten, um im Rahmen der Erkennung verteilter Deadlocks Sperrdiagramme zu erfassen.

Ursachen

Wenn ein Prozess auf eine Sperre wartet, führt er eine Überprüfung auf verteilte Deadlocks durch, nachdem er länger gewartet hat als `rds_aurora.limitless_distributed_deadlock_timeout`.

Aktion

Untersuchen Sie die Ursachen für Sperrkonflikte in Ihrer Anwendung und ziehen Sie eine Optimierung von `rds_aurora.limitless_distributed_deadlock_timeout` in Betracht.

Warteereignis „AuroraLimitless:DistributedDeadlockDetection“

Der Prozess kommuniziert mit anderen Knoten, um verteilte Deadlocks zu erkennen.

Ursachen

Wenn ein Prozess auf eine Sperre wartet, führt er eine Überprüfung auf verteilte Deadlocks durch, nachdem er länger gewartet hat als `rds_aurora.limitless_distributed_deadlock_timeout`.

Aktion

Untersuchen Sie die Ursachen für Sperrkonflikte in Ihrer Anwendung und ziehen Sie eine Optimierung von `rds_aurora.limitless_distributed_deadlock_timeout` in Betracht.

Warteereignis „AuroraLimitless:DistributedDeadlockAbort“

Der Prozess kommuniziert mit einem anderen Knoten, um eine Sitzung abzubrechen, die als Opfer in einem verteilten Deadlock ausgewählt wurde.

Ursachen

Anwendungsmuster führen zu verteilten Deadlocks.

Aktion

Untersuchen Sie Anwendungsmuster, die zu verteilten Deadlocks führen.

Warteereignis „AuroraLimitless:GatherRemoteStats“

Der Prozess sammelt Statistiken von anderen Knoten im Cluster.

Ursachen

Überwachungs- oder Aktivitätsabfragen und -ansichten wie beispielsweise`limitless_stat_activity` rufen Statistiken von anderen Knoten ab.

Aktion

Es sind keine weiteren Maßnahmen erforderlich.

Warteereignis „AuroraLimitless:GlobalSequenceRefresh“

Der Prozess generiert einen neuen Sequenzwert und muss einen neuen Block aus der globalen Sequenz anfordern.

Ursachen

Eine hohe Rate der Generierung von Sequenzwerten kann in diesem Fall zu Verzögerungen führen, wenn `rds_aurora.limitless_sequence_chunk_size` nicht ausreicht.

Aktion

Dies ist ein normales Ereignis. Wenn Sie feststellen, dass dieses Ereignis zu lange dauert, sollten Sie eine Optimierung von `rds_aurora.limitless_sequence_chunk_size` in Betracht ziehen. Weitere Informationen finden Sie in der Dokumentation zu Sequenzen in Limitless Database.

Warteereignis „AuroraLimitless:GlobalVacuumTimeExchange“

Der Prozess tauscht Snapshot-Daten zur Unterstützung der Bereinigung aus.

Ursachen

Knoten in Limitless Database tauschen die ältesten aktiven Snapshot-Daten mit anderen Knoten aus, um den korrekten Sperrzeitpunkt für die Ausführung der Bereinigung zu berechnen.

Aktion

Es sind keine weiteren Maßnahmen erforderlich.

Warteereignis „AuroraLimitless:DistributedTransactionMonitorGather“

Der Prozess sammelt Transaktionsmetadaten von anderen Knoten, um die verteilte Transaktionsbereinigung zu unterstützen.

Ursachen

Knoten in Limitless Database tauschen Transaktionsmetadaten mit anderen Knoten aus, um festzustellen, wann der Status verteilter Transaktionen gelöscht werden kann.

Aktion

Es sind keine weiteren Maßnahmen erforderlich.

Warteereignis „AuroraLimitlessActivity:AdminTaskSchedulerMain“

Wartet in der Hauptschleife des Aufgabenplanerprozesses.

Warteereignis „AuroraLimitlessActivity:AdminTaskExecutorMain“

Wartet in der Hauptschleife des Aufgabenausführungsprozesses.

Warteereignis „AuroraLimitlessActivity:AdminTaskMonitorMain“

Wartet in der Hauptschleife des Aufgabenüberwachungsprozesses.

Warteereignis „AuroraLimitlessActivity:DatabaseCleanupMonitorMain“

Wartet in der Hauptschleife des Aufgabenüberwachungsprozesses der Datenbank.

Warteereignis „AuroraLimitlessActivity:TopologyCleanupMonitorMain“

Wartet in der Hauptschleife des Bereinigungsüberwachungsprozesses der Topologie.

Warteereignis „AuroraLimitlessActivity:ToplogyChangeMonitorMain“

Wartet in der Hauptschleife des Änderungsüberwachungsprozesses der Topologie.

Warteereignis „AuroraLimitlessActivity:DistributedTransactionMonitorMain“

Wartet in der Hauptschleife des verteilten Transaktionsüberwachungsprozesses.

Warteereignis „AuroraLimitlessActivity:GlobalVacuumMonitorMain“

Wartet in der Hauptschleife des globalen Bereinigungsüberwachungsprozesses.

Sichern und Wiederherstellen von Aurora PostgreSQL Limitless Database

Sie können einen DB-Cluster, der Aurora PostgreSQL Limitless Database verwendet, sichern und wiederherstellen.

Inhalt

- Sicheren eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet
  - Erstellen eines DB-Cluster-Snapshots
- Wiederherstellen eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet
  - Wiederherstellen eines DB-Clusters aus einem DB-Snapshot
  - Wiederherstellen eines DB-Clusters mit zeitpunktbezogener Wiederherstellung
- PostgreSQL-Dienstprogramme zur Sicherung und Wiederherstellung werden nicht unterstützt

Sicheren eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet

Die Sicherung eines DB-Clusters mit Aurora PostgreSQL Limitless Database weist im Vergleich zur Sicherung eines standardmäßigen Aurora-DB-Clusters Ähnlichkeiten und Unterschiede in der Funktionalität auf.

- Wenn Sie einen manuellen DB-Cluster-Snapshot eines Aurora-DB-Clusters erstellen, der Limitless Database verwendet, enthält der Snapshot Daten aus der DB-Shard-Gruppe.
- Kontinuierliche Backups umfassen Daten aus der DB-Shard-Gruppe.
- Automatisierte tägliche Snapshots umfassen Daten aus der DB-Shard-Gruppe.
- DB-Cluster Snapshots lassen sich kopieren. Weitere Informationen finden Sie unter Kopieren eines DB-Cluster-Snapshots.
- DB-Cluster Snapshots lassen sich teilen. Weitere Informationen finden Sie unter Freigeben eines DB-Cluster-Snapshots.
- Sie können das Dienstprogramm `pg_dump` oder `pg_dumpall` nicht verwenden, um Datenbanken in der DB-Shard-Gruppe zu sichern.
- Beim Löschen von DB-Clustern lassen sich für Aurora PostgreSQL Limitless Database finale Snapshots erstellen.

- Beim Löschen von DB-Clustern lassen sich für Aurora PostgreSQL Limitless Database keine automatisierten Sicherungen beibehalten.

Erstellen eines DB-Cluster-Snapshots

DB-Cluster-Snapshots von Aurora PostgreSQL Limitless Database erstellt man auf die gleiche Weise wie von einem standardmäßigen Aurora-DB-Cluster, wie im folgenden AWS CLI-Beispiel gezeigt:

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier my-db-cluster \
    --db-cluster-snapshot-identifier my-db-cluster-snapshot
```

Informationen zum Sichern von DB-Clustern finden Sie unter Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters.

Wiederherstellen eines DB-Clusters, der Aurora PostgreSQL Limitless Database verwendet

Die Wiederherstellung eines DB-Clusters mit Aurora PostgreSQL Limitless Database weist im Vergleich zur Wiederherstellung eines standardmäßigen Aurora-DB-Clusters Ähnlichkeiten und Unterschiede in der Funktionalität auf.

- Sie können einen DB-Cluster von Limitless Database nur aus einem Quell-DB-Cluster wiederherstellen, der eine mit Limitless Database kompatible DB-Engine-Version ausführt, z. B. `16.4-limitless`.
- Wenn Sie einen DB-Cluster aus dem manuellen Snapshot eines DB-Clusters wiederherstellen, der Limitless Database verwendet, wird der gesamte DB-Cluster-Speicher wiederhergestellt. Dies umfasst den Speicher der DB-Shard-Gruppe.

  Sie müssen eine DB-Shard-Gruppe erstellen, um auf den Speicher für Ihre Limitless Database zugreifen zu können.
- Mit der zeitpunktbezogenen Wiederherstellung (PITR) können Sie einen DB-Cluster auf jeden beliebigen Zeitpunkt innerhalb des Aufbewahrungszeitraums wiederherstellen. Der wiederhergestellte DB-Cluster umfasst den Speicher der DB-Shard-Gruppe.

  Sie müssen eine DB-Shard-Gruppe erstellen, um auf den Speicher für Ihre Limitless Database zugreifen zu können.
- PITR wird für gelöschte DB-Cluster von Aurora PostgreSQL Limitless Database nicht unterstützt.
- Wenn Sie einen DB-Cluster aus einem automatisierten täglichen Snapshot wiederherstellen, wird auch der Speicher für die DB-Shard-Gruppe wiederhergestellt.
- Wenn Sie einen DB-Cluster von Aurora PostgreSQL Limitless Database wiederherstellen, müssen Sie die erweiterte Überwachung und Performance Insights aktivieren. Geben Sie unbedingt die Performance-Insights KMS-Schlüssel-ID an.

Nachdem Sie einen DB-Cluster von Aurora PostgreSQL Limitless Database wiederhergestellt haben, sollten Sie dessen Funktionalität überprüfen, indem Sie Ihre Abfragen darauf ausführen.

Wiederherstellen eines DB-Clusters aus einem DB-Snapshot

Die folgenden AWS CLI-Beispiele zeigen, wie Sie einen DB-Cluster von Aurora PostgreSQL Limitless Database aus einem DB-Cluster-Snapshot wiederherstellen.

Sie müssen DB-Engine-Version `16.4-limitless` verwenden.

So stellen Sie einen DB-Cluster von Limitless Database aus einem DB-Cluster-Snapshot wieder her

1. Stellen Sie den DB-Cluster wieder her:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-db-cluster \
    --snapshot-identifier my-db-cluster-snapshot \
    --engine aurora-postgresql \
    --engine-version 16.4-limitless \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. Erstellen Sie die DB-Shard-Gruppe:

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   Weitere Informationen finden Sie unter Hinzufügen einer DB-Shard-Gruppe zu einem vorhandenen DB-Cluster von Aurora PostgreSQL Limitless Database.

Weitere Informationen zum Wiederherstellen eines Aurora-DB-Clusters aus DB-Cluster-Snapshots finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.

Wiederherstellen eines DB-Clusters mit zeitpunktbezogener Wiederherstellung

Die folgenden AWS CLI-Beispiele zeigen, wie Sie einen DB-Cluster von Aurora PostgreSQL Limitless Database mit einer zeitpunktbezogenen Wiederherstellung (PITR) wiederherstellen.

So stellen Sie einen DB-Cluster von Limitless Database mit PITR wieder her

1. Stellen Sie den DB-Cluster wieder her:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-db-cluster \
    --db-cluster-identifier my-new-db-cluster \
    --use-latest-restorable-time \
    --enable-performance-insights \
    --performance-insights-retention-period 31 \
    --performance-insights-kms-key-id arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --monitoring-interval 5 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/EMrole
```

2. Erstellen Sie die DB-Shard-Gruppe:

```
aws rds create-db-shard-group \
    --db-cluster-identifier my-new-db-cluster \
    --db-shard-group-identifier my-new-DB-shard-group \
    --max-acu 1000
```

   Weitere Informationen finden Sie unter Hinzufügen einer DB-Shard-Gruppe zu einem vorhandenen DB-Cluster von Aurora PostgreSQL Limitless Database.

Weitere Informationen zu PITR finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit.

PostgreSQL-Dienstprogramme zur Sicherung und Wiederherstellung werden nicht unterstützt

Die folgenden PostgreSQL-Dienstprogramme werden weder für den primären DB-Cluster noch für die DB-Shard-Gruppe unterstützt:

- `pg_dump`
- `pg_dumpall`
- `pg_restore`

Sie können sie zwar mit Open-Source-Binärdateien oder alternativen Methoden verwenden, dies kann jedoch zu uneinheitlichen Ergebnissen führen.

Aktualisieren von Amazon Aurora PostgreSQL Limitless Database

Für das Upgrade der Aurora PostgreSQL Limitless Database gilt Folgendes:

- Es werden Upgrades von Nebenversionen unterstützt.
- Das Patching von Aurora PostgreSQL Limitless Database wird unterstützt. Patches werden als ausstehende Wartungsarbeiten angezeigt und müssen während Ihres Wartungsfensters installiert werden.

Aktualisieren von DB-Clustern, die Amazon Aurora PostgreSQL Limitless Database verwenden

Sie aktualisieren einen DB-Cluster, indem Sie ihn ändern und eine neue DB-Engine-Version auswählen. Sie können die AWS-Managementkonsole oder die AWS CLI verwenden.

Konsole

So aktualisieren Sie Ihren DB-Cluster von Limitless Database

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Navigieren Sie zur Seite Datenbanken.
3. Wählen Sie den DB-Cluster von Limitless Database aus, dem Sie eine DB-Shard-Gruppe hinzufügen möchten.
4. Wählen Sie Ändern aus.

   Die Seite DB-Cluster ändern wird angezeigt.
5. Wählen Sie als DB-Engine-Version die neue DB-Engine-Version aus, zum Beispiel Aurora PostgreSQL mit Limitless Database (mit PostgreSQL 16.6 kompatibel).
6. Klicken Sie auf Weiter.
7. Legen Sie auf der Zusammenfassungsseite fest, ob die Änderungen sofort oder während des nächsten Wartungszeitraums übernommen werden sollen, und wählen Sie dann Cluster ändern aus.

CLI

Verwenden Sie den AWS CLI-Befehl modify-db-cluster, wie im folgenden Beispiel gezeigt.

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-sv2-cluster \
    --engine-version 16.6-limitless \
    --apply-immediately
```

Referenz zu Aurora PostgreSQL Limitless Database

Wir bieten die folgenden Referenzthemen für Aurora PostgreSQL Limitless Database.

Themen

- Unterstützte und nicht unterstützte Data Definition Language (DDL)-SQL-Befehle
- DDL-Einschränkungen und andere Informationen für Aurora PostgreSQL Limitless Database
- Unterstützte und nicht unterstützte Data Manipulation Language (DML)- und SQL-Befehle zur Abfrageverarbeitung
- DML-Einschränkungen und andere Informationen für Aurora PostgreSQL Limitless Database
- Variablen in Aurora PostgreSQL Limitless Database
- DB-Cluster-Parameter in der Aurora PostgreSQL Limitless-Datenbank

Unterstützte und nicht unterstützte Data Definition Language (DDL)-SQL-Befehle

In der folgenden Tabelle sind die DDL-Befehle aufgeführt, die von Aurora PostgreSQL Limitless Database unterstützt und nicht unterstützt werden, mit Verweisen auf Einschränkungen oder weitere Informationen.

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
ALTER AGGREGATE	Nein	Nicht zutreffend
ALTER COLLATION	Ja	Keine
ALTER CONVERSION	Ja	Keine
ALTER DATABASE	Nein	Nicht zutreffend
ALTER DEFAULT PRIVILEGES	Nein	Nicht zutreffend
ALTER DOMAIN	Nein	Nicht zutreffend
ALTER EVENT TRIGGER	Nein	Nicht zutreffend

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
ALTER EXTENSION	Ja	Erweiterungen
ALTER FOREIGN DATA WRAPPER	Nein	Nicht zutreffend
ALTER FOREIGN TABLE	Nein	Nicht zutreffend
ALTER FUNCTION	Ja	Funktionen
ALTER GROUP	Ja	Keine
ALTER INDEX	Ja	Keine
ALTER LANGUAGE	Nein	Nicht zutreffend
ALTER LARGE OBJECT	Nein	Nicht zutreffend
ALTER MATERIALIZED VIEW	Nein	Nicht zutreffend
ALTER OPERATOR	Ja	Keine
ALTER OPERATOR CLASS	Ja	Keine
ALTER OPERATOR FAMILY	Ja	Keine
ALTER POLICY	Nein	Nicht zutreffend
ALTER PROCEDURE	Ja	Keine
ALTER PUBLICATION	Nein	Nicht zutreffend
ALTER ROLE	Ja	Keine
ALTER ROUTINE	Nein	Nicht zutreffend
ALTER RULE	Nein	Nicht zutreffend
ALTER SCHEMA	Ja	Keine

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
ALTER SEQUENCE	Ja	Sequenzen
ALTER SERVER	Nein	Nicht zutreffend
ALTER STATISTICS	Nein	Nicht zutreffend
ALTER SUBSCRIPTION	Nein	Nicht zutreffend
ALTER SYSTEM	Nein	Nicht zutreffend
ALTER TABLE	Ja	ALTER TABLE
ALTER TABLESPACE	Nein	Nicht zutreffend
ALTER TEXT SEARCH CONFIGURATION	Nein	Nicht zutreffend
ALTER TEXT SEARCH DICTIONARY	Nein	Nicht zutreffend
ALTER TEXT SEARCH PARSER	Nein	Nicht zutreffend
ALTER TEXT SEARCH TEMPLATE	Nein	Nicht zutreffend
ALTER TRIGGER	Nein	Nicht zutreffend
ALTER TYPE	Ja	Keine
ALTER USER	Ja	Keine
ALTER USER MAPPING	Nein	Nicht zutreffend
ALTER VIEW	Ja	Keine
COMMENT	Nein	Nicht zutreffend

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
CREATE ACCESS METHOD	Nein	Nicht zutreffend
CREATE AGGREGATE	Nein	Nicht zutreffend
CREATE CAST	Ja	Keine
CREATE COLLATION	Ja	Keine
CREATE CONVERSION	Ja	Keine
CREATE DATABASE	Ja	CREATE DATABASE
CREATE DOMAIN	Nein	Nicht zutreffend
CREATE EVENT TRIGGER	Nein	Nicht zutreffend
CREATE EXTENSION	Ja	Erweiterungen
CREATE FOREIGN DATA WRAPPER	Nein	Nicht zutreffend
CREATE FOREIGN TABLE	Nein	Nicht zutreffend
CREATE FUNCTION	Ja	Funktionen
CREATE GROUP	Ja	Keine
CREATE INDEX	Ja	CREATE INDEX
CREATE LANGUAGE	Nein	Nicht zutreffend
CREATE MATERIALIZED VIEW	Nein	Nicht zutreffend
CREATE OPERATOR	Ja	Keine
CREATE OPERATOR CLASS	Ja	Keine

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
CREATE OPERATOR FAMILY	Ja	Keine
CREATE POLICY	Ja	Keine
CREATE PROCEDURE	Ja	Keine
CREATE PUBLICATION	Nein	Nicht zutreffend
CREATE ROLE	Ja	Keine
CREATE RULE	Nein	Nicht zutreffend
CREATE SCHEMA	Ja	CREATE SCHEMA
CREATE SEQUENCE	Ja	Sequenzen
CREATE SERVER	Nein	Nicht zutreffend
CREATE STATISTICS	Nein	Nicht zutreffend
CREATE SUBSCRIPTION	Nein	Nicht zutreffend
CREATE TABLE	Ja	CREATE TABLE
CREATE TABLE AS	Ja	CREATE TABLE AS
CREATE TABLESPACE	Nein	Nicht zutreffend
CREATE TEMPORARY TABLE	Nein	Nicht zutreffend
CREATE TEMPORARY TABLE AS	Nein	Nicht zutreffend
CREATE TEXT SEARCH CONFIGURATION	Nein	Nicht zutreffend

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
CREATE TEXT SEARCH DICTIONARY	Nein	Nicht zutreffend
CREATE TEXT SEARCH PARSER	Nein	Nicht zutreffend
CREATE TEXT SEARCH TEMPLATE	Nein	Nicht zutreffend
CREATE TRANSFORM	Nein	Nicht zutreffend
CREATE TRIGGER	Nein	Nicht zutreffend
CREATE TYPE	Ja	Keine
CREATE USER	Ja	Keine
CREATE USER MAPPING	Nein	Nicht zutreffend
CREATE VIEW	Ja	Keine
DROP ACCESS METHOD	Nein	Nicht zutreffend
DROP AGGREGATE	Ja	Keine
DROP CAST	Ja	Keine
DROP COLLATION	Ja	Keine
DROP CONVERSION	Ja	Keine
DROP DATABASE	Ja	DROP DATABASE
DROP DOMAIN	Nein	Nicht zutreffend
DROP EVENT TRIGGER	Nein	Nicht zutreffend
DROP EXTENSION	Ja	Erweiterungen

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
DROP FOREIGN DATA WRAPPER	Nein	Nicht zutreffend
DROP FOREIGN TABLE	Nein	Nicht zutreffend
DROP FUNCTION	Ja	Funktionen
DROP GROUP	Ja	Keine
DROP INDEX	Ja	Keine
DROP LANGUAGE	Nein	Nicht zutreffend
DROP MATERIALIZED VIEW	Nein	Nicht zutreffend
DROP OPERATOR	Ja	Keine
DROP OPERATOR CLASS	Ja	Keine
DROP OPERATOR FAMILY	Ja	Keine
DROP OWNED	Nein	Nicht zutreffend
DROP POLICY	Nein	Nicht zutreffend
DROP PROCEDURE	Ja	Keine
DROP PUBLICATION	Nein	Nicht zutreffend
DROP ROLE	Ja	Keine
DROP ROUTINE	Nein	Nicht zutreffend
DROP RULE	Nein	Nicht zutreffend
DROP SCHEMA	Ja	Keine
DROP SEQUENCE	Ja	Keine

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
DROP SERVER	Nein	Nicht zutreffend
DROP STATISTICS	Nein	Nicht zutreffend
DROP SUBSCRIPTION	Nein	Keine
DROP TABLE	Ja	Keine
DROP TABLESPACE	Nein	Nicht zutreffend
DROP TEXT SEARCH CONFIGURATION	Nein	Nicht zutreffend
DROP TEXT SEARCH DICTIONARY	Nein	Nicht zutreffend
DROP TEXT SEARCH PARSER	Nein	Nicht zutreffend
DROP TEXT SEARCH TEMPLATE	Nein	Nicht zutreffend
DROP TRANSFORM	Nein	Nicht zutreffend
DROP TRIGGER	Nein	Nicht zutreffend
DROP TYPE	Ja	Keine
DROP USER	Ja	Keine
DROP USER MAPPING	Nein	Nicht zutreffend
DROP VIEW	Ja	Keine
GRANT	Ja	Keine
REASSIGN OWNED	Nein	Nicht zutreffend

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
REVOKE	Ja	Keine
SECURITY LABEL	Nein	Nicht zutreffend
SELECT INTO	Ja	SELECT INTO
SET	Ja	Keine
SET CONSTRAINTS	Nein	Nicht zutreffend
SET ROLE	Ja	Keine
SET SESSION AUTHORIZATION	Ja	Keine
SET TRANSACTION	Ja	Keine
TRUNCATE	Ja	Keine

DDL-Einschränkungen und andere Informationen für Aurora PostgreSQL Limitless Database

In den folgenden Themen werden Einschränkungen beschrieben oder weitere Informationen zu DDL-SQL-Befehlen in Aurora PostgreSQL Limitless Database bereitgestellt.

Themen

- ALTER TABLE
- CREATE DATABASE
- CREATE INDEX
- CREATE SCHEMA
- CREATE TABLE
- CREATE TABLE AS
- DROP DATABASE
- SELECT INTO
- Beschränkungen
- Standardwerte
- Erweiterungen
- Fremdschlüssel
- Funktionen
- Sequenzen

ALTER TABLE

Der Befehl `ALTER TABLE` wird in Aurora PostgreSQL Limitless Database in der Regel unterstützt. Weitere Informationen finden Sie im Abschnitt ALTER TABLE der PostgreSQL-Dokumentation.

Einschränkungen

`ALTER TABLE` weist die folgenden Einschränkungen für unterstützte Optionen auf.

Entfernen einer Spalte

- In Sharded-Tabellen können Sie keine Spalten entfernen, die Teil des Shard-Schlüssels sind.
- In Referenztabellen können Sie keine Primärschlüsselspalten entfernen.

Ändern des Datentyps einer Spalte

- Der Ausdruck `USING` wird nicht unterstützt.
- In Sharded-Tabellen können Sie den Datentyp von Spalten nicht ändern, die Teil des Shard-Schlüssels sind.

Hinzufügen oder Entfernen einer Beschränkung

Einzelheiten darüber, was nicht unterstützt wird, finden Sie unter Beschränkungen.

Ändern des Standardwerts einer Spalte

Standardwerte werden unterstützt. Weitere Informationen finden Sie unter Standardwerte.

Nicht unterstützte Optionen

Einige Optionen werden nicht unterstützt, da sie von nicht unterstützten Features wie Auslösern abhängen.

Die folgenden Optionen auf Tabellenebene für `ALTER TABLE` werden nicht unterstützt:

- `ALL IN TABLESPACE`
- `ATTACH PARTITION`
- `DETACH PARTITION`
- `ONLY`-Flag
- `RENAME CONSTRAINT`

Die folgenden Optionen auf Spaltenebene für `ALTER TABLE` werden nicht unterstützt:

- ADD GENERATED
- DROP EXPRESSION [IF EXISTS]
- DROP IDENTITY [IF EXISTS]
- RESET
- RESTART
- SET
- SET COMPRESSION
- SET STATISTICS

CREATE DATABASE

In Aurora PostgreSQL Limitless Database werden nur uneingeschränkte Datenbanken unterstützt.

Während `CREATE DATABASE` ausgeführt wird, schlagen Datenbanken, die in einem oder mehreren Knoten erfolgreich erstellt wurden, möglicherweise in anderen Knoten fehl, da eine Datenbankerstellung ein nicht-transaktionaler Vorgang ist. In diesem Fall werden Datenbankobjekte, die erfolgreich erstellt wurden, innerhalb eines vordefinierten Zeitraums automatisch von allen Knoten entfernt, damit in der DB-Shard-Gruppe die Konsistenz erhalten bleibt. In diesem Zeitraum kann die Neuerstellung einer Datenbank mit demselben Namen zu einem Fehler führen, der darauf hinweist, dass die Datenbank bereits existiert.

Die folgenden Optionen werden unterstützt:

- Kollation:

```
CREATE DATABASE name WITH
    [LOCALE = locale]
    [LC_COLLATE = lc_collate]
    [LC_CTYPE = lc_ctype]
    [ICU_LOCALE = icu_locale]
    [ICU_RULES = icu_rules]
    [LOCALE_PROVIDER = locale_provider]
    [COLLATION_VERSION = collation_version];
```

- `CREATE DATABASE WITH OWNER`:

```
CREATE DATABASE name WITH OWNER = user_name;
```

Die folgenden Optionen werden nicht unterstützt:

- `CREATE DATABASE WITH TABLESPACE`:

```
CREATE DATABASE name WITH TABLESPACE = tablespace_name;
```

- `CREATE DATABASE WITH TEMPLATE`:

```
CREATE DATABASE name WITH TEMPLATE = template;
```

CREATE INDEX

`CREATE INDEX CONCURRENTLY` wird für Sharded-Tabellen unterstützt:

```
CREATE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

`CREATE UNIQUE INDEX` wird für alle Tabellentypen unterstützt:

```
CREATE UNIQUE INDEX index_name ON table_name(column_name);
```

`CREATE UNIQUE INDEX CONCURRENTLY` wird nicht unterstützt:

```
CREATE UNIQUE INDEX CONCURRENTLY index_name ON table_name(column_name);
```

Weitere Informationen finden Sie unter UNIQUE. Allgemeine Informationen zum Erstellen von Indizes finden Sie im Abschnitt CREATE INDEX der PostgreSQL-Dokumentation.

Anzeigen von Indizes

Nicht alle Indizes sind auf Routern sichtbar, wenn Sie `\d table_name` oder ähnliche Befehle verwenden. Verwenden Sie stattdessen die Ansicht `pg_catalog.pg_indexes`, um Indizes abzurufen, wie im folgenden Beispiel gezeigt.

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"id"}';
CREATE TABLE items (id int PRIMARY KEY, val int);
CREATE INDEX items_my_index on items (id, val);

postgres_limitless=> SELECT * FROM pg_catalog.pg_indexes WHERE tablename='items';
```

```
 schemaname | tablename |   indexname    | tablespace |
   indexdef
------------+-----------+----------------+------------
+-------------------------------------------------------------------------
 public     | items     | items_my_index |            | CREATE INDEX items_my_index
 ON ONLY public.items USING btree (id, val)
 public     | items     | items_pkey     |            | CREATE UNIQUE INDEX
 items_pkey ON ONLY public.items USING btree (id)
(2 rows)
```

CREATE SCHEMA

`CREATE SCHEMA` mit einem Schemaelement wird nicht unterstützt:

```
CREATE SCHEMA my_schema CREATE TABLE (column_name INT);
```

Es tritt ein Fehler auf, der in etwa folgendermaßen aussieht:

```
ERROR: CREATE SCHEMA with schema elements is not supported
```

CREATE TABLE

Beziehungen in `CREATE TABLE`-Anweisungen werden nicht unterstützt, zum Beispiel:

```
CREATE TABLE orders (orderid int, customerId int, orderDate date) WITH
 (autovacuum_enabled = false);
```

`IDENTITY`-Spalten werden nicht unterstützt, zum Beispiel:

```
CREATE TABLE orders (orderid INT GENERATED ALWAYS AS IDENTITY);
```

Aurora PostgreSQL Limitless Database unterstützt bis zu 54 Zeichen für Namen von Sharded-Tabellen.

CREATE TABLE AS

Wenn Sie eine Tabelle mit `CREATE TABLE AS` erstellen möchten, müssen Sie die Variable `rds_aurora.limitless_create_table_mode` verwenden. Für Sharded-Tabellen müssen

Sie außerdem die Variable `rds_aurora.limitless_create_table_shard_key` verwenden. Weitere Informationen finden Sie unter Erstellen von Limitless-Tabellen mithilfe von Variablen.

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

CREATE TABLE ctas_table AS SELECT 1 a;

-- "source" is the source table whose columns and data types are used to create the new
 "ctas_table2" table.
CREATE TABLE ctas_table2 AS SELECT a,b FROM source;
```

Sie können `CREATE TABLE AS` nicht zum Erstellen von Referenztabellen verwenden, da sie Primärschlüsselbeschränkungen erfordern. `CREATE TABLE AS` propagiert Primärschlüssel nicht an neue Tabellen.

Allgemeine Informationen finden Sie im Abschnitt CREATE TABLE AS der PostgreSQL-Dokumentation.

DROP DATABASE

Sie können Datenbanken, die Sie erstellt haben, löschen.

Der Befehl `DROP DATABASE` wird asynchron im Hintergrund ausgeführt. Während der Ausführung erhalten Sie eine Fehlermeldung, wenn Sie versuchen, eine neue Datenbank mit demselben Namen zu erstellen.

SELECT INTO

`SELECT INTO` funktioniert ähnlich wie CREATE TABLE AS. Sie müssen die Variable `rds_aurora.limitless_create_table_mode` verwenden. Für Sharded-Tabellen müssen Sie außerdem die Variable `rds_aurora.limitless_create_table_shard_key` verwenden. Weitere Informationen finden Sie unter Erstellen von Limitless-Tabellen mithilfe von Variablen.

```
-- Set the variables.
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"a"}';

-- "source" is the source table whose columns and data types are used to create the new
 "destination" table.
```

```
SELECT * INTO destination FROM source;
```

Derzeit wird der Vorgang `SELECT INTO` über den Router ausgeführt, nicht direkt über die Shards. Dies kann die Leistung beeinträchtigen.

Weitere Informationen dazu finden Sie im Abschnitt SELECT INTO der PostgreSQL-Dokumentation.

Beschränkungen

Die folgenden Einschränkungen gelten für Beschränkungen in Aurora PostgreSQL Limitless Database.

CHECK

Einfache Beschränkungen, die Vergleichsoperatoren mit Literalen beinhalten, werden unterstützt. Komplexere Ausdrücke und Beschränkungen, die Funktionsvalidierungen erfordern, werden nicht unterstützt, wie in den folgenden Beispielen gezeigt.

```
CREATE TABLE my_table (
    id  INT CHECK (id > 0)                                        -- supported
  , val INT CHECK (val > 0 AND val < 1000)                        -- supported
  , tag TEXT CHECK (length(tag) > 0)                              -- not supported:
 throws "Expression inside CHECK constraint is not supported"
  , op_date TIMESTAMP WITH TIME ZONE CHECK (op_date <= now())  -- not supported:
 throws "Expression inside CHECK constraint is not supported"
);
```

Sie können Beschränkungen explizite Namen geben, wie im folgenden Beispiel gezeigt.

```
CREATE TABLE my_table (
    id  INT CONSTRAINT positive_id  CHECK (id > 0)
  , val INT CONSTRAINT val_in_range CHECK (val > 0 AND val < 1000)
);
```

Sie können die Beschränkungssyntax auf Tabellenebene mit der Beschränkung `CHECK` verwenden, wie im folgenden Beispiel gezeigt.

```
CREATE TABLE my_table (
    id INT CONSTRAINT positive_id  CHECK (id > 0)
  , min_val INT CONSTRAINT min_val_in_range CHECK (min_val > 0 AND min_val < 1000)
  , max_val INT
  , CONSTRAINT max_val_in_range CHECK (max_val > 0 AND max_val < 1000 AND max_val >
 min_val)
);
```

EXCLUDE

Ausschlussbeschränkungen werden in Aurora PostgreSQL Limitless Database nicht unterstützt.

FOREIGN KEY

Weitere Informationen finden Sie unter Fremdschlüssel.

NOT NULL

`NOT NULL`-Beschränkungen werden uneingeschränkt unterstützt.

PRIMARY KEY

Der Primärschlüssel impliziert eindeutige Beschränkungen, weshalb die Einschränkungen für eindeutige Beschränkungen auch für den Primärschlüssel gelten. Das bedeutet Folgendes:

- Wenn eine Tabelle in eine Sharded-Tabelle konvertiert wird, muss der Shard-Schlüssel eine Teilmenge des Primärschlüssels sein. Das heißt, der Primärschlüssel enthält alle Spalten des Shard-Schlüssels.
- Wenn eine Tabelle in eine Referenztabelle konvertiert wird, muss sie einen Primärschlüssel haben.

Die folgenden Beispiele veranschaulichen die Verwendung von Primärschlüsseln.

```
-- Create a standard table.
CREATE TABLE public.my_table (
    item_id INT
  , location_code INT
  , val INT
  , comment text
);

-- Change the table to a sharded table using the 'item_id' and 'location_code'
 columns as shard keys.
CALL rds_aurora.limitless_alter_table_type_sharded('public.my_table',
 ARRAY['item_id', 'location_code']);
```

Hier wird versucht, einen Primärschlüssel hinzuzufügen, der keinen Shard-Schlüssel enthält:

```
-- Add column 'item_id' as the primary key.
-- Invalid because the primary key doesnt include all columns from the shard key:
-- 'location_code' is part of the shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR

-- add column "val" as primary key
-- Invalid because primary key does not include all columns from shard key:
```

```
--  item_id and location_code iare part of shard key but not part of the primary key
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id); -- ERROR
```

Hier wird versucht, einen Primärschlüssel hinzuzufügen, der einen Shard-Schlüssel enthält:

```
-- Add the 'item_id' and 'location_code' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code); -- OK

-- Add the 'item_id', 'location_code', and 'val' columns as the primary key.
-- Valid because the primary key contains the shard key.
ALTER TABLE public.my_table ADD PRIMARY KEY (item_id, location_code, val); -- OK
```

In diesem Beispiel wird eine Standardtabelle in eine Referenztabelle konvertiert:

```
-- Create a standard table.
CREATE TABLE zipcodes (zipcode INT PRIMARY KEY, details VARCHAR);

-- Convert the table to a reference table.
CALL rds_aurora.limitless_alter_table_type_reference('public.zipcode');
```

Weitere Informationen zum Erstellen von Sharded-Tabellen und Referenztabellen finden Sie unter Erstellen von Tabellen in Aurora PostgreSQL Limitless Database.

UNIQUE

In Sharded-Tabellen muss der eindeutige Schlüssel den Shard-Schlüssel enthalten, das heißt, der Shard-Schlüssel muss eine Teilmenge des eindeutigen Schlüssels sein. Dies wird überprüft, wenn der Tabellentyp in „Sharded“ geändert wird. In Referenztabellen gibt es keine Einschränkung.

```
CREATE TABLE customer (
    customer_id INT NOT NULL
  , zipcode INT
  , email TEXT UNIQUE
);
```

UNIQUE-Beschränkungen auf Tabellenebene werden unterstützt, wie im folgenden Beispiel gezeigt.

```
CREATE TABLE customer (
    customer_id INT NOT NULL
```

```
  , zipcode INT
  , email TEXT
  , CONSTRAINT zipcode_and_email UNIQUE (zipcode, email)
);
```

Das folgende Beispiel zeigt die gemeinsame Verwendung eines Primärschlüssels und eines eindeutigen Schlüssels. Beide Schlüssel müssen den Shard-Schlüssel enthalten.

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"p_id"}';
CREATE TABLE t1 (
p_id BIGINT NOT NULL,
c_id BIGINT NOT NULL,
PRIMARY KEY (p_id),
UNIQUE (p_id, c_id)
);
```

Weitere Informationen finden Sie im Abschnitt Beschränkungen der PostgreSQL-Dokumentation.

Standardwerte

Aurora PostgreSQL Limitless Database unterstützt Ausdrücke in Standardwerten.

Das folgende Beispiel veranschaulicht die Verwendung von Standardwerten.

```
CREATE TABLE t (
    a INT DEFAULT 5,
    b TEXT DEFAULT 'NAN',
    c NUMERIC
);

CALL rds_aurora.limitless_alter_table_type_sharded('t', ARRAY['a']);
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c
---+-----+---
 5 | NAN |
(1 row)
```

Ausdrücke werden unterstützt, wie im folgenden Beispiel gezeigt.

```
CREATE TABLE t1 (a NUMERIC DEFAULT random());
```

Im folgenden Beispiel wird eine neue Spalte hinzugefügt, die NOT NULL ist und einen Standardwert hat.

```
ALTER TABLE t ADD COLUMN d BOOLEAN NOT NULL DEFAULT FALSE;
SELECT * FROM t;

 a |  b  | c | d
---+-----+---+---
 5 | NAN |   | f
(1 row)
```

Im folgenden Beispiel wird eine vorhandene Spalte geändert und ein Standardwert hinzugefügt.

```
ALTER TABLE t ALTER COLUMN c SET DEFAULT 0.0;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
(2 rows)
```

Im folgenden Beispiel wird ein Standardwert entfernt.

```
ALTER TABLE t ALTER COLUMN a DROP DEFAULT;
INSERT INTO t DEFAULT VALUES;
SELECT * FROM t;

 a |  b  | c   |  d
---+-----+-----+-----
 5 | NAN |     | f
 5 | NAN | 0.0 | f
   | NAN | 0.0 | f
(3 rows)
```

Weitere Informationen finden Sie im Abschnitt Standardwerte der PostgreSQL-Dokumentation.

Erweiterungen

Die folgenden PostgreSQL-Erweiterungen werden in Aurora PostgreSQL Limitless Database unterstützt:

- `aurora_limitless_fdw`: Diese Erweiterung ist vorinstalliert. Sie können sie nicht löschen.
- `aws_s3`: Diese Erweiterung funktioniert in Aurora PostgreSQL Limitless Database ähnlich wie in Aurora PostgreSQL.

  Sie können Daten aus einem Amazon-S3-Bucket in den DB-Cluster einer Aurora PostgreSQL Limitless Database importieren oder Daten aus dem DB-Cluster einer Aurora PostgreSQL Limitless Database in einen Amazon-S3-Bucket exportieren. Weitere Informationen erhalten Sie unter Importieren von Amazon S3 in einen Aurora-PostgreSQL-DB-Cluster und Exportieren von Daten aus einem/einer Aurora PostgreSQL-DB-Cluster zu Amazon S3.
- `btree_gin`
- `citext`
- `ip4r`
- `pg_buffercache`: Diese Erweiterung verhält sich in Aurora PostgreSQL Limitless Database anders als in der PostgreSQL-Community. Weitere Informationen finden Sie unter pg_buffercache-Unterschiede in Aurora PostgreSQL Limitless Database.
- `pg_stat_statements`
- `pg_trgm`
- `pgcrypto`
- `pgstattuple`: Diese Erweiterung verhält sich in Aurora PostgreSQL Limitless Database anders als in der PostgreSQL-Community. Weitere Informationen finden Sie unter pgstattuple-Unterschiede in Aurora PostgreSQL Limitless Database.
- `pgvector`
- `plpgsql`: Diese Erweiterung ist vorinstalliert, aber Sie können sie löschen.
- `PostGIS`: Lange Transaktionen und Tabellenverwaltungsfunktionen werden nicht unterstützt. Die Geo-Referenztabelle lässt sich nicht ändern.
- `unaccent`
- `uuid`

Die meisten PostgreSQL-Erweiterungen werden in Aurora PostgreSQL Limitless Database derzeit nicht unterstützt. Sie können jedoch weiterhin die Konfigurationseinstellung shared_preload_libraries (SPL) verwenden, um Erweiterungen in den Cluster der primären Aurora PostgreSQL-Datenbank zu laden. Sie werden auch in die Aurora PostgreSQL Limitless Database geladen, funktionieren dort aber möglicherweise nicht richtig.

So können Sie beispielsweise die Erweiterung `pg_hint_plan` zwar laden, es bleibt aber ungewiss, ob die in den Abfragekommentaren übergebenen Hinweise verwendet werden.

> **Note**
>
> Sie können keine Objekte ändern, die der Erweiterung pg_stat_statements zugeordnet sind. Informationen zur Installation von `pg_stat_statements` finden Sie unter limitless_stat_statements.

Sie können die Funktionen `pg_available_extensions` und `pg_available_extension_versions` verwenden, um die Erweiterungen zu finden, die in Aurora PostgreSQL Limitless Database unterstützt werden.

Die folgenden DDLs werden für Erweiterungen unterstützt:

CREATE EXTENSION

Sie können Erweiterungen wie in PostgreSQL erstellen.

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ CASCADE ]
```

Weitere Informationen finden Sie im Abschnitt CREATE EXTENSION der PostgreSQL-Dokumentation.

ALTER EXTENSION

Die folgenden DDLs werden unterstützt:

```
ALTER EXTENSION name UPDATE [ TO new_version ]
```

```
ALTER EXTENSION name SET SCHEMA new_schema
```

Weitere Informationen finden Sie im Abschnitt ALTER EXTENSION der PostgreSQL-Dokumentation.

DROP EXTENSION

Sie können Erweiterungen wie in PostgreSQL löschen.

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Weitere Informationen finden Sie im Abschnitt DROP EXTENSION der PostgreSQL-Dokumentation.

Die folgenden DDLs werden für Erweiterungen nicht unterstützt:

ALTER EXTENSION

Sie können keine Mitgliedsobjekte zu Erweiterungen hinzufügen oder daraus löschen.

```
ALTER EXTENSION name ADD member_object

ALTER EXTENSION name DROP member_object
```

pg_buffercache-Unterschiede in Aurora PostgreSQL Limitless Database

Wenn Sie in Aurora PostgreSQL Limitless Database die Erweiterung pg_buffercache installieren und die Ansicht `pg_buffercache` verwenden, erhalten Sie pufferbezogene Informationen nur von dem Knoten, mit dem Sie gerade verbunden sind: dem Router. Ähnlich erhalten Sie bei Verwendung der Funktion `pg_buffercache_summary` oder `pg_buffercache_usage_counts` nur Informationen vom verbundenen Knoten.

Sie können über zahlreiche Knoten verfügen und müssen möglicherweise für jeden Knoten einzeln auf Pufferinformationen zugreifen, um Probleme effektiv zu diagnostizieren. Daher stellt Limitless Database die folgenden Funktionen bereit:

- `rds_aurora.limitless_pg_buffercache(subcluster_id)`
- `rds_aurora.limitless_pg_buffercache_summary(subcluster_id)`
- `rds_aurora.limitless_pg_buffercache_usage_counts(subcluster_id)`

Durch Eingabe der Subcluster-ID eines beliebigen Knotens, unabhängig davon, ob es sich um einen Router oder einen Shard handelt, können Sie einfach auf die für diesen Knoten spezifischen Pufferinformationen zugreifen. Diese Funktionen sind direkt verfügbar, wenn Sie die Erweiterung `pg_buffercache` in Limitless Database installieren.

> **Note**
>
> Aurora PostgreSQL Limitless Database unterstützt diese Funktionen für Version 1.4 und höher der Erweiterung `pg_buffercache`.

Die in der Ansicht `limitless_pg_buffercache` angezeigten Spalten unterscheiden sich geringfügig von denen in der Ansicht `pg_buffercache`:

- `bufferid`: bleibt unverändert gegenüber `pg_buffercache`
- `relname`: Anstatt wie in `pg_buffercache` die Dateiknotennummer anzuzeigen, bildet `limitless_pg_buffercache` den zugehörigen `relname` ab, sofern dieser Name in der aktuellen Datenbank oder in gemeinsam genutzten Systemkatalogen verfügbar ist. Andernfalls wird `NULL` angezeigt.
- `parent_relname`: In dieser neuen Spalte, die in `pg_buffercache` nicht vorhanden ist, wird der übergeordnete `relname` angezeigt, wenn der Wert in der Spalte `relname` eine partitionierte Tabelle darstellt (im Fall von Sharded-Tabellen). Andernfalls wird `NULL` angezeigt.
- `spcname`: Anstatt wie in `pg_buffercache` den Tablespace-Objektbezeichner (OID) anzuzeigen, bildet `limitless_pg_buffercache` den Tablespace-Namen ab.
- `datname`: Anstatt wie in `pg_buffercache` die Datenbank-OID anzuzeigen, bildet `limitless_pg_buffercache` den Datenbanknamen ab.
- `relforknumber`: bleibt unverändert gegenüber `pg_buffercache`
- `relblocknumber`: bleibt unverändert gegenüber `pg_buffercache`
- `isdirty`: bleibt unverändert gegenüber `pg_buffercache`
- `usagecount`: bleibt unverändert gegenüber `pg_buffercache`
- `pinning_backends`: bleibt unverändert gegenüber `pg_buffercache`

Die Spalten in den Ansichten `limitless_pg_buffercache_summary` und `limitless_pg_buffercache_usage_counts` sind dieselben wie in den regulären Ansichten `pg_buffercache_summary` und `pg_buffercache_usage_counts`.

Mithilfe dieser Funktionen können Sie auf detaillierte Informationen zum Puffercache aller Knoten in Ihrer Limitless-Database-Umgebung zugreifen und so eine effektivere Diagnose und Verwaltung Ihrer Datenbanksysteme ermöglichen.

pgstattuple-Unterschiede in Aurora PostgreSQL Limitless Database

In Aurora PostgreSQL unterstützt die Erweiterung pgstattuple derzeit keine Fremdtabellen, partitionierten Tabellen oder partitionierten Indizes. In Aurora PostgreSQL Limitless Database gehören jedoch häufig von Benutzern erstellte Objekte zu diesen nicht unterstützten Typen. Zwar gibt es reguläre Tabellen und Indizes (z. B. Katalogtabellen und ihre Indizes), aber die meisten Objekte befinden sich auf fremden Knoten, was sie zu Fremdobjekten für den Router macht.

Wir wissen, wie wichtig diese Erweiterung für das Abrufen von Statistiken auf Tupel-Ebene ist – eine wichtige Voraussetzung, um Tabellen- und Indexüberlastungen vermeiden und Diagnoseinformationen erfassen zu können. Daher unterstützt Aurora PostgreSQL Limitless Database die Erweiterung `pgstattuple` in unbegrenzten Datenbanken.

Aurora PostgreSQL Limitless Database enthält die folgenden Funktionen im Schema `rds_aurora`:

Funktionen für Statistiken auf Tupel-Ebene

`rds_aurora.limitless_pgstattuple(`*`relation_name`*`)`

- Zweck: Statistiken auf Tupel-Ebene für Standardtabellen und deren Indizes extrahieren
- Eingabe: `relation_name` (Text): der Beziehungsname
- Ausgabe: Spalten, die mit denen übereinstimmen, die von der Funktion `pgstattuple` in Aurora PostgreSQL zurückgegeben werden

`rds_aurora.limitless_pgstattuple(`*`relation_name`*`, `*`subcluster_id`*`)`

- Zweck: Statistiken auf Tupel-Ebene für Referenztabellen, Sharded-Tabellen, Katalogtabellen und deren Indizes extrahieren
- Eingabe:
  - `relation_name` (Text): der Beziehungsname
  - `subcluster_id` (Text): die Subcluster-ID des Knotens, auf dem die Statistiken extrahiert werden sollen
- Ausgabe:
  - Bei Referenz- und Katalogtabellen (einschließlich ihrer Indizes) stimmen die Spalten mit denen in Aurora PostgreSQL überein.

- Bei Sharded-Tabellen stellen die Statistiken nur die Partition der Tabelle dar, die sich auf dem angegebenen Subcluster befindet.

Funktionen für Indexstatistiken

`rds_aurora.limitless_pgstatindex(`*`relation_name`*`)`

- Zweck: Statistiken für B-Baum-Indizes von Standardtabellen extrahieren
- Eingabe: `relation_name` (Text): der Name des B-Baum-Indexes
- Ausgabe: Alle Spalten außer `root_block_no` werden zurückgegeben. Die zurückgegebenen Spalten stimmen mit denen überein, die die Funktion `pgstatindex` in Aurora PostgreSQL liefert.

`rds_aurora.limitless_pgstatindex(`*`relation_name`*`, `*`subcluster_id`*`)`

- Zweck: Statistiken für B-Baum-Indizes von Referenz-, Sharded- und Katalogtabellen extrahieren
- Eingabe:
  - `relation_name` (Text): der Name des B-Baum-Indexes
  - `subcluster_id` (Text): die Subcluster-ID des Knotens, auf dem die Statistiken extrahiert werden sollen
- Ausgabe:
  - Bei Indizes von Referenz- und Katalogtabellen werden alle Spalten (außer `root_block_no`) zurückgegeben. Die zurückgegebenen Spalten stimmen mit denen in Aurora PostgreSQL überein.
  - Bei Sharded-Tabellen stellen die Statistiken nur die Partition des Indexes der Tabelle dar, die sich auf dem angegebenen Subcluster befindet. Die Spalte `tree_level` zeigt den Durchschnitt aller Tabellenausschnitte im angeforderten Subcluster.

`rds_aurora.limitless_pgstatginindex(`*`relation_name`*`)`

- Zweck: Statistiken für GINs (Generalized Inverted Indexes) von Standardtabellen extrahieren
- Eingabe: `relation_name` (Text): der Name des GIN
- Ausgabe: Spalten, die mit denen übereinstimmen, die von der Funktion `pgstatginindex` in Aurora PostgreSQL zurückgegeben werden

`rds_aurora.limitless_pgstatginindex(`*`relation_name`*`, `*`subcluster_id`*`)`

- Zweck: Statistiken für GINs von Referenz-, Sharded- und Katalogtabellen extrahieren
- Eingabe:

    - `relation_name` (Text): der Name des Indexes
    - `subcluster_id` (Text): die Subcluster-ID des Knotens, auf dem die Statistiken extrahiert werden sollen
- Ausgabe:
    - Bei GINs von Referenz- und Katalogtabellen stimmen die Spalten mit denen in Aurora PostgreSQL überein.
    - Bei Sharded-Tabellen stellen die Statistiken nur die Partition des Indexes der Tabelle dar, die sich auf dem angegebenen Subcluster befindet.

`rds_aurora.limitless_pgstathashindex(`*`relation_name`*`)`

- Zweck: Statistiken für Hash-Indizes von Standardtabellen extrahieren
- Eingabe: `relation_name` (Text): der Name des Hash-Indexes
- Ausgabe: Spalten, die mit denen übereinstimmen, die von der Funktion `pgstathashindex` in Aurora PostgreSQL zurückgegeben werden

`rds_aurora.limitless_pgstathashindex(`*`relation_name`*`, `*`subcluster_id`*`)`

- Zweck: Statistiken für Hash-Indizes von Referenz-, Sharded- und Katalogtabellen extrahieren
- Eingabe:
    - `relation_name` (Text): der Name des Indexes
    - `subcluster_id` (Text): die Subcluster-ID des Knotens, auf dem die Statistiken extrahiert werden sollen
- Ausgabe:
    - Bei Hash-Indizes von Referenz- und Katalogtabellen stimmen die Spalten mit denen in Aurora PostgreSQL überein.
    - Bei Sharded-Tabellen stellen die Statistiken nur die Partition des Indexes der Tabelle dar, die sich auf dem angegebenen Subcluster befindet.

Funktionen zur Seitenanzahl

`rds_aurora.limitless_pg_relpages(`*`relation_name`*`)`

- Zweck: die Seitenanzahl für Standardtabellen und deren Indizes extrahieren
- Eingabe: `relation_name` (Text): der Beziehungsname
- Ausgabe: die Seitenanzahl der angegebenen Beziehung

`rds_aurora.limitless_pg_relpages(`*`relation_name`*`, `*`subcluster_id`*`)`

- Zweck: die Seitenanzahl für Referenz-, Sharded- und Katalogtabellen (einschließlich ihrer Indizes) extrahieren
- Eingabe:
  - `relation_name` (Text): der Beziehungsname
  - `subcluster_id` (Text): die Subcluster-ID des Knotens, auf dem die Seitenanzahl extrahiert werden soll
- Ausgabe: Bei Sharded-Tabellen entspricht die Seitenanzahl der Summe der Seiten aus allen Tabellenausschnitten des angegebenen Subclusters.

Funktionen für annähernde Statistiken auf Tupel-Ebene

`rds_aurora.limitless_pgstattuple_approx(`*`relation_name`*`)`

- Zweck: annähernde Statistiken auf Tupel-Ebene für Standardtabellen und deren Indizes extrahieren
- Eingabe: `relation_name` (Text): der Beziehungsname
- Ausgabe: Spalten, die mit denen übereinstimmen, die von der Funktion pgstattuple_approx in Aurora PostgreSQL zurückgegeben werden

`rds_aurora.limitless_pgstattuple_approx(`*`relation_name`*`, `*`subcluster_id`*`)`

- Zweck: annähernde Statistiken auf Tupel-Ebene für Referenz-, Sharded- und Katalogtabellen (sowie deren Indizes) extrahieren
- Eingabe:
  - `relation_name` (Text): der Beziehungsname
  - `subcluster_id` (Text): die Subcluster-ID des Knotens, auf dem die Statistiken extrahiert werden sollen
- Ausgabe:
  - Bei Referenz- und Katalogtabellen (einschließlich ihrer Indizes) stimmen die Spalten mit denen in Aurora PostgreSQL überein.
  - Bei Sharded-Tabellen stellen die Statistiken nur die Partition der Tabelle dar, die sich auf dem angegebenen Subcluster befindet.

> **Note**
>
> Derzeit unterstützt Aurora PostgreSQL Limitless Database die Erweiterung `pgstattuple` für materialisierte Ansichten, TOAST-Tabellen oder temporäre Tabellen nicht.

In Aurora PostgreSQL Limitless Database müssen Sie die Eingabe als Text bereitstellen, obwohl Aurora PostgreSQL auch andere Formate unterstützt.

Fremdschlüssel

Beschränkungen bezüglich Fremdschlüsseln (`FOREIGN KEY`) werden mit einigen Einschränkungen unterstützt:

- `CREATE TABLE` mit `FOREIGN KEY` wird nur für Standardtabellen unterstützt. Wenn Sie `FOREIGN KEY` verwenden möchten, um eine Sharded- oder Referenztabelle anzufertigen, erstellen Sie zunächst die Tabelle ohne Fremdschlüsselbeschränkung. Ändern Sie sie dann mit der folgenden Anweisung:

  ```
  ALTER TABLE ADD CONSTRAINT;
  ```

- Eine Standardtabelle kann nicht in eine Sharded- oder Referenztabelle konvertiert werden, wenn für die Tabelle eine Fremdschlüsselbeschränkung gilt. Löschen Sie die Beschränkung und fügen Sie sie nach der Konvertierung hinzu.
- Die folgenden Einschränkungen gelten für die verschiedenen Tabellentypen, was Fremdschlüsselbeschränkungen betrifft:
  - Für eine Standardtabelle kann eine Fremdschlüsselbeschränkung hinsichtlich einer anderen Standardtabelle gelten.
  - Für eine Sharded-Tabelle kann eine Fremdschlüsselbeschränkung gelten, wenn die übergeordneten und untergeordneten Tabellen nebeneinander angeordnet sind und der Fremdschlüssel eine Obermenge des Shard-Schlüssels bildet.
  - Für eine Sharded-Tabelle kann eine Fremdschlüsselbeschränkung hinsichtlich einer Referenztabelle gelten.
  - Für eine Referenztabelle kann eine Fremdschlüsselbeschränkung hinsichtlich einer anderen Referenztabelle gelten.

Themen

- Fremdschlüsseloptionen
- Beispiele

Fremdschlüsseloptionen

Fremdschlüssel werden in Aurora PostgreSQL Limitless Database für einige DDL-Optionen unterstützt. In der folgenden Tabelle sind Optionen aufgeführt, die zwischen Tabellen in Aurora PostgreSQL Limitless Database unterstützt und nicht unterstützt werden.

DDL-Option	Referenz zu Referenz	Sharded zu Sharded (nebeneinander angeordnet)	Sharded zu Referenz	Standard zu Standard
`DEFERRABLE`	Ja	Ja	Ja	Ja
`INITIALLY DEFERRED`	Ja	Ja	Ja	Ja
`INITIALLY IMMEDIATE`	Ja	Ja	Ja	Ja
`MATCH FULL`	Ja	Ja	Ja	Ja
`MATCH PARTIAL`	Nein	Nein	Nein	Nein
`MATCH SIMPLE`	Ja	Ja	Ja	Ja
`NOT DEFERRABLE`	Ja	Ja	Ja	Ja
`NOT VALID`	Ja	Nein	Nein	Ja
`ON DELETE CASCADE`	Ja	Ja	Ja	Ja
`ON DELETE NO ACTION`	Ja	Ja	Ja	Ja
`ON DELETE RESTRICT`	Ja	Ja	Ja	Ja
`ON DELETE SET DEFAULT`	Nein	Nein	Nein	Nein
`ON DELETE SET NULL`	Ja	Nein	Nein	Ja

DDL-Option	Referenz zu Referenz	Sharded zu Sharded (nebeneinander angeordnet)	Sharded zu Referenz	Standard zu Standard
`ON UPDATE CASCADE`	Nein	Nein	Nein	Ja
`ON UPDATE NO ACTION`	Ja	Ja	Ja	Ja
`ON UPDATE RESTRICT`	Ja	Ja	Ja	Ja
`ON UPDATE SET DEFAULT`	Nein	Nein	Nein	Nein
`ON UPDATE SET NULL`	Ja	Nein	Nein	Ja

Beispiele

- Standard zu Standard:

```
set rds_aurora.limitless_create_table_mode='standard';

CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);

SELECT constraint_name, table_name, constraint_type
FROM information_schema.table_constraints WHERE constraint_type='FOREIGN KEY';
```

```
 constraint_name         | table_name  | constraint_type
-------------------------+-------------+-----------------
 orders_product_no_fkey  | orders      | FOREIGN KEY
(1 row)
```

- Sharded zu Sharded (nebeneinander angeordnet):

```
set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"product_no"}';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
set rds_aurora.limitless_create_table_collocate_with='products';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- Sharded zu Referenz:

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{"order_id"}';
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);
```

```
ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

- Referenz zu Referenz:

```
set rds_aurora.limitless_create_table_mode='reference';
CREATE TABLE products(
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    quantity integer
);

ALTER TABLE orders ADD CONSTRAINT order_product_fk FOREIGN KEY (product_no)
 REFERENCES products (product_no);
```

Funktionen

Funktionen werden in Aurora PostgreSQL Limitless Database unterstützt.

Die folgenden DDLs werden für Funktionen unterstützt:

CREATE FUNCTION

Mit diesem Befehl können Sie Funktionen wie in Aurora PostgreSQL erstellen. Sie können damit nur nicht ihre Volatilität ändern, während Sie sie ersetzen.

Weitere Informationen finden Sie im Abschnitt CREATE FUNCTION der PostgreSQL-Dokumentation.

ALTER FUNCTION

Mit diesem Befehl können Sie Funktionen wie in Aurora PostgreSQL ändern, nur nicht ihre Volatilität.

Weitere Informationen finden Sie im Abschnitt ALTER FUNCTION der PostgreSQL-Dokumentation.

DROP FUNCTION

Sie können Funktionen wie in Aurora PostgreSQL löschen.

```
DROP FUNCTION [ IF EXISTS ] name [ ( [ [ argmode ] [ argname ] argtype [, ...] ] ) ]
 [, ...]
    [ CASCADE | RESTRICT ]
```

Weitere Informationen finden Sie im Abschnitt DROP FUNCTION der PostgreSQL-Dokumentation.

Themen

- Verteilung der Funktionen
- Funktionsvolatilität

Verteilung der Funktionen

Wenn alle Anweisungen einer Funktion auf einen einzelnen Shard ausgerichtet sind, empfiehlt es sich, die gesamte Funktion auf den Ziel-Shard zu übertragen. Auf diese Weise wird das Ergebnis

zurück an den Router propagiert und die Funktion wird nicht am Router selbst aufgedröselt. Die Pushdown-Option für Funktionen und gespeicherte Verfahren ist nützlich für Kunden, die ihre Funktion oder ihr gespeichertes Verfahren näher an der Datenquelle, also dem Shard, ausführen möchten.

Um eine Funktion zu verteilen, müssen Sie sie zuerst erstellen und dann das Verfahren `rds_aurora.limitless_distribute_function` aufrufen. Für diese Funktion gilt folgende Syntax:

```
SELECT rds_aurora.limitless_distribute_function('function_prototype',
 ARRAY['shard_key'], 'collocating_table');
```

Die Funktion nutzt die folgenden Parameter:

- *`function_prototype`*: die Funktion, die verteilt werden soll. Erwähnen Sie nur die Eingabeargumente und keines der Ausgabeargumente.

  Wenn eines der Argumente als OUT-Parameter definiert ist, nehmen Sie seinen Typ nicht in die Argumente von `function_prototype` auf.
- `ARRAY['`*`shard_key`*`']`: die Liste der Funktionsargumente, die als Shard-Schlüssel für die Funktion identifiziert wurden
- *`collocating_table`*: die Sharded-Tabelle, die den Datenbereich auf dem Ziel-Shard enthält

Um den Shard zu identifizieren, auf dem diese Funktion ausgeführt werden soll, hasht das System das Argument `ARRAY['`*`shard_key`*`']` und erkennt in *`collocating_table`* den Shard, der den Bereich mit diesem Hash-Wert hostet.

Einschränkungen

Wenn Sie eine Funktion oder ein Verfahren verteilen, befasst sie oder es sich nur mit Daten, die durch den Shard-Schlüsselbereich in diesem Shard begrenzt sind. In Fällen, in denen Funktionen oder Verfahren versuchen, auf Daten von einem anderen Shard zuzugreifen, unterscheiden sich die von den verteilten Funktionen oder Verfahren zurückgegeben Ergebnisse von denen der nicht verteilten Funktionen oder Verfahren.

Sie erstellen beispielsweise eine Funktion, die Abfragen enthält, die mehrere Shards betreffen werden, rufen dann aber das Verfahren `rds_aurora.limitless_distribute_function` auf, um sie zu verteilen. Wenn Sie diese Funktion aufrufen, indem Sie Argumente für einen Shard-

Schlüssel angeben, werden die Ergebnisse ihrer Ausführung wahrscheinlich durch die in diesem Shard vorhandenen Werte begrenzt. Die Ergebnisse unterscheiden sich von denen, die ohne Verteilung der Funktion generiert wurden.

Beispiele

Nehmen wir an, wir haben die Funktion `func`, die Sharded-Tabelle `customers` und den Shard-Schlüssel `customer_id`.

```
postgres_limitless=> CREATE OR REPLACE FUNCTION func(c_id integer, sc integer)
 RETURNS int
    language SQL
    volatile
    AS $$
    UPDATE customers SET score = sc WHERE customer_id = c_id RETURNING score;
    $$;
```

Jetzt verteilen wir diese Funktion:

```
SELECT rds_aurora.limitless_distribute_function('func(integer, integer)',
 ARRAY['c_id'], 'customers');
```

Im Folgenden sehen Sie Beispiel-Abfragepläne.

```
EXPLAIN(costs false, verbose true) SELECT func(27+1,10);

                    QUERY PLAN
 ------------------------------------------------
  Foreign Scan
    Output: (func((27 + 1), 10))
    Remote SQL:  SELECT func((27 + 1), 10) AS func
  Single Shard Optimized
 (4 rows)
```

```
EXPLAIN(costs false, verbose true)
 SELECT * FROM customers,func(customer_id, score) WHERE customer_id=10 AND score=27;

                          QUERY PLAN
 ------------------------------------------------------------------------
  Foreign Scan
    Output: customer_id, name, score, func
```

```
    Remote SQL:  SELECT customers.customer_id,
      customers.name,
      customers.score,
      func.func
     FROM public.customers,
      LATERAL func(customers.customer_id, customers.score) func(func)
    WHERE ((customers.customer_id = 10) AND (customers.score = 27))
  Single Shard Optimized
 (10 rows)
```

Das folgende Beispiel zeigt ein Verfahren mit IN- und OUT-Parametern als Argumente.

```
CREATE OR REPLACE FUNCTION get_data(OUT id INTEGER, IN arg_id INT)
    AS $$
    BEGIN
        SELECT customer_id,
        INTO id
        FROM customer
        WHERE customer_id = arg_id;
    END;
    $$ LANGUAGE plpgsql;
```

Das folgende Beispiel verteilt das Verfahren nur unter Verwendung von IN-Parametern.

```
EXPLAIN(costs false, verbose true) SELECT * FROM get_data(1);

             QUERY PLAN
 ----------------------------------
  Foreign Scan
    Output: id
    Remote SQL:  SELECT customer_id
     FROM get_data(1) get_data(id)
  Single Shard Optimized
 (6 rows)
```

Funktionsvolatilität

Sie können feststellen, ob eine Funktion unveränderlich, stabil oder volatil ist, indem Sie den Wert `provolatile` in der Ansicht pg_proc überprüfen. Der Wert `provolatile` gibt an, ob das Ergebnis der Funktion nur von ihren Eingabeargumenten abhängt oder auch von externen Faktoren beeinflusst wird.

Der Wert ist einer der folgenden:

- `i`: unveränderliche Funktionen, die immer dasselbe Ergebnis für dieselben Eingaben liefern
- `s`: stabile Funktionen, deren Ergebnisse (für feste Eingaben) sich innerhalb eines Scans nicht ändern
- `v`: flüchtige Funktionen, deren Ergebnisse sich jederzeit ändern können Nutzen Sie `v` auch für Funktionen mit Nebeneffekten, sodass deren Aufrufe nicht einfach optimiert werden können.

Die folgenden Beispiele zeigen flüchtige Funktionen.

```
SELECT proname, provolatile FROM pg_proc WHERE proname='pg_sleep';

 proname  | provolatile
----------+-------------
 pg_sleep | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='uuid_generate_v4';

     proname      | provolatile
------------------+-------------
 uuid_generate_v4 | v
(1 row)

SELECT proname, provolatile FROM pg_proc WHERE proname='nextval';

 proname | provolatile
---------+-------------
 nextval | v
(1 row)
```

Die Volatilität einer vorhandenen Funktion lässt sich in Aurora PostgreSQL Limitless Database nicht ändern. Dies gilt gleichermaßen für Befehle des Typs `ALTER FUNCTION` und des Typs `CREATE OR REPLACE FUNCTION`, wie in den folgenden Beispielen gezeigt.

```
-- Create an immutable function
CREATE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
```

```
$$IMMUTABLE;

-- Altering the volatility throws an error
ALTER FUNCTION immutable_func1 STABLE;

-- Replacing the function with altered volatility throws an error
CREATE OR REPLACE FUNCTION immutable_func1(name text) RETURNS text language plpgsql
AS $$
BEGIN
    RETURN name;
END;
$$VOLATILE;
```

Wir empfehlen Ihnen dringend, Funktionen die richtigen Volatilitäten zuzuweisen. Wenn Ihre Funktion beispielsweise `SELECT` aus mehreren Tabellen verwendet oder auf Datenbankobjekte verweist, legen Sie sie nicht als `IMMUTABLE` fest. Wenn sich der Tabelleninhalt jemals ändert, geht damit die Unveränderlichkeit verloren.

Aurora PostgreSQL erlaubt `SELECT` innerhalb von unveränderlichen Funktionen, aber die Ergebnisse könnten falsch sein. Aurora PostgreSQL Limitless Database kann sowohl Fehler als auch falsche Ergebnisse zurückgeben. Weitere Informationen finden Sie im Abschnitt Volatilitätskategorien von Funktionen der PostgreSQL-Dokumentation.

Sequenzen

Benannte Sequenzen sind Datenbankobjekte, die eindeutige Zahlen in auf- oder absteigender Reihenfolge generieren. `CREATE SEQUENCE` erstellt einen Generator für neue Sequenznummern. Sequenzwerte sind garantiert eindeutig.

Wenn Sie eine benannte Sequenz in Aurora PostgreSQL Limitless Database erstellen, wird ein verteiltes Sequenzobjekt erstellt. Daraufhin verteilt Aurora PostgreSQL Limitless Database nicht überlappende Blöcke von Sequenzwerten auf alle Distributed Transaction Router (Router). Blöcke werden auf Routern als lokale Sequenzobjekte dargestellt; daher werden Sequenzoperationen wie `nextval` und `currval` lokal ausgeführt. Router arbeiten unabhängig voneinander und fordern bei Bedarf neue Blöcke aus der verteilten Sequenz an.

Weitere Informationen zu Sequenzen finden Sie im Abschnitt CREATE SEQUENCE der PostgreSQL-Dokumentation.

Themen

- Anfordern eines neuen Blocks
- Einschränkungen
- Nicht unterstützte Optionen
- Beispiele
- Sequenzansichten
- Beheben von Sequenzproblemen

Anfordern eines neuen Blocks

Sie konfigurieren die Größe der Blöcke, die Routern zugewiesen werden, mithilfe des Parameters `rds_aurora.limitless_sequence_chunk_size`. Der Standardwert ist `250000`. Jeder Router besitzt zunächst zwei Blöcke: einen aktiven und einen reservierten. Aktive Blöcke werden verwendet, um lokale Sequenzobjekte zu konfigurieren (Einstellung `minvalue` und`maxvalue`), und reservierte Blöcke werden in einer internen Katalogtabelle gespeichert. Wenn ein aktiver Block den Mindest- oder Höchstwert erreicht, wird er durch den reservierten Block ersetzt. Dazu wird `ALTER SEQUENCE` intern verwendet, was bedeutet, dass `AccessExclusiveLock` erworben wird.

Hintergrund-Worker werden alle 10 Sekunden auf Router-Knoten ausgeführt, um Sequenzen nach verwendeten reservierten Blöcken zu durchsuchen. Wenn ein verwendeter Block gefunden wird, fordert der Worker bei der verteilten Sequenz einen neuen Block an. Achten Sie darauf,

die Blockgröße so festzulegen, dass die Hintergrund-Worker genügend Zeit haben, neue Blöcke anzufordern. Remoteanfragen treten niemals im Kontext von Benutzersitzungen auf, was bedeutet, dass Sie nicht direkt eine neue Sequenz anfordern können.

Einschränkungen

Die folgenden Einschränkungen gelten für Sequenzen in Aurora PostgreSQL Limitless Database:

- Der `pg_sequence`-Katalog, die `pg_sequences`-Funktion und die `SELECT * FROM` *`sequence_name`*-Anweisung zeigen alle nur den lokalen Sequenzstatus, nicht den verteilten Status.
- Sequenzwerte sind garantiert eindeutig und innerhalb einer Sitzung garantiert monoton. Sie können jedoch bei `nextval`-Anweisungen, die in anderen Sitzungen ausgeführt werden, in Unordnung geraten, wenn diese Sitzungen mit anderen Routern verbunden sind.
- Stellen Sie sicher, dass die Sequenzgröße (Anzahl der verfügbaren Werte) ausreicht, um auf alle Router verteilt zu werden. Verwenden Sie den Parameter `rds_aurora.limitless_sequence_chunk_size`, um die `chunk_size` zu konfigurieren. (Jeder Router hat zwei Blöcke.)
- Die `CACHE`-Option wird unterstützt, aber der Cache muss kleiner sein als `chunk_size`.

Nicht unterstützte Optionen

Die folgenden Optionen werden für Sequenzen in Aurora PostgreSQL Limitless Database nicht unterstützt.

Sequenz-Bearbeitungsfunktionen

Die Funktion `setval` wird nicht unterstützt. Weitere Informationen finden Sie im Abschnitt Sequenz-Bearbeitungsfunktionen in der PostgreSQL-Dokumentation.

CREATE SEQUENCE

Die folgenden Optionen werden nicht unterstützt.

```
CREATE [{ TEMPORARY | TEMP} | UNLOGGED] SEQUENCE
    [[ NO ] CYCLE]
```

Weitere Informationen finden Sie im Abschnitt CREATE SEQUENCE der PostgreSQL-Dokumentation.

ALTER SEQUENCE

Die folgenden Optionen werden nicht unterstützt.

```
ALTER SEQUENCE
    [[ NO ] CYCLE]
```

Weitere Informationen finden Sie im Abschnitt ALTER SEQUENCE der PostgreSQL-Dokumentation.

ALTER TABLE

Der Befehl `ALTER TABLE` wird für Sequenzen nicht unterstützt.

Beispiele

CREATE/DROP SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
       1
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16960 |       20 |        1 |            1 |  10000 |      1 |        1 | f
(1 row)

% connect to another router
postgres_limitless=> SELECT nextval('s');

 nextval
---------
   10001
```

```
(1 row)

postgres_limitless=> SELECT * FROM pg_sequence WHERE seqrelid='s'::regclass;

 seqrelid | seqtypid | seqstart | seqincrement | seqmax | seqmin | seqcache |
 seqcycle
----------+----------+----------+--------------+--------+--------+----------
+----------
    16959 |       20 |    10001 |            1 |  20000 |  10001 |        1 | f
(1 row)

postgres_limitless=> DROP SEQUENCE s;
DROP SEQUENCE
```

ALTER SEQUENCE

```
postgres_limitless=> CREATE SEQUENCE s;
CREATE SEQUENCE

postgres_limitless=> ALTER SEQUENCE s RESTART 500;
ALTER SEQUENCE

postgres_limitless=> SELECT nextval('s');

 nextval
---------
     500
(1 row)

postgres_limitless=> SELECT currval('s');

 currval
---------
     500
(1 row)
```

Sequenz-Bearbeitungsfunktionen

```
postgres=# CREATE TABLE t(a bigint primary key, b bigint);
CREATE TABLE

postgres=# CREATE SEQUENCE s minvalue 0 START 0;
CREATE SEQUENCE
```

```
postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));
INSERT 0 1

postgres=# SELECT * FROM t;

 a | b
---+---
 0 | 0
 1 | 1
(2 rows)

postgres=# ALTER SEQUENCE s RESTART 10000;
ALTER SEQUENCE

postgres=# INSERT INTO t VALUES (nextval('s'), currval('s'));

INSERT 0 1

postgres=# SELECT * FROM t;

   a   |   b
-------+-------
     0 |     0
     1 |     1
 10000 | 10000
(3 rows)
```

Sequenzansichten

Aurora PostgreSQL Limitless Database bietet die folgenden Ansichten für Sequenzen.

rds_aurora.limitless_distributed_sequence

Diese Ansicht zeigt den Status und die Konfiguration einer verteilten Sequenz. Die Spalten `minvalue`, `maxvalue`, `start`, `inc` und `cache` haben dieselbe Bedeutung wie in der Ansicht pg_sequences und zeigen die Optionen, mit denen die Sequenz erstellt wurde. Die Spalte `lastval` zeigt den letzten zugewiesenen oder reservierten Wert in einem verteilten Sequenzobjekt. Dies bedeutet nicht, dass der Wert bereits verwendet wurde, da Router Sequenzblöcke lokal speichern.

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_distributed_sequence WHERE
 sequence_name='test_serial_b_seq';

 schema_name |   sequence_name   | lastval | minvalue |  maxvalue  | start | inc |
 cache
-------------+-------------------+---------+----------+------------+-------+-----
+-------
 public      | test_serial_b_seq | 1250000 |        1 | 2147483647 |     1 |   1 |
   1
(1 row)
```

rds_aurora.limitless_sequence_metadata

Diese Ansicht zeigt verteilte Sequenzmetadaten und aggregiert Sequenzmetadaten von Cluster-Knoten. Sie verwendet die folgenden Spalten:

- `subcluster_id`: die ID des Cluster-Knotens, dem ein Block gehört
- Aktiver Block: ein Block einer Sequenz, der verwendet wird (`active_minvalue`, `active_maxvalue`)
- Reservierter Block: der lokale Block, der als Nächstes verwendet wird (`reserved_minvalue`, `reserved_maxvalue`)
- `local_last_value`: der zuletzt beobachtete Wert aus einer lokalen Sequenz
- `chunk_size`: die Größe eines Blocks, wie sie bei der Erstellung konfiguriert wurde

```
postgres_limitless=> SELECT * FROM rds_aurora.limitless_sequence_metadata WHERE
 sequence_name='test_serial_b_seq' order by subcluster_id;
```

```
 subcluster_id |   sequence_name   | schema_name | active_minvalue | active_maxvalue
 | reserved_minvalue | reserved_maxvalue | chunk_size | chunk_state |
 local_last_value
---------------+-------------------+-------------+-----------------
+-----------------+-------------------+-------------------+------------
+-------------+------------------
 1             | test_serial_b_seq | public      |          500001 |          750000
 |           1000001 |           1250000 |     250000 |           1 |
 550010
 2             | test_serial_b_seq | public      |          250001 |          500000
 |            750001 |           1000000 |     250000 |           1 |

(2 rows)
```

Beheben von Sequenzproblemen

Die folgenden Probleme können bei Sequenzen auftreten.

Blockgröße reicht nicht aus

Wenn für die Blockgröße ein zu geringer Wert festgelegt wurde und die Transaktionsrate hoch ist, haben die Hintergrund-Worker möglicherweise nicht genug Zeit, um neue Blöcke anzufordern, bevor die aktiven Blöcke verbraucht sind. Dies kann zu Konflikten und Warteereignissen wie `LIMITLESS:AuroraLimitlessSequenceReplace`, `LWLock:LockManager`, `Lockrelation` und `LWlock:bufferscontent` führen.

Erhöhen Sie den Wert des Parameters `rds_aurora.limitless_sequence_chunk_size`.

Sequenz-Cache zu hoch

In PostgreSQL erfolgt das Sequenz-Caching auf Sitzungsebene. Jede Sitzung weist während eines Zugriffs auf das Sequenzobjekt aufeinanderfolgende Sequenzwerte zu und erhöht den `last_value` des Sequenzobjekts entsprechend. Sobald `nextval` innerhalb dieser Sitzung erneut verwendet wird, werden dann einfach die vorab zugewiesenen Werte zurückgegeben, ohne dass auf das Sequenzobjekt zugegriffen wird.

Alle Nummern, die innerhalb einer Sitzung zugewiesen, aber nicht verwendet wurden, gehen verloren, wenn die Sitzung endet, was zu „Lücken“ in der Sequenz führt. Dies kann zu einem raschen Verbrauch des sequence_chunk und somit zu Konflikten und Warteereignissen wie `LIMITLESS:AuroraLimitlessSequenceReplace`, `LWLock:LockManager`, `Lockrelation` und `LWlock:bufferscontent` führen.

Verringern Sie den festgelegten Wert für den Sequenz-Cache.

Die folgende Abbildung zeigt Warteereignisse, die durch Sequenzprobleme verursacht wurden.

Unterstützte und nicht unterstützte Data Manipulation Language (DML)- und SQL-Befehle zur Abfrageverarbeitung

In der folgenden Tabelle sind die DML-Befehle aufgeführt, die von Aurora PostgreSQL Limitless Database unterstützt und nicht unterstützt werden, mit Verweisen auf Einschränkungen oder weitere Informationen.

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
ABORT	Ja	Keine
ANALYZE	Ja	ANALYZE
BEGIN	Ja	Keine
CALL	Ja	Keine
CHECKPOINT	Ja	Keine
CLOSE	Ja	Keine
CLUSTER	Ja	CLUSTER
COMMIT	Ja	Keine
COMMIT PREPARED	Nein	Nicht zutreffend
COPY	Ja	Keine
DEALLOCATE	Ja	Keine
DECLARE	Ja	Keine
DELETE	Ja	Keine
DISCARD	Ja	Keine
DO	Ja	Keine
END	Ja	Keine

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
EXECUTE	Ja	Keine
EXPLAIN	Ja	EXPLAIN
FETCH	Ja	Keine
IMPORT FOREIGN SCHEMA	Nein	Nicht zutreffend
INSERT	Ja	INSERT
LISTEN	Nein	Nicht zutreffend
LOCK	Ja	Keine
MERGE	Nein	Nicht zutreffend
MOVE	Ja	Keine
NOTIFY	Nein	Nicht zutreffend
OPEN	Ja	Keine
PREPARE	Ja	Keine
PREPARE TRANSACTION	Nein	Nicht zutreffend
REFRESH MATERIALIZED VIEW	Nein	Nicht zutreffend
REINDEX	Nein	Nicht zutreffend
RELEASE SAVEPOINT	Ja	Keine
ROLLBACK	Ja	Keine
ROLLBACK PREPARED	Nein	Nicht zutreffend
ROLLBACK TO SAVEPOINT	Ja	Keine

Befehl	Unterstützt?	Einschränkungen oder weitere Informationen
SAVEPOINT	Ja	Keine
SELECT	Ja	Keine
SELECT INTO	Ja	Keine
ZEIGEN	Ja	Keine
START TRANSACTION	Ja	Keine
UNLISTEN	Nein	Keine
UPDATE	Ja	UPDATE
UPDATE … WHERE CURRENT OF	Nein	Nicht zutreffend
VACUUM	Ja	VACUUM
VALUES	Ja	Keine

DML-Einschränkungen und andere Informationen für Aurora PostgreSQL Limitless Database

In den folgenden Themen werden Einschränkungen beschrieben oder weitere Informationen zu DML und zu SQL-Befehlen für die Abfrageverarbeitung in Aurora PostgreSQL Limitless Database bereitgestellt.

Themen

- ANALYZE
- CLUSTER
- EXPLAIN
- INSERT
- UPDATE
- VACUUM

ANALYZE

Der Befehl `ANALYZE` sammelt Statistiken über den Inhalt von Tabellen in der Datenbank. Anschließend verwendet der Abfrageplaner diese Statistiken, um die effizientesten Ausführungspläne für Abfragen zu ermitteln. Weitere Informationen finden Sie unter ANALYZE in der PostgreSQL-Dokumentation.

In Aurora PostgreSQL Limitless Database sammelt der Befehl `ANALYZE` bei seiner Ausführung Tabellenstatistiken für alle Router und Shards.

Um zu verhindern, dass während der Ausführung von `ANALYZE` Statistiken für jeden Router berechnet werden, werden Tabellenstatistiken auf einem der Router berechnet und dann auf Peer-Router kopiert.

CLUSTER

Der Befehl `CLUSTER` ordnet eine Tabelle anhand eines Indexes physisch neu an. Der Index muss zuvor für die Tabelle definiert worden sein. In Aurora PostgreSQL Limitless Database findet das Clustering lokal für den Teil des Index statt, der auf jedem Shard vorhanden ist.

Weitere Informationen finden Sie unter CLUSTER in der PostgreSQL-Dokumentation.

EXPLAIN

Sie verwenden den folgenden Parameter, um die Ausgabe des Befehls `EXPLAIN` zu konfigurieren:

- `rds_aurora.limitless_explain_options`: Bestimmt, was in die Ausgabe `EXPLAIN` aufgenommen werden soll. Der Standardwert lautet `single_shard_optimization`: Er zeigt an, ob Pläne für Single-Shard optimiert sind, Shard-Pläne sind jedoch nicht enthalten.

In diesem Beispiel werden in der Ausgabe `EXPLAIN` keine Pläne aus Shards angezeigt.

```
postgres_limitless=> EXPLAIN SELECT * FROM employees where id =25;

                      QUERY PLAN
------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
 Single Shard Optimized
(2 rows)
```

Jetzt legen wir den Wert `rds_aurora.limitless_explain_options` fest, damit `shard_plans` und `single_shard_optimization` eingeschlossen sind. Wir können die Ausführungspläne von Anweisungen sowohl auf Routern als auch auf Shards anzeigen lassen. Außerdem deaktivieren wir den Parameter `enable_seqscan`, um zu erzwingen, dass der Indexscan auf Shard-Ebene verwendet wird.

```
postgres_limitless=> SET rds_aurora.limitless_explain_options = shard_plans,
 single_shard_optimization;
SET

postgres_limitless=> SET enable_seqscan = OFF;
SET

postgres_limitless=> EXPLAIN SELECT * FROM employees WHERE id = 25;

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Foreign Scan  (cost=100.00..101.00 rows=100 width=0)
   Remote Plans from Shard postgres_s4:
         Index Scan using employees_ts00287_id_idx on employees_ts00287
 employees_fs00003  (cost=0.14..8.16 rows=1 width=15)
           Index Cond: (id = 25)
 Single Shard Optimized
```

```
(5 rows)
```

Weitere Informationen zum Befehl `EXPLAIN` finden Sie in der PostgreSQL-Dokumentation unter EXPLAIN.

INSERT

Die meisten `INSERT`-Befehle werden in Aurora PostgreSQL Limitless Database unterstützt.

PostgreSQL hat keinen expliziten `UPSERT`-Befehl, unterstützt aber `INSERT ... ON CONFLICT`-Anweisungen.

`INSERT ... ON CONFLICT` wird nicht unterstützt, wenn die Konfliktaktion eine Unterabfrage oder eine veränderbare Funktion hat:

```
-- RANDOM is a mutable function.
INSERT INTO sharded_table VALUES (1, 100) ON CONFLICT (id) DO UPDATE SET other_id =
 RANDOM();

ERROR: Aurora Limitless Tables doesn't support pushdown-unsafe functions with DO UPDATE
 clauses.
```

Weitere Informationen zum Befehl `INSERT` finden Sie in der PostgreSQL-Dokumentation unter INSERT.

UPDATE

Das Aktualisieren des Shard-Schlüssels wird nicht unterstützt. Sie haben beispielsweise eine Sharded-Tabelle namens `customers` mit dem Shard-Schlüssel `customer_id`. Die folgenden DML-Anweisungen verursachen Fehler:

```
postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_id =1;
ERROR:  Shard key column update is not supported

postgres_limitless=> UPDATE customers SET customer_id = 11 WHERE customer_name='abc';
ERROR:  Shard key column update is not supported
```

Um einen Shard-Schlüssel zu aktualisieren, müssen Sie zuerst die Zeile mit dem Shard-Schlüssel löschen (`DELETE`) und dann eine neue Zeile mit dem aktualisierten Shard-Schlüsselwert einfügen (`INSERT`).

Weitere Informationen zum Befehl `UPDATE` finden Sie in der PostgreSQL-Dokumentation unter Aktualisieren von Daten.

VACUUM

Sie können sowohl Sharded- als auch Referenztabellen leeren. Die folgenden `VACUUM`-Funktionen werden in Aurora PostgreSQL Limitless Database vollständig unterstützt:

- VACUUM
- ANALYZE
- DISABLE_PAGE_SKIPPING
- FREEZE
- FULL
- INDEX_CLEANUP
- PARALLEL
- PROCESS_TOAST
- TRUNCATE
- VERBOSE

Für `VACUUM` gelten in Aurora PostgreSQL Limitless Database die folgenden Einschränkungen:

- Die Erweiterung pg_visibility_map wird nicht unterstützt.
- Die Suche nach ungenutzten Indizes über die Ansicht pg_stat_all_indexes wird nicht unterstützt.
- Konsolidierte Ansichten für pg_stat_user_indexes, pg_class und pg_stats sind nicht implementiert.

Weitere Informationen zum Befehl `VACUUM` finden Sie in der PostgreSQL-Dokumentation unter VACUUM. Weitere Informationen zum Leeren von Tabellen in Aurora PostgreSQL Limitless Database finden Sie unter Zurückgewinnen von Speicherplatz durch Bereinigung.

Variablen in Aurora PostgreSQL Limitless Database

Sie können die folgenden Variablen verwenden, um Aurora PostgreSQL Limitless Database zu konfigurieren.

rds_aurora.limitless_active_shard_key

Legt bei der Abfrage der Datenbank einen einzelnen Shard-Schlüssel fest, wodurch an alle `SELECT`- und DML-Abfragen der Shard-Schlüssel als konstantes Prädikat angehängt wird. Weitere Informationen finden Sie unter Festlegen eines aktiven Shard-Schlüssels.

rds_aurora.limitless_create_table_collocate_with

Legen Sie für diese Variable einen bestimmten Tabellennamen fest, um neu erstellte Tabellen dieser Tabelle zuzuordnen. Weitere Informationen finden Sie unter Erstellen von Limitless-Tabellen mithilfe von Variablen.

rds_aurora.limitless_create_table_mode

Legt den Modus für die Tabellenerstellung fest. Weitere Informationen finden Sie unter Erstellen von Limitless-Tabellen mithilfe von Variablen.

rds_aurora.limitless_create_table_shard_key

Legen Sie für diese Variable ein Array von Spaltennamen fest, die als Shard-Schlüssel verwendet werden sollen. Weitere Informationen finden Sie unter Erstellen von Limitless-Tabellen mithilfe von Variablen.

rds_aurora.limitless_explain_options

Legt fest, was in die Ausgabe `EXPLAIN` aufgenommen werden soll. Weitere Informationen finden Sie unter EXPLAIN.

DB-Cluster-Parameter in der Aurora PostgreSQL Limitless-Datenbank

Sie können die folgenden DB-Cluster-Parameter verwenden, um Aurora PostgreSQL Limitless Database zu konfigurieren.

rds_aurora.limitless_adaptive_fetch_size

Verbessert das Vorabrufen von Batches. Wenn dieser Parameter auf `true` gesetzt ist, ermöglicht er eine sich selbst anpassende (adaptive) Abrufgröße für den Vorabruf. Wenn der Parameter auf `false` gesetzt ist, bleibt die Abrufgröße konstant.

rds_aurora.limitless_auto_scale_options

Legt die verfügbaren Optionen für das Hinzufügen von Routern oder das Unterteilen von Shards in einer DB-Shard-Gruppe fest. Der Wert kann `add_router`, `split_shard` oder Beides lauten.

Weitere Informationen erhalten Sie unter Hinzufügen eines Routers zu einer DB-Shard-Gruppe und Unterteilen eines Shards in einer DB-Shard-Gruppe.

rds_aurora.limitless_distributed_deadlock_timeout

Legt die Zeit in Millisekunden fest, die während einer Sperre gewartet wird, bevor geprüft wird, ob ein verteilter Deadlock vorliegt. Der Standardwert ist `1000` (1 Sekunde).

Weitere Informationen finden Sie unter Verteilte Deadlocks in Aurora PostgreSQL Limitless Database.

rds_aurora.limitless_enable_auto_scale

Ermöglicht das Hinzufügen von Routern und das Unterteilen von Shards in einer DB-Shard-Gruppe.

Weitere Informationen erhalten Sie unter Hinzufügen eines Routers zu einer DB-Shard-Gruppe und Unterteilen eines Shards in einer DB-Shard-Gruppe.

rds_aurora.limitless_finalize_split_shard_mode

Bestimmt, wie vom System initiierte Shard-Splits finalisiert werden. Weitere Informationen finden Sie unter Unterteilen eines Shards in einer DB-Shard-Gruppe.

rds_aurora.limitless_maximum_adaptive_fetch_size

Legt die Obergrenze für die adaptive Abrufgröße fest. Der Bereich ist 1 bis `INT_MAX`. Der Standardwert ist `1000`.

Verwenden von Amazon Aurora Global Database

Mit der Funktion von Amazon Aurora Global Database richten Sie mehrere Aurora-DB-Cluster ein, die sich über mehrere AWS-Regionen erstrecken. Aurora synchronisiert automatisch alle im primären DB-Cluster vorgenommenen Änderungen mit einem oder mehreren sekundären Clustern. Eine globale Aurora-Datenbank hat einen primären DB-Cluster in einer Region und bis zu 10 sekundäre DB-Cluster in verschiedenen Regionen. Diese multiregionale Konfiguration ermöglicht eine schnelle Wiederherstellung nach einem seltenen Ausfall, der sich auf die gesamte AWS-Region auswirken könnte. Eine vollständige Kopie all Ihrer Daten an mehreren geografischen Standorten ermöglicht auch Lesevorgänge mit geringer Latenzzeit für Anwendungen, die von weit voneinander entfernten globalen Standorten aus eine Verbindung herstellen.

Themen

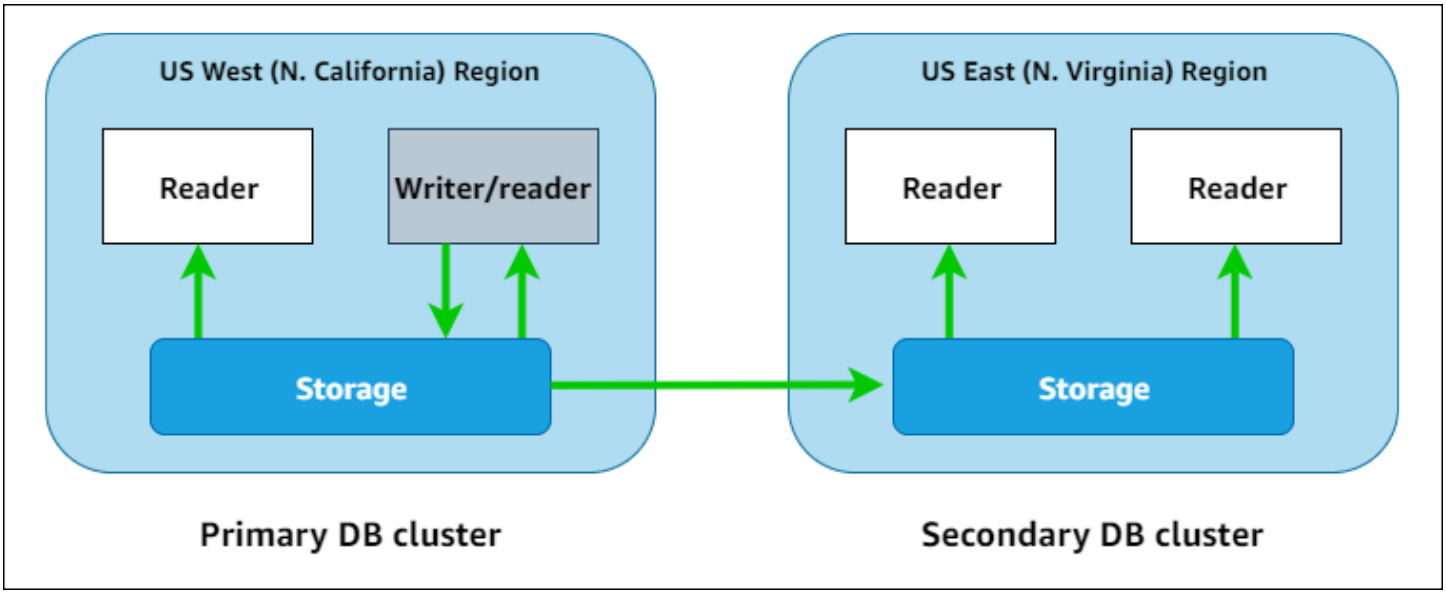
- Übersicht über Amazon Aurora Global Database
- Vorteile von Amazon Aurora Global Database
- Verfügbarkeit von Regionen und Versionen
- Einschränkungen von Amazon Aurora Global Database
- Erste Schritte mit Amazon Aurora Global Database
- Verwalten einer Amazon Aurora Global Database
- Verbinden mit Amazon Aurora Global Database
- Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank
- Verwenden von Umstellung oder Failover in Amazon Aurora Global Database
- Überwachen einer globalen Amazon-Aurora-Datenbank
- Blue/Green Bereitstellungen für Amazon Aurora Global Database verwenden
- Verwenden von globalen Amazon-Aurora-Datenbanken mit anderen AWS-Services
- Aktualisieren einer globalen Datenbank von Amazon Aurora

Übersicht über Amazon Aurora Global Database

Durch die Verwendung der Funktion von Amazon Aurora Global Database können Sie Ihre global verteilten Anwendungen mit einer einzigen Aurora-Datenbank ausführen, die sich über mehrere AWS-Regionen erstreckt.

Eine globale Aurora-Datenbank besteht aus einer primären Datenbank, AWS-Region in die Ihre Daten geschrieben werden, und bis zu 10 schreibgeschützten sekundären Datenbanken. AWS-Regionen Schreibvorgänge erfolgen direkt auf dem primären DB-Cluster in der primären AWS-Region. Am einfachsten ist es, eine Verbindung zum Writer-Endpunkt von Aurora Global Database herzustellen, der immer auf den primären DB-Cluster verweist, auch nach einer Umstellung oder einem Failover zu einer anderen AWS-Region. Nach jedem Schreibvorgang repliziert Aurora Daten AWS-Regionen mithilfe einer dedizierten Infrastruktur auf die Sekundärseite, wobei die Latenz in der Regel unter einer Sekunde liegt.

In der folgenden Abbildung finden Sie ein Beispiel für eine globale Aurora-Datenbank, die sich über zwei AWS-Regionen erstreckt.



Sie können jeden sekundären Cluster separat hochskalieren, indem Sie eine oder mehrere Aurora-Reader-Instances zum Bedienen schreibgeschützter Workloads hinzufügen. Sie können Aurora Serverless v2 für die Reader-Instances verwenden, um noch detaillierter und flexibler Skalieren zu können.

Nur der primäre Cluster führt Schreibvorgänge aus. Clients, die Schreibvorgänge ausführen, stellen eine Verbindung zum Writer-Endpunkt der globalen Aurora-Datenbank her, der immer auf die Writer-DB-Instance des primären Clusters verweist. Wie im Diagramm dargestellt, verwendet Aurora das Cluster-Speicher-Volume und nicht die Datenbank-Engine für die schnelle Replikation mit geringem Overhead. Weitere Informationen hierzu finden Sie unter Übersicht über Amazon-Aurora-Speicher.

Aurora Global Database ist für Anwendungen mit weltweiter Präsenz konzipiert. Die mehrfachen schreibgeschützten sekundären DB-Cluster AWS-Regionen helfen dabei, Lesevorgänge in der Nähe der Anwendungsbenutzer zu optimieren. Mit dem Funktion der Schreibweiterleitung können Sie Ihre

globale Aurora-Datenbank auch so konfigurieren, dass sekundäre Cluster Daten an primäre Cluster senden. Weitere Informationen finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

Aurora Global Database unterstützt je nach Szenario zwei verschiedene Operationen zum Ändern der Region Ihres primären DB-Clusters: Umstellung von Aurora Global Database und Failover von Aurora Global Database.

- Verwenden Sie für geplante operative Verfahren wie die regionale Rotation den Umstellungsmechanismus (früher als „verwaltetes geplantes Failover“ bezeichnet). Mit dieser Funktion können Sie den primären Cluster einer fehlerfreien globalen Aurora-Datenbank ohne Datenverlust in eine der sekundären Regionen verschieben. Weitere Informationen hierzu finden Sie unter Durchführen von Umstellungen für Amazon Aurora Global Databases.
- Um Ihre globale Aurora-Datenbank nach einem Ausfall in der primären Region wiederherzustellen, verwenden Sie den Failover-Mechanismus. Mit dieser Funktion führen Sie ein Failover Ihres primären DB-Clusters zu einer anderen Region durch (regionsübergreifendes Failover). Weitere Informationen hierzu finden Sie unter Ausführen von verwalteten geplanten Failovers für globale Aurora-Datenbanken.

Vorteile von Amazon Aurora Global Database

Durch die Verwendung von Aurora Global Database profitieren Sie von folgenden Vorteilen:

- Globale Lesevorgänge mit lokaler Latenz – Wenn Sie Niederlassungen auf der ganzen Welt haben, können Sie Aurora Global Database nutzen, um die wichtigsten Datenquellen in der primären AWS-Region auf dem neuesten Stand zu halten. Niederlassungen in Ihren anderen Regionen können mit lokaler Latenz auf die Daten in ihrer eigenen Region zugreifen.
- Skalierbare sekundäre Aurora-DB-Cluster – Sie können Ihre sekundären Cluster skalieren, indem Sie einer sekundären AWS-Region weitere schreibgeschützte Instances hinzufügen. Der sekundäre Cluster ist schreibgeschützt und kann daher bis zu 16 schreibgeschützte DB-Instances anstelle des üblichen Limits von 15 für einen einzelnen Aurora-Cluster unterstützen.
- Schnelle Replikation von primären zu sekundären Aurora-DB-Clustern – Die von Aurora Global Database ausgeführte Replikation führt zu geringen Leistungseinbußen auf dem primären DB-Cluster. Die Ressourcen der DB-Instances werden ausschließlich für Lese- und Schreib-Workloads von Anwendungen genutzt.

- Wiederherstellung nach regionsweiten Ausfällen – Die sekundären Cluster ermöglichen Ihnen, eine globale Aurora-Datenbank in einer neuen primären AWS-Region schneller (niedrigere RTO) und mit weniger Datenverlust (niedrigerer RPO) als herkömmliche Replikationslösungen verfügbar zu machen.

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Aurora-Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur Verfügbarkeit von Versionen und Regionen im Zusammenhang mit Aurora Global Database finden Sie unter Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken.

Einschränkungen von Amazon Aurora Global Database

Für Aurora Global Database gelten aktuell die folgenden Einschränkungen:

- Aurora Global Database ist in bestimmten AWS-Regionen und für bestimmte Versionen von Aurora MySQL und Aurora PostgreSQL verfügbar. Weitere Informationen finden Sie unter Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken.
- Aurora Global Database stellt bestimmte Konfigurationsanforderungen an unterstützte DB-Instance-Klassen von Aurora, beispielsweise eine maximale Anzahl von AWS-Regionen. Weitere Informationen finden Sie unter Anforderungen an die Konfiguration einer globalen Amazon-Aurora-Datenbank.
- Bei Aurora MySQL mit MySQL-5.7-Kompatibilität sind für Umstellungen von Aurora Global Database die Version 2.09.1 oder höhere Nebenversionen erforderlich.
- Sie können verwaltete regionsübergreifende Umstellungen oder Failover nur mit Aurora Global Database durchführen, wenn die primären und sekundären DB-Cluster dieselben Engine-Haupt- und Nebenversionen haben. Je nach Engine und Engine-Versionen müssen die Patch-Level möglicherweise identisch sein oder können unterschiedlich sein. Eine Liste der Engines und Engine-Versionen, die diese Operationen zwischen primären und sekundären Clustern mit unterschiedlichen Patch-Leveln unterstützen, finden Sie unter Patch-Level-Kompatibilität für verwaltete regionsübergreifende Umstellungen und Failover. Wenn Ihre Engine-Versionen identische Patch-Level erfordern, können Sie das Failover manuell durchführen, indem Sie die Schritte unter Ausführen von manuellen geplanten Failovers für globale Aurora-Datenbanken ausführen.
- Aurora Global Database unterstützt derzeit die folgenden Aurora-Funktionen nicht:

  - Aurora Serverless v1
  - Rückverfolgung in Aurora
- Informationen zu Einschränkungen bei der Verwendung der RDS-Proxy-Funktion mit Aurora Global Database finden Sie unter Einschränkungen für RDS Proxy mit globalen Datenbanken.
- Die automatische Aktualisierung von Nebenversionen gilt nicht für Aurora-MySQL- und Aurora-PostgreSQL-Cluster, die Teil einer globalen Datenbank sind. Beachten Sie, dass Sie diese Einstellung für eine DB-Instance angeben können, die Teil eines globalen Datenbank-Clusters ist, aber die Einstellung hat keine Auswirkungen.
- Aurora Global Database unterstützt derzeit kein Aurora-Auto-Scaling für sekundäre DB-Cluster.
- Um Database Activity Streams (DAS) auf einer globalen Aurora-Datenbank zu verwenden, auf der Aurora MySQL 5.7 ausgeführt wird, muss die Engine-Version 2.08 oder höher sein. Weitere Informationen zu DAS finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.
- Für ein Upgrade von Aurora Global Database gelten aktuell die folgenden Einschränkungen:
  - Sie können keine benutzerdefinierte Parametergruppe auf den globalen Datenbank-Cluster anwenden, während Sie ein Hauptversions-Upgrade dieser globalen Aurora-Datenbank durchführen. Sie erstellen Ihre benutzerdefinierten Parametergruppen in jeder Region des globalen Clusters und wenden sie nach dem Upgrade manuell auf die regionalen Cluster an.
  - Mit einer globalen Aurora-Datenbank, die auf Aurora MySQL basiert, können Sie kein direktes Upgrade von Aurora MySQL Version 2 auf Version 3 durchführen, wenn der `lower_case_table_names`-Parameter aktiviert ist. Weitere Informationen zu den möglichen Verfahren finden Sie unter Hauptversions-Upgrades.
  - Mit Aurora Global Database können Sie kein Upgrade der Hauptversion der DB-Engine von Aurora PostgreSQL durchführen, wenn die Recovery Point Objective (RPO)-Funktion aktiviert ist. Weitere Informationen über die RPO-Funktion finden Sie unter Verwalten von RPOs für Aurora PostgreSQL–basierte globale Datenbanken.
  - Bei einer globalen Aurora-Datenbank können Sie kein Nebenversions-Upgrade von Aurora-MySQL-Version 3.01 oder 3.02 auf 3.03 oder höher durchführen, indem Sie den Standardprozess verwenden. Weitere Informationen zu dem zu verwendenden Prozess finden Sie unter Upgrade von Aurora MySQL durch Ändern der Engine-Version.

  Informationen zum Upgrade von Aurora Global Database finden Sie unter Aktualisieren einer globalen Datenbank von Amazon Aurora.

- Sie können die Aurora-DB-Cluster in Ihrer globalen Datenbank nicht einzeln anhalten oder starten. Weitere Informationen hierzu finden Sie unter Stoppen und Starten eines Amazon Aurora-DB-Clusters.
- Reader-DB-Instances von Aurora, die mit dem sekundären Aurora-DB-Cluster verbunden sind, können unter bestimmten Umständen neu gestartet werden. Wenn die AWS-Region Writer-DB-Instance der primären Instanz neu gestartet oder ausgefallen wird, werden auch die Leser-DB-Instances in den sekundären Regionen neu gestartet. Der sekundäre Cluster ist erst dann wieder verfügbar, wenn alle Reader-DB-Instances wieder mit der Writer-Instance des primären DB-Clusters synchronisiert sind. Das Verhalten des primären Clusters beim Neustart oder beim Failover ist dasselbe wie bei einem einzelnen, nicht globalen DB-Cluster. Weitere Informationen finden Sie unter Replikation mit Amazon Aurora.

  Stellen Sie sicher, dass Sie die Auswirkungen auf Ihre globale Datenbank verstehen, bevor Sie Änderungen an Ihrem primären DB-Cluster vornehmen. Weitere Informationen hierzu finden Sie unter Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall.
- Aurora Global Database unterstützt derzeit nicht den `inaccessible-encryption-credentials-recoverable` Status, wenn Amazon Aurora den Zugriff auf den AWS KMS Schlüssel für den DB-Cluster verliert. In diesen Fällen wechselt der verschlüsselte DB-Cluster direkt in den Beendigungszustand `inaccessible-encryption-credentials`. Weitere Informationen zu diesen Zuständen finden Sie unter Anzeigen des DB-Clusterstatus.
- Secrets Manager unterstützt Aurora Global Database nicht. Wenn Sie einer globalen Datenbank eine Region hinzufügen, müssen Sie zuerst die Secrets-Manager-Integration für die DB-Instance deaktivieren.
- Für Aurora-PostgreSQL-basierte DB-Cluster, die Aurora Global Database verwenden, gelten folgende Einschränkungen:
  - Die Verwaltung des Cluster-Cache wird für sekundäre DB-Cluster von Aurora PostgreSQL, die Teil der globalen Aurora-Datenbanken sind, nicht unterstützt.
  - Wenn der primäre DB-Cluster Ihrer globalen Datenbank auf dem Replikat einer PostgreSQL-Instance von Amazon RDS basiert, können Sie keinen sekundären Cluster erstellen. Versuchen Sie nicht, mithilfe der API-Operation, der oder der AWS-ManagementkonsoleCreateDBCluster API-Operation eine sekundäre AWS CLI Komponente aus diesem Cluster zu erstellen. Solche Versuche werden unterbrochen und der sekundäre Cluster wird nicht erstellt.

Wir empfehlen, dass Sie sekundäre DB-Cluster für Ihre globalen Datenbanken erstellen, indem Sie die Version der Aurora-DB-Engine verwenden, die auch für den primären Cluster verwendet wird. Weitere Informationen finden Sie unter Erstellen einer globalen Datenbank von Amazon Aurora.

Erste Schritte mit Amazon Aurora Global Database

Um mit Aurora Global Database zu beginnen, müssen Sie zunächst entscheiden, welche Aurora-DB-Engine Sie verwenden möchten und in welchen AWS-Regionen. Nur bestimmte Versionen der Datenbank-Engines von Aurora MySQL und Aurora PostgreSQL in bestimmten AWS-Regionen unterstützen Aurora Global Database. Die vollständige Liste finden Sie unter Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken.

Sie können eine globale Aurora-Datenbank auf eine der folgenden Arten erstellen:

- Erstellen einer neuen globalen Aurora-Datenbank mit neuen Aurora-DB-Clustern und Aurora-DB-Instances – Sie können dies tun, indem Sie die unter Erstellen einer globalen Datenbank von Amazon Aurora beschriebenen Schritte ausführen. Nachdem Sie den primären Aurora-DB-Cluster erstellt haben, fügen Sie die sekundäre AWS-Region hinzu, indem Sie die unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank beschriebenen Schritte ausführen.
- Verwenden eines vorhandenen Aurora-DB-Clusters, der die Funktion von Aurora Global Database unterstützt und diesem eine AWS-Region hinzufügen – Dies können Sie nur tun, wenn Ihr vorhandener Aurora-DB-Cluster eine DB-Engine-Version verwendet, die global kompatibel ist.

  Überprüfen Sie, ob Sie in Region hinzufügen als Aktion auf AWS-Managementkonsole auswählen können, wenn Ihr Aurora-DB-Cluster ausgewählt ist. Wenn das möglich ist, können Sie diesen Aurora-DB-Cluster als globalen Aurora-Cluster verwenden. Weitere Informationen finden Sie unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.

Bevor Sie eine globale Aurora-Datenbank erstellen, sollten Sie sich mit allen Konfigurationsanforderungen vertraut machen.

Themen

- Anforderungen an die Konfiguration einer globalen Amazon-Aurora-Datenbank
- Erstellen einer globalen Datenbank von Amazon Aurora
- Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank
- Erstellen eines Aurora-Headless-DB-Clusters in einer sekundären Region

- Erstellen einer globalen Amazon-Aurora-Datenbank aus einem Aurora- oder Amazon-RDS-Snapshot

Anforderungen an die Konfiguration einer globalen Amazon-Aurora-Datenbank

Bevor Sie anfangen, mit Ihrer globalen Datenbank zu arbeiten, sollten Sie die Cluster- und Instance-Namen, AWS-Regionen sowie die Anzahl der Instances und Instance-Klassen planen, die Sie verwenden möchten. Stellen Sie sicher, dass Ihre Auswahl den folgenden Konfigurationsanforderungen entspricht:

Eine globale Aurora-Datenbank umfasst mindestens zwei AWS-Regionen. Die primäre AWS-Region unterstützt einen Aurora-DB-Cluster mit einer Writer-Aurora-DB-Instance. Eine sekundäre AWS-Region führt einen schreibgeschützten Aurora-DB-Cluster aus, der vollständig aus Aurora-Replikaten besteht. Es ist mindestens eine sekundäre AWS-Region erforderlich, wobei eine globale Aurora-Datenbank bis zu 10 sekundäre AWS-Regionen haben kann. In der Tabelle sind die maximal zulässigen Aurora-DB-Cluster, Aurora-DB-Instances und Aurora-Replicas aufgeführt, die in einer globalen Aurora-Datenbank zulässig sind.

Beschreibung	Primär AWS-Region	Sekundär AWS-Regionen
Aurora-DB-Cluster	1	10 (maximal)
Writer-Inst	1	0
Schreibgeschützte Instances (Aurora-Replicas) pro Aurora-DB-Cluster	15 (Max.)	16 (Total)
Schreibgeschützte Instances (maximal zulässig, bei tatsächli cher Anzahl von sekundären Regionen)	15 - s	s = Gesamtzahl der sekundären AWS-Regionen

Die Aurora-DB-Cluster, aus denen eine globale Aurora-Datenbank besteht, haben die folgenden spezifischen Anforderungen:

- DB-Instance-Klassenanforderungen – Eine globale Aurora-Datenbank erfordert DB-Instance-Klassen, die für speicherintensive Anwendungen optimiert sind. Weitere Informationen zu den arbeitsspeicheroptimierten DB-Instance-Klassen finden Sie unter DB-Instance-Klassenarten. Wir empfehlen, db.r5 oder eine höhere Instance-Klasse zu nutzen.
- AWS-Region-Anforderungen – Eine globale Aurora-Datenbank benötigt einen primären Aurora-DB-Cluster in einer AWS-Region und mindestens einen sekundären Aurora-DB-Cluster in einer anderen Region. Sie können bis zu 10 sekundäre (schreibgeschützte) Aurora-DB-Cluster erstellen und jeder muss sich in einer anderen Region befinden. Mit anderen Worten, es können sich keine zwei Aurora-DB-Cluster in einer globalen Aurora-Datenbank in derselben AWS-Region befinden.

  Informationen darüber, welche Kombinationen aus Aurora-DB-Engine, Engine-Version und AWS-Region Sie bei Aurora Global Database verwenden können, finden Sie unter Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken.
- Benennungsanforderungen – Die Namen, die Sie für jeden Ihrer Aurora-DB-Cluster auswählen, müssen in allen AWS-Regionen eindeutig sein. Sie können nicht denselben Namen für verschiedene Aurora-DB-Cluster verwenden, obwohl sie sich in verschiedenen Regionen befinden.
- Kapazitätsanforderungen für Aurora Serverless v2 – Für eine globale Datenbank mit Aurora Serverless v2 beträgt die erforderliche Mindestkapazität für den DB-Cluster in der primären AWS-Region 8 ACUs.

Bevor Sie die in diesem Abschnitt beschriebenen Verfahren befolgen können, benötigen Sie ein AWS-Konto. Schließen Sie die Einrichtungsaufgaben für die Arbeit mit Amazon Aurora ab. Weitere Informationen finden Sie unter Einrichten Ihrer Umgebung für Amazon Aurora. Sie müssen auch andere vorab erforderliche Schritte zum Erstellen eines Aurora-DB-Clusters ausführen. Weitere Informationen hierzu finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Wenn Sie bereit sind, Ihre globale Datenbank einzurichten, finden Sie unter Erstellen einer globalen Datenbank von Amazon Aurora Informationen über das Verfahren zum Erstellen aller erforderlichen Ressourcen. Sie können auch das Verfahren unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank befolgen, um eine globale Datenbank mit einem vorhandenen Aurora-Cluster als primärem Cluster zu erstellen.

Erstellen einer globalen Datenbank von Amazon Aurora

Führen Sie die folgenden Schritte aus, um eine globale Aurora-Datenbank sowie die zugehörigen Ressourcen mit der AWS-Managementkonsole, der AWS CLI oder der RDS-API zu erstellen.

> **Note**
>
> Wenn Sie über einen vorhandenen Aurora-DB-Cluster verfügen, auf dem eine global kompatible Aurora-Datenbank-Engine ausgeführt wird, können Sie das Verfahren abkürzen. In diesem Fall können Sie eine weitere AWS-Region zum vorhandenen DB-Cluster hinzufügen, um Ihre globale Aurora-Datenbank zu erstellen. Lesen Sie dazu den Abschnitt Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.

Konsole

Die Schritte zum Erstellen einer globalen Aurora-Datenbank beginnen mit der Anmeldung bei einer AWS-Region, die die globale Aurora-Datenbankfunktion unterstützt. Eine vollständige Liste finden Sie hier: Unterstützte Regionen und DB-Engines für globale Aurora-Datenbanken.

Einer der folgenden Schritte ist die Auswahl einer Virtual Private Cloud (VPC), die auf Amazon VPC für Ihren Aurora-DB-Cluster basiert. Um Ihre eigene VPC zu verwenden, empfehlen wir Ihnen, diese im Voraus zu erstellen, damit sie von Ihnen ausgewählt werden kann. Erstellen Sie gleichzeitig alle zugehörigen Subnetze und nach Bedarf eine Subnetz- und Sicherheitsgruppe. Um zu erfahren wie dies geht, vgl. Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).

Allgemeine Informationen zum Erstellen eines Aurora-DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

So erstellen Sie eine globale Aurora-Datenbank

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Datenbank erstellen aus. Gehen Sie auf der Seite Datenbank erstellen wie folgt vor:
   - Wählen Sie als Datenbankerstellungsmethode Standarderstellung aus. Wählen Sie nicht Einfache Erstellung.
   - Wählen Sie für `Engine type` im Abschnitt Engine-Optionen den entsprechenden Engine-Typ, Aurora (MySQL-kompatibel) oder Aurora (PostgreSQL-kompatibel).
3. Fahren Sie fort mit dem Erstellen der globalen Datenbank, indem Sie den Schritten aus den folgenden Verfahren folgen:

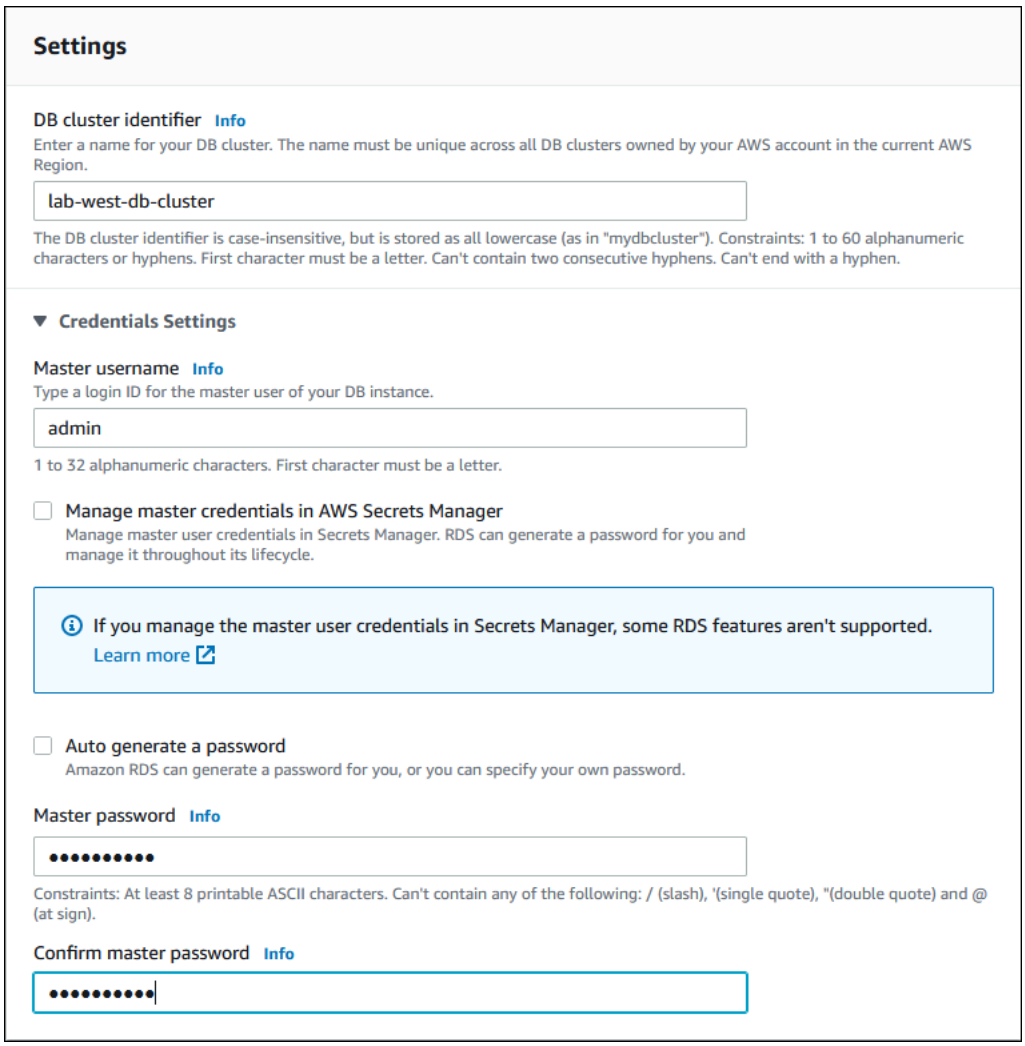
Erstellen einer globalen Datenbank mit Aurora MySQL

Die folgenden Schritte gelten für alle Versionen von Aurora MySQL.

So erstellen Sie eine globale Aurora-Datenbank mit Aurora MySQL:

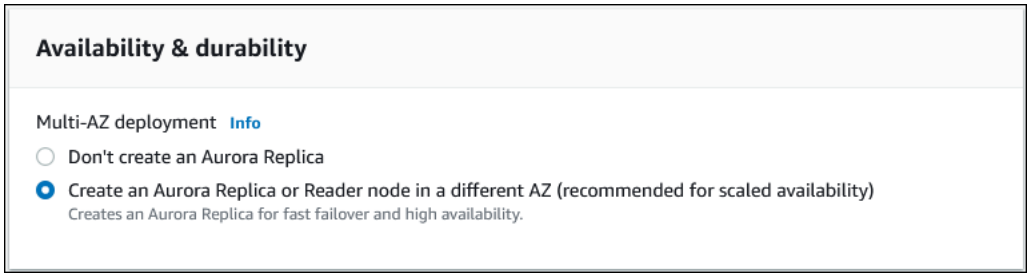
Füllen Sie die Seite Datenbank erstellen aus.

1. Wählen Sie unter Engine-Optionen Folgendes aus:
   - Wählen Sie für Engine-Version die Version von Aurora MySQL, die Sie für Ihre globale Aurora-Datenbank verwenden möchten.
2. Wählen Sie für VorlagenProduktion. Oder Sie können Dev/Test wählen, falls dies für Ihren Anwendungsfall geeignet ist. Verwenden Sie Dev/Test nicht in Produktionsumgebungen.
3. Für Settings (Einstellungen) nehmen Sie folgendes vor:
   a. Geben Sie einen aussagekräftigen Namen für die DB-Cluster-ID ein. Wenn Sie mit dem Erstellen der Aurora globalen Datenbank fertig sind, identifiziert dieser Name den primären DB-Cluster.
   b. Geben Sie Ihr eigenes Passwort für das `admin`-Benutzerkonto für die DB-Instance ein oder lassen Sie Aurora eines für Sie erstellen. Wenn Sie ein Passwort automatisch generieren, erhalten Sie die Option zum Kopieren des Passworts.



4. Wählen Sie für DB instance class (DB-Instance-Klasse) `db.r5.large` oder eine andere arbeitsspeicheroptimierte DB-Instance-Klasse aus. Wir empfehlen, db.r5 oder eine höhere Instance-Klasse zu nutzen.

5. Für Verfügbarkeit und Haltbarkeit empfehlen wir Ihnen, Aurora für die Erstellung einer Aurora-Replica in einer anderen Availability Zone (AZ) zu wählen. Wenn Sie jetzt keine Aurora-Replica erstellen, müssen Sie dies später tun.



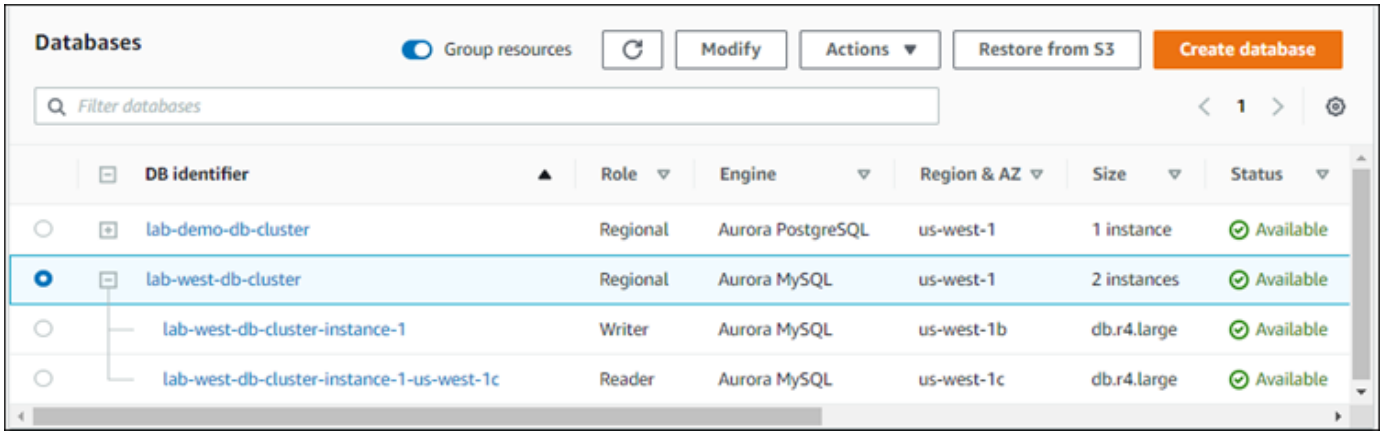
6. Wählen Sie bei Anbindung die auf Amazon VPC basierende Virtual Private Cloud (VPC) aus, die die virtuelle Netzwerkumgebung für diese DB-Instance definiert. Sie können die Standardwerte auswählen, um diese Aufgabe zu vereinfachen.
7. Schließen Sie die Einstellungen für die Datenbankauthentifizierung ab. Um den Prozess zu vereinfachen, können Sie jetzt Passwort-Authentifizierung wählen und AWS Identity and Access Management (IAM) später einrichten.
8. Für Zusätzliche Konfiguration führen Sie die folgenden Schritte aus:

   a. Geben Sie einen Namen für Anfänglicher Datenbankname ein, um die primäre Aurora-DB-Instance für diesen Cluster zu erstellen. Dies ist der Writer-Knoten für den Aurora primären DB-Cluster.

      Belassen Sie die für die DB-Clusterparametergruppe und die DB-Parametergruppe ausgewählten Standardwerte, es sei denn, Sie haben Ihre eigenen benutzerdefinierten Parametergruppen, die Sie verwenden möchten.

   b. Deaktivieren Sie das Kontrollkästchen Rückverfolgung aktivieren, wenn es aktiviert ist. Globale Datenbanken von Aurora unterstützen keine Rückverfolgung. Sie können andernfalls die anderen Standardeinstellungen für die Zusätzliche Konfiguration akzeptieren.
9. Wählen Sie Create database (Datenbank erstellen) aus.

Es kann einige Minuten dauern, bis Aurora den Prozess zum Erstellen der Aurora-DB-Instance, ihrer Aurora Replica und des Aurora-DB-Clusters abgeschlossen hat. Sie können anhand des Status feststellen, wann der Aurora-DB-Cluster als primärer DB-Cluster in einer globalen Aurora-Datenbank verwendet werden kann. Wenn dies der Fall ist, lautet der Status und der des Writer- und Replikat-Knotens Verfügbar, wie nachstehend gezeigt.



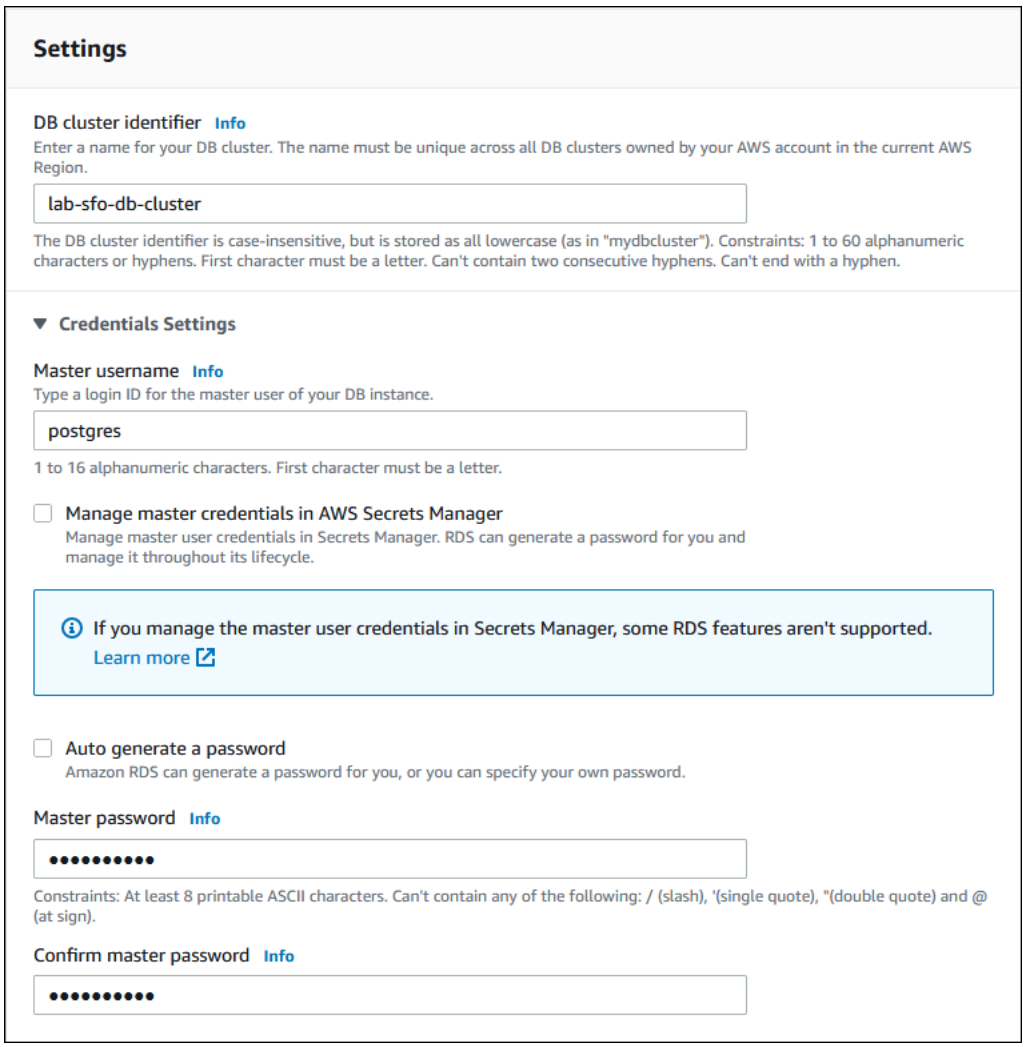
Wenn Ihr primärer DB-Cluster verfügbar ist, erstellen Sie die globale Aurora-Datenbank, indem Sie einen sekundären Cluster hinzufügen. Befolgen Sie dafür die unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank beschriebenen Schritte.

Erstellen einer globalen Datenbank mit Aurora PostgreSQL

So erstellen Sie eine globale Aurora-Datenbank mit Aurora PostgreSQL

Füllen Sie die Seite Datenbank erstellen aus.

1. Wählen Sie unter Engine-Optionen Folgendes aus:
   - Wählen Sie für Engine-Version die Version von Aurora PostgreSQL, die Sie für Ihre Aurora globale Datenbank verwenden möchten.
2. Wählen Sie für VorlagenProduktion. Oder Sie können Dev/Test wählen, falls dies geeignet ist. Verwenden Sie Dev/Test nicht in Produktionsumgebungen.
3. Für Settings (Einstellungen) nehmen Sie folgendes vor:
   a. Geben Sie einen aussagekräftigen Namen für die DB-Cluster-ID ein. Wenn Sie mit dem Erstellen der Aurora globalen Datenbank fertig sind, identifiziert dieser Name den primären DB-Cluster.
   b. Geben Sie Ihr eigenes Passwort für das Standard-Admin-Konto für den DB-Cluster ein, oder lassen Sie es Aurora für Sie generieren. Wenn Sie „Passwort automatisch generieren“ wählen, erhalten Sie die Option zum Kopieren des Passworts.



4. Wählen Sie für DB instance class (DB-Instance-Klasse) `db.r5.large` oder eine andere arbeitsspeicheroptimierte DB-Instance-Klasse aus. Wir empfehlen, db.r5 oder eine höhere Instance-Klasse zu nutzen.
5. Für Verfügbarkeit und Haltbarkeit empfehlen wir Ihnen, Aurora für die Erstellung einer Aurora-Replica in einer anderen AZ für Sie zu wählen. Wenn Sie jetzt keine Aurora-Replica erstellen, müssen Sie dies später tun.
6. Wählen Sie bei Anbindung die auf Amazon VPC basierende Virtual Private Cloud (VPC) aus, die die virtuelle Netzwerkumgebung für diese DB-Instance definiert. Sie können die Standardwerte auswählen, um diese Aufgabe zu vereinfachen.
7. (Optional) Schließen Sie die Einstellungen für die Datenbankauthentifizierung ab. Die Passwortauthentifizierung ist immer aktiviert. Um den Prozess zu vereinfachen, können Sie diesen Abschnitt überspringen und später die IAM- oder Passwort- und Kerberos-Authentifizierung einrichten.

8. Für Zusätzliche Konfigurationen führen Sie die folgenden Schritte aus:

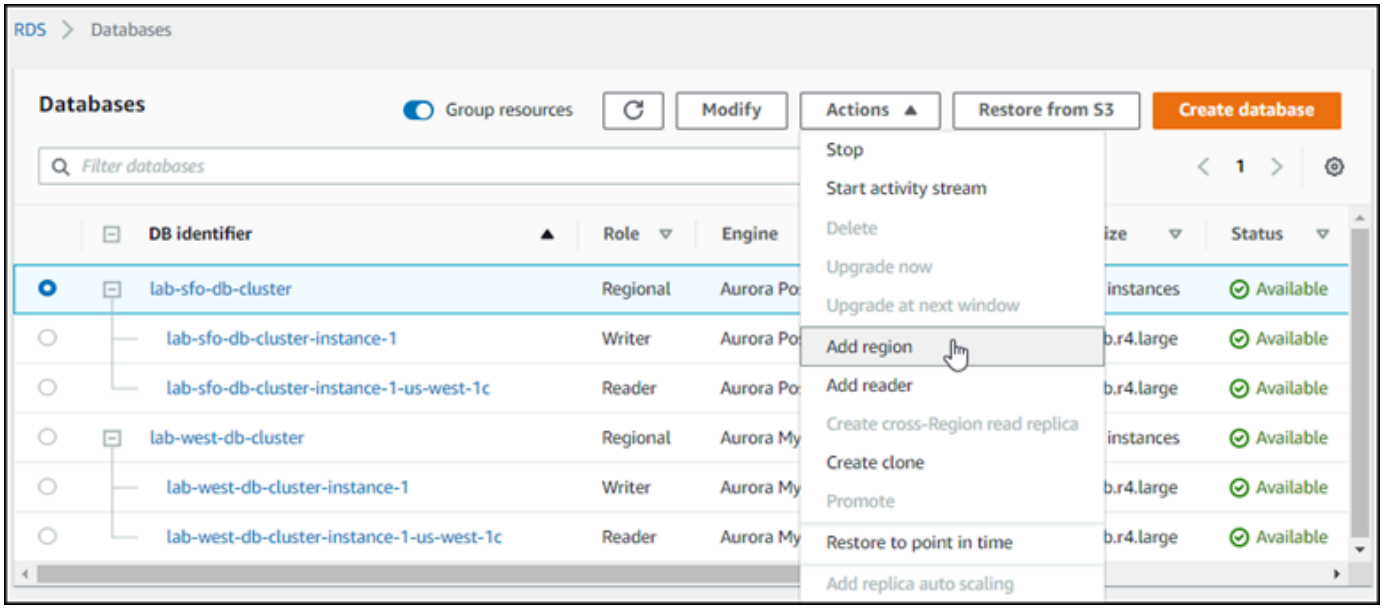
   a. Geben Sie einen Namen für Anfänglicher Datenbankname ein, um die primäre Aurora-DB-Instance für diesen Cluster zu erstellen. Dies ist der Writer-Knoten für den Aurora primären DB-Cluster.

      Belassen Sie die für die DB-Clusterparametergruppe und die DB-Parametergruppe ausgewählten Standardwerte, es sei denn, Sie haben Ihre eigenen benutzerdefinierten Parametergruppen, die Sie verwenden möchten.

   b. Akzeptieren Sie alle anderen Standardeinstellungen für zusätzliche Konfigurationen wie Verschlüsselung, Protokollexporte usw.

9. Wählen Sie Datenbank erstellen aus.

   Es kann einige Minuten dauern, bis Aurora den Prozess zum Erstellen der Aurora-DB-Instance, ihrer Aurora Replica und des Aurora-DB-Clusters abgeschlossen hat. Wenn der Cluster einsatzbereit ist, zeigen der Aurora-DB-Cluster und seine Writer- und Replica-Nodes den Status Verfügbar an. Dies wird der primäre DB-Cluster Ihrer Aurora globalen Datenbank, nachdem Sie einen sekundären Cluster hinzugefügt haben.



Wenn Ihr primäre DB-Cluster verfügbar ist, erstellen Sie einen oder mehrere sekundäre Cluster, indem Sie die Schritte Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.

AWS CLI

Die AWS CLI-Befehle in den folgenden Prozeduren führen die folgenden Aufgaben aus:

1. Erstellen einer globalen Aurora-Datenbank, Eingabe eines Namens und Spezifizierung des Typs der Aurora-Datenbank-Engine, den Sie verwenden möchten.
2. Erstellen Sie einen Aurora-DB-Cluster für die Aurora globale Datenbank.
3. Erstellen Sie eine Aurora-DB-Instance für den Cluster. Dies ist der primäre Aurora-DB-Cluster der globalen Datenbank.
4. Erstellen Sie eine zweite DB-Instance für den Aurora-DB-Cluster. Dies ist ein Reader zur Vervollständigung des Aurora-DB-Clusters.
5. Erstellen Sie einen zweiten Aurora-DB-Cluster in einer anderen Region und fügen Sie ihn dann Ihrer Aurora globalen Datenbank hinzu, indem Sie die unter beschriebenen Schritte ausführe Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.

Folgen Sie der Vorgehensweise für Ihre Aurora-Datenbank-Engine.

Erstellen einer globalen Datenbank mit Aurora MySQL

So erstellen Sie eine globale Aurora-Datenbank mit Aurora MySQL:

1. Verwenden Sie den `create-global-cluster`-CLI-Befehl zur Übergabe des Namens der AWS-Region, der Aurora-Datenbank-Engine und der Version.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
       --engine aurora-mysql \
       --engine-version version # optional
   ```

   Für Windows:

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-mysql ^
       --engine-version version # optional
   ```

   Dadurch wird eine „leere“ Aurora globale Datenbank mit nur einem Namen (Identifier) und einer Aurora-Datenbank-Engine erstellt. Es kann einige Minuten dauern, bis die Aurora globale Datenbank verfügbar ist. Bevor Sie mit dem nächsten Schritt fortfahren, prüfen Sie mit dem `describe-global-clusters`-CLI-Befehl, ob sie verfügbar ist.

```
aws rds describe-global-clusters --region primary_region --global-cluster-
identifier global_database_id
```

Wenn die Aurora globale Datenbank verfügbar ist, können Sie ihren primären Aurora-DB-Cluster erstellen.

2. Um einen primären Aurora-DB-Cluster zu erstellen, verwenden Sie den `create-db-cluster`-CLI-Befehl. Fügen Sie den Namen Ihrer Aurora globalen Datenbank mit dem Parameter `--global-cluster-identifier` ein.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-cluster \
     --region primary_region \
     --db-cluster-identifier primary_db_cluster_id \
     --master-username userid \
     --master-user-password password \
     --engine aurora-mysql \
     --engine-version version \
     --global-cluster-identifier global_database_id
   ```

   Für Windows:

   ```
   aws rds create-db-cluster ^
     --region primary_region ^
     --db-cluster-identifier primary_db_cluster_id ^
     --master-username userid ^
     --master-user-password password ^
     --engine aurora-mysql ^
     --engine-version version ^
     --global-cluster-identifier global_database_id
   ```

   Verwenden Sie den `describe-db-clusters` AWS CLI-Befehl, um zu bestätigen, dass der Aurora-DB-Cluster bereit ist. Um einen bestimmten Aurora-DB-Cluster herauszugreifen, verwenden Sie den Parameter `--db-cluster-identifier`. Oder Sie können den Aurora-DB-Cluster-Namen im Befehl weg lassen, um Details zu all Ihren Aurora-DB-Clustern in der angegebenen Region zu erhalten.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

Wenn die Antwort `"Status": "available"` für den Cluster anzeigt, ist er einsatzbereit.

3. Erstellen Sie die DB-Instance für Ihren primären Aurora-DB-Cluster. Verwenden Sie dazu den `create-db-instance`-CLI-Befehl. Geben Sie den Namen Ihres Aurora-DB-Clusters an und geben Sie die Konfigurationsdetails für die Instance an. Sie müssen die Parameter `--master-username` und `--master-user-password` im Befehl nicht übergeben, da diese vom Aurora-DB-Cluster abgerufen werden.

   Für `--db-instance-class` können Sie nur arbeitsspeicheroptimierte Klassen verwenden, z. B. `db.r5.large`. Wir empfehlen, db.r5 oder eine höhere Instance-Klasse zu nutzen. Informationen zu diesen Klassen finden Sie unter DB-Instance-Klassenarten.

   Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-mysql \
  --engine-version version \
  --region primary_region
```

   Für Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-mysql ^
  --engine-version version ^
  --region primary_region
```

   Die Operation `create-db-instance` kann einige Zeit in Anspruch nehmen. Überprüfen Sie den Status, um festzustellen, ob die Aurora-DB-Instance verfügbar ist, bevor Sie fortfahren.

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

Wenn der Befehl den Status `available` zurückgibt, können Sie eine weitere Aurora-DB-Instance für Ihren primären DB-Cluster erstellen. Dies ist die Reader-Instance (Aurora-Replica) für den Aurora-DB-Cluster.

4. Um eine weitere Aurora-DB-Instance für den Cluster zu erstellen, verwenden Sie den `create-db-instance`-CLI-Befehl.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-instance \
     --db-cluster-identifier primary_db_cluster_id \
     --db-instance-class instance_class \
     --db-instance-identifier replica_db_instance_id \
     --engine aurora-mysql
   ```

   Für Windows:

   ```
   aws rds create-db-instance ^
     --db-cluster-identifier primary_db_cluster_id ^
     --db-instance-class instance_class ^
     --db-instance-identifier replica_db_instance_id ^
     --engine aurora-mysql
   ```

Wenn die DB-Instance verfügbar ist, beginnt die Replikation vom Writer-Knoten zum Replica. Bevor Sie fortfahren, überprüfen Sie, ob die DB-Instance mit dem `describe-db-instances`-CLI-Befehl verfügbar ist.

Zu diesem Zeitpunkt verfügen Sie über eine Aurora globale Datenbank mit ihrem primären Aurora-DB-Cluster, der eine Writer-DB-Instance und eine Aurora-Replica enthält. Sie können jetzt einen schreibgeschützten Aurora-DB-Cluster in einer anderen Region hinzufügen, um Ihre Aurora globale Datenbank zu vervollständigen. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.

Erstellen einer globalen Datenbank mit Aurora PostgreSQL

Wenn Sie mithilfe der folgenden Befehle Aurora-Objekte für eine globale Aurora-Datenbank erstellen, kann es einige Minuten dauern, bis jedes verfügbar ist. Wir empfehlen, dass Sie nach Abschluss eines bestimmten Befehls den Status des jeweiligen Aurora-Objekts überprüfen, um sicherzustellen, dass der Status „Verfügbar“ lautet.

Verwenden Sie dazu den `describe-global-clusters`-CLI-Befehl.

```
aws rds describe-global-clusters --region primary_region
    --global-cluster-identifier global_database_id
```

So erstellen Sie eine globale Aurora-Datenbank mit Aurora PostgreSQL

1. Verwenden Sie den `create-global-cluster`-CLI-Befehl.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-global-cluster --region primary_region \
       --global-cluster-identifier global_database_id \
       --engine aurora-postgresql \
       --engine-version version # optional
   ```

   Für Windows:

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-postgresql ^
       --engine-version version # optional
   ```

   Wenn die Aurora globale Datenbank verfügbar ist, können Sie ihren primären Aurora-DB-Cluster erstellen.

2. Um einen primären Aurora-DB-Cluster zu erstellen, verwenden Sie den `create-db-cluster`-CLI-Befehl. Fügen Sie den Namen Ihrer Aurora globalen Datenbank mit dem Parameter `--global-cluster-identifier` ein.

   Für Linux, macOS oder Unix:

   ```
   aws rds create-db-cluster \
     --region primary_region \
     --db-cluster-identifier primary_db_cluster_id \
     --master-username userid \
     --master-user-password password \
     --engine aurora-postgresql \
     --engine-version version \
     --global-cluster-identifier global_database_id
   ```

Für Windows:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

Prüfen Sie, ob der Aurora-DB-Cluster bereit ist. Wenn die Antwort des folgenden Befehls `"Status": "available"` für den Aurora-DB-Cluster angezeigt wird, können Sie fortfahren.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

3. Erstellen Sie die DB-Instance für Ihren primären Aurora-DB-Cluster. Verwenden Sie dazu den `create-db-instance`-CLI-Befehl.

   Übergeben Sie den Namen Ihres Aurora-DB-Clusters mit dem Parameter `--db-cluster-identifier`.

   Sie müssen die Parameter `--master-username` und `--master-user-password` im Befehl nicht übergeben, da diese vom Aurora-DB-Cluster abgerufen werden.

   Für `--db-instance-class` können Sie nur arbeitsspeicheroptimierte Klassen verwenden, z. B. `db.r5.large`. Wir empfehlen, db.r5 oder eine höhere Instance-Klasse zu nutzen. Informationen zu diesen Klassen finden Sie unter DB-Instance-Klassenarten.

   Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-postgresql \
  --engine-version version \
  --region primary_region
```

Für Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --region primary_region
```

4. Prüfen Sie den Status der Aurora-DB-Instance, bevor Sie fortfahren.

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

Wenn die Antwort zeigt, dass der Status der Aurora-DB-Instance `available` lautet, können Sie eine weitere Aurora-DB-Instance für Ihren primären DB-Cluster erstellen.

5. Um eine Aurora-Replica für Aurora-DB-Cluster zu erstellen, verwenden Sie den `create-db-instance`CLI-Befehl.

Für Linux, macOS oder Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-postgresql
```

Für Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-postgresql
```

Wenn die DB-Instance verfügbar ist, beginnt die Replikation vom Writer-Knoten zum Replica. Bevor Sie fortfahren, überprüfen Sie, ob die DB-Instance mit dem `describe-db-instances`-CLI-Befehl verfügbar ist.

Ihre Aurora globale Datenbank existiert nun, verfügt aber nur in ihrer primären Region einen Aurora-DB-Cluster, der aus einer Writer-DB-Instance und einer Aurora-Replica besteht. Sie können jetzt einen schreibgeschützten Aurora-DB-Cluster in einer anderen Region hinzufügen, um Ihre Aurora globale Datenbank zu vervollständigen. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.

RDS-API

Zum Erstellen einer globalen Aurora-Datenbank mit der RDS-API führen Sie die Aktion CreateGlobalCluster aus.

Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank

Sie können das folgende Verfahren verwenden, um einer vorhandenen globalen Datenbank einen zusätzlichen sekundären Cluster hinzuzufügen. Sie können auch eine globale Datenbank aus einem eigenständigen Aurora-DB-Cluster erstellen, indem Sie mit diesem Verfahren die erste sekundäre AWS-Region hinzufügen.

Eine globale Aurora-Datenbank benötigt mindestens einen sekundären Aurora-DB-Cluster in einer anderen AWS-Region als der primäre Aurora-DB-Cluster. Sie können bis zu 10 sekundäre DB-Cluster an Ihre globale Aurora-Datenbank anhängen. Wiederholen Sie das folgende Verfahren für jeden neuen sekundären DB-Cluster. Reduzieren Sie für jeden sekundären DB-Cluster, den Sie Ihrer globalen Aurora-Datenbank hinzufügen, die Anzahl der Aurora-Replicas, die für den primären DB-Cluster zulässig sind, um eins.

Wenn Ihre globale Aurora-Datenbank beispielsweise 10 sekundäre Regionen hat, kann Ihr primärer DB-Cluster nur 5 (statt 15) Aurora-Replikate haben. Weitere Informationen finden Sie unter Anforderungen an die Konfiguration einer globalen Amazon-Aurora-Datenbank.

Die Anzahl der Aurora-Replikate (Reader-Instances) im primären DB-Cluster bestimmt die Anzahl der sekundären DB-Cluster, die Sie hinzufügen können. Die Gesamtzahl der Reader-Instances im primären DB-Cluster plus die Anzahl der sekundären Cluster darf nicht mehr als 15 betragen. Wenn Sie zum Beispiel 14 Reader-Instances im primären DB-Cluster und einen sekundären Cluster haben, können Sie der globalen Datenbank keinen weiteren sekundären Cluster hinzufügen.

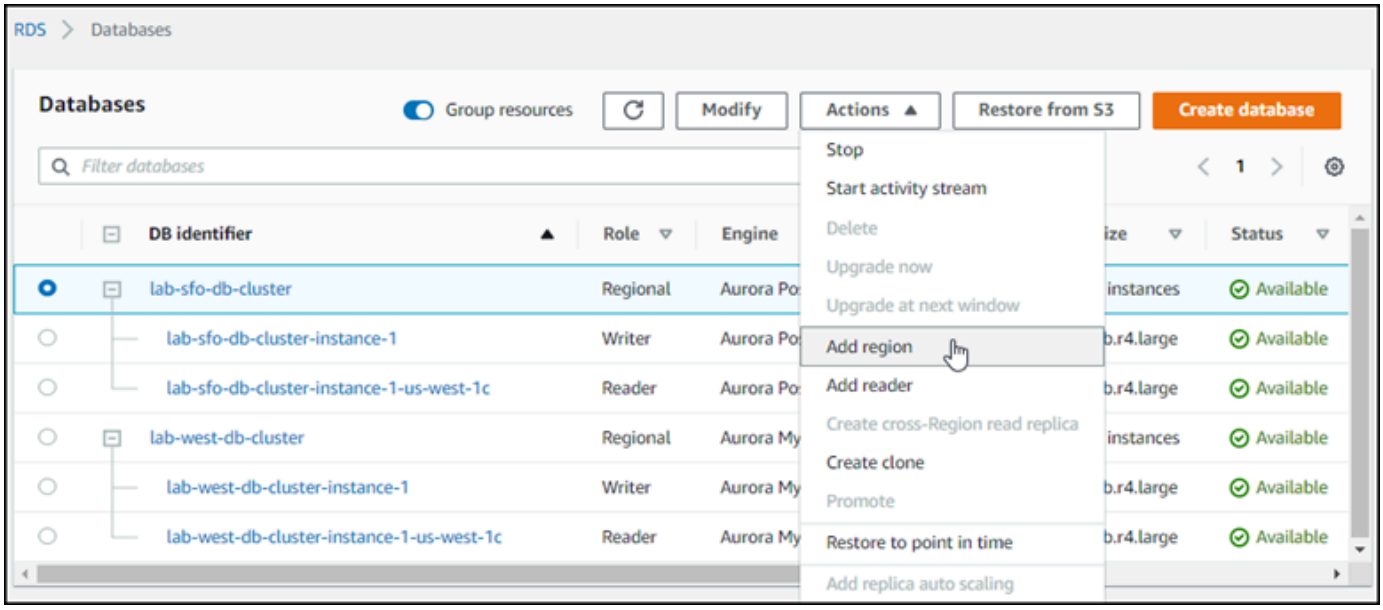
> **Note**
>
> Wenn Sie für Aurora MySQL Version 3 einen sekundären Cluster erstellen, stellen Sie sicher, dass der Wert von `lower_case_table_names` mit dem Wert im primären Cluster

übereinstimmt. Diese Einstellung ist ein Datenbankparameter, der beeinflusst, wie der Server die Groß- und Kleinschreibung von Bezeichnern behandelt. Weitere Informationen zu Datenbankparametern finden Sie unter Parametergruppen für Amazon Aurora.
Es wird empfohlen, beim Erstellen eines sekundären Clusters dieselbe DB-Engine-Version für den primären und den sekundären Cluster zu verwenden. Aktualisieren Sie bei Bedarf den primären Cluster auf die Version des sekundären Clusters. Weitere Informationen finden Sie unter Patch-Level-Kompatibilität für verwaltete regionsübergreifende Umstellungen und Failover.

Konsole

Um ein AWS-Region zu einer globalen Aurora-Datenbank hinzuzufügen

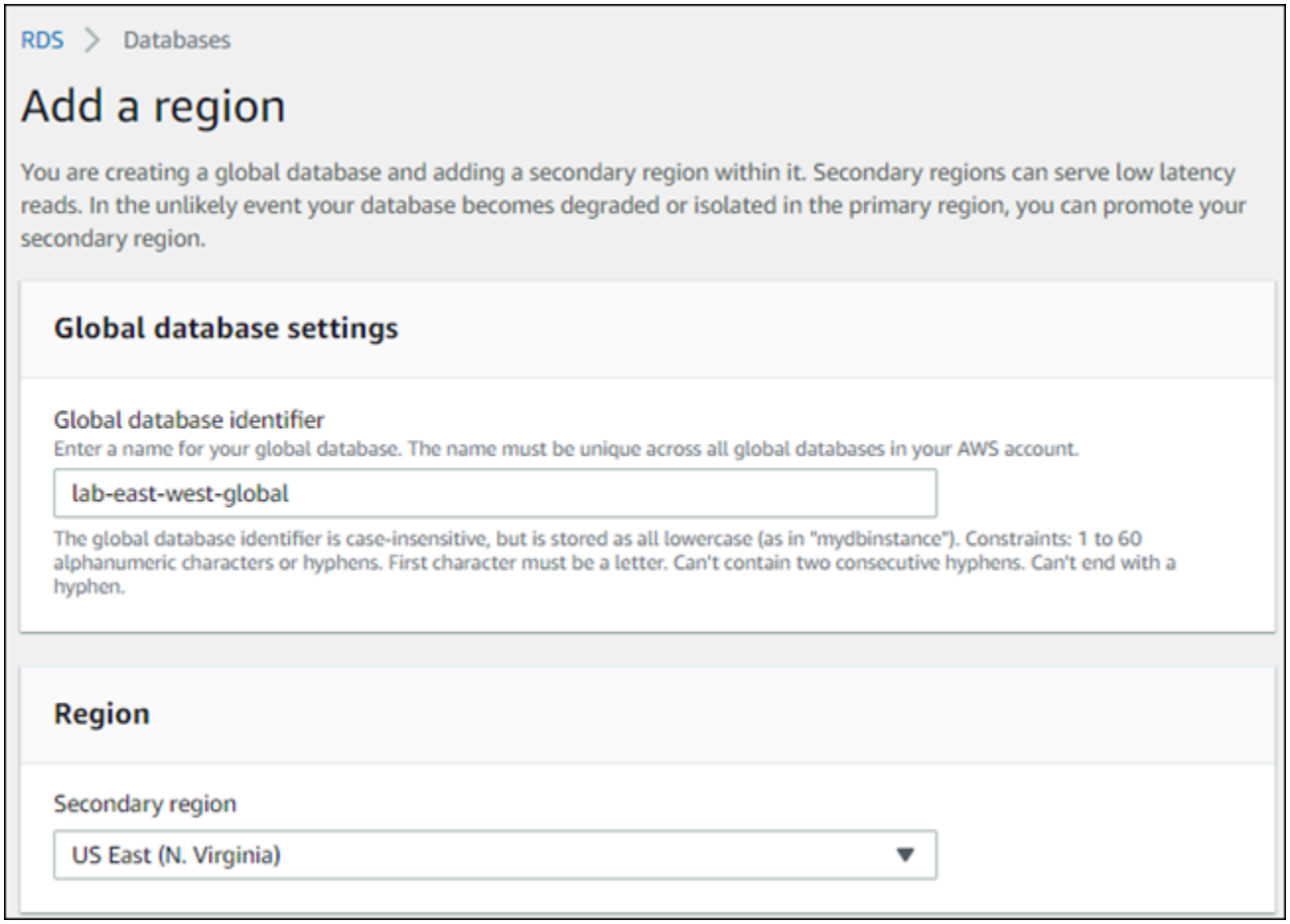
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich der AWS-Managementkonsole die Option Datenbanken aus.
3. Wählen Sie die Aurora globale Datenbank aus, die einen sekundären Aurora-DB-Cluster benötigt. Stellen Sie sicher, dass der primäre Aurora-DB-Cluster is `Available`.
4. Wählen Sie für Aktionen die Option AWS-Region hinzufügen aus.



5. Wählen Sie auf der Seite Eine Region hinzufügen die sekundäre AWS-Region aus.

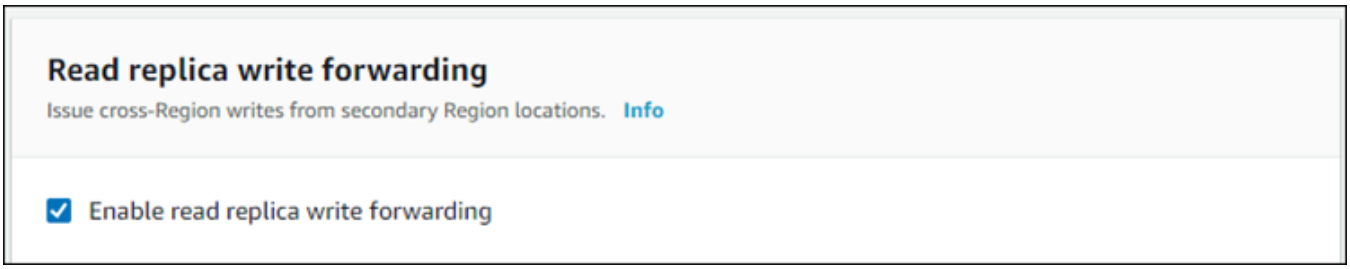
Sie können keine AWS-Region auswählen, die bereits über einen sekundären Aurora-DB-Cluster für dieselbe globale Aurora-Datenbank verfügt. Außerdem kann dies nicht dieselbe Region sein wie die des primären Aurora-DB-Clusters.

> **Note**
>
> Die globalen Datenbanken von Babelfish für Aurora PostgreSQL funktionieren nur in sekundären Regionen, wenn die Parameter, die die Babelfish-Einstellungen steuern, in diesen Regionen aktiviert sind. Weitere Informationen finden Sie unter Einstellungen der DB-Cluster-Parametergruppe für Babelfish



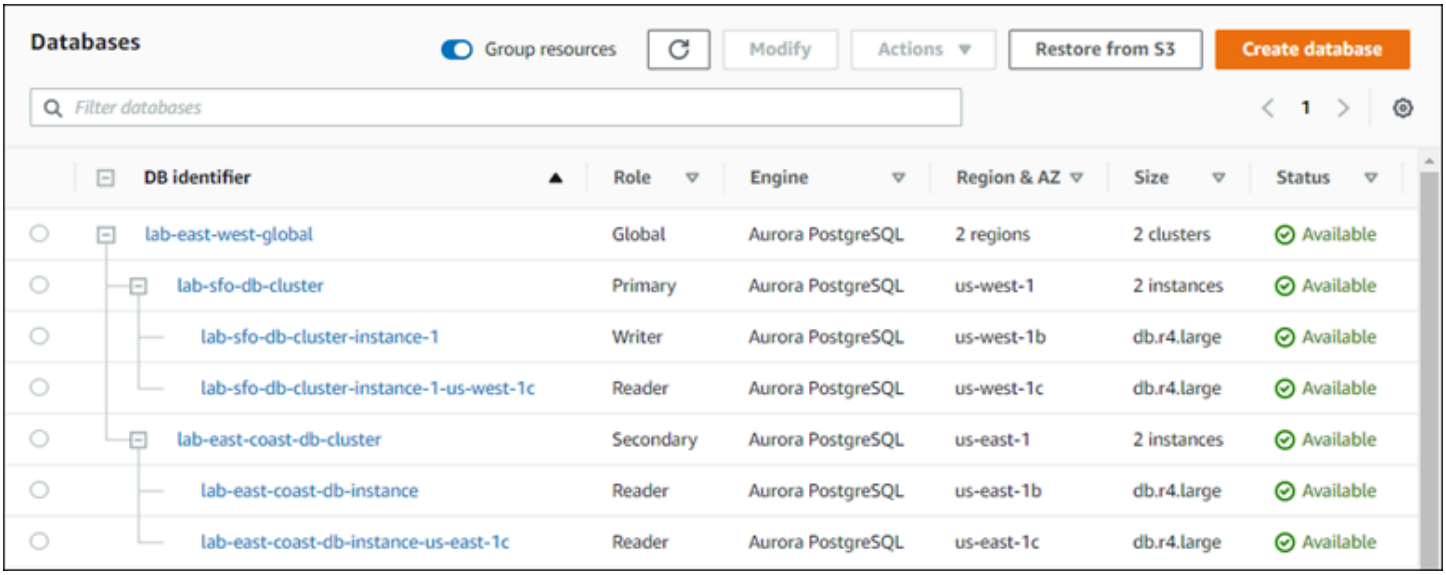
6. Füllen Sie die restlichen Felder für den sekundären Aurora-Cluster in der neuen AWS-Region aus. Dies sind die gleichen Konfigurationsoptionen wie für jede Aurora-DB-Cluster-Instance, mit Ausnahme der folgenden Option nur für Aurora MySQL–basierte globale Aurora-Datenbanken:

- Read Replica-Schreibweiterleitung aktivieren – Mit dieser optionalen Einstellung leiten die sekundären DB-Cluster Ihrer globalen Aurora-Datenbank Schreibvorgänge an den primären Cluster weiter. Weitere Informationen finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.



7. Wählen Sie AWS-Region hinzufügen aus.

Nachdem Sie die Region zu Ihrer globalen Aurora-Datenbank hinzugefügt haben, können Sie sie in der Liste der Datenbanken in AWS-Managementkonsole sehen, wie im Screenshot gezeigt.



AWS CLI

Fügen Sie eine sekundäre AWS-Region zu einer globalen Aurora-Datenbank wie folgt hinzu:

Um Ihrer globalen Datenbank mithilfe der CLI einen sekundären Cluster hinzuzufügen, müssen Sie bereits über das globale Cluster-Container-Objekt verfügen. Wenn Sie den `create-global-cluster`-Befehl noch nicht ausgeführt haben, finden Sie weitere Informationen zum CLI-Verfahren unter Erstellen einer globalen Datenbank von Amazon Aurora.

1. Verwenden Sie den `create-db-cluster`-CLI-Befehl mit dem Namen (`--global-cluster-identifier`) Ihrer globalen Aurora-Datenbank. Für andere Parameter, führen Sie die folgenden Schritte aus:
2. Wählen Sie für `--region` eine andere AWS-Region als die Ihrer primären Aurora-Region aus.
3. Wählen Sie bestimmte Werte für die Parameter `--engine-version` und `--engine` aus. Diese Werte entsprechen denen des primären Aurora-DB-Clusters in Ihrer globalen Aurora-Datenbank.
4. Geben Sie für einen verschlüsselten Cluster Ihre primäre AWS-Region als `--source-region` für die Verschlüsselung an.

Im folgenden Beispiel wird ein neuer Aurora-DB-Cluster erstellt und als schreibgeschützter sekundärer Aurora-DB-Cluster an eine globale Aurora-Datenbank angehängt. Im letzten Schritt wird dem neuen Aurora-DB-Cluster eine Aurora-DB-Instance hinzugefügt.

Für Linux, macOS oder Unix:

```
aws rds --region secondary_region \
  create-db-cluster \
    --db-cluster-identifier secondary_cluster_id \
    --global-cluster-identifier global_database_id \
    --engine aurora-mysql | aurora-postgresql \
    --engine-version version

aws rds --region secondary_region \
  create-db-instance \
    --db-instance-class instance_class \
    --db-cluster-identifier secondary_cluster_id \
    --db-instance-identifier db_instance_id \
    --engine aurora-mysql | aurora-postgresql
```

Für Windows:

```
aws rds --region secondary_region ^
  create-db-cluster ^
    --db-cluster-identifier secondary_cluster_id ^
    --global-cluster-identifier global_database_id_id ^
    --engine aurora-mysql | aurora-postgresql ^
    --engine-version version

aws rds --region secondary_region ^
```

```
  create-db-instance ^
    --db-instance-class instance_class ^
    --db-cluster-identifier secondary_cluster_id ^
    --db-instance-identifier db_instance_id ^
    --engine aurora-mysql | aurora-postgresql
```

RDS-API

Um einer globalen Aurora-Datenbank mit der RDS-API eine neue AWS-Region hinzuzufügen, führen Sie die Operation CreateDBCluster aus. Geben Sie die Kennung der vorhandenen globalen Datenbank mithilfe des Parameters `GlobalClusterIdentifier` an.

Erstellen eines Aurora-Headless-DB-Clusters in einer sekundären Region

Obwohl eine globale Aurora-Datenbank mindestens einen sekundären Aurora-DB-Cluster in einer AWS-Region außerhalb der primären Region benötigt, können Sie eine Headless-Konfiguration für den sekundären Cluster verwenden. Ein sekundärer Aurora-Headless-DB-Cluster hat keine DB-Instance. Diese Art der Konfiguration kann die Ausgaben für eine globale Aurora-Datenbank senken. In einem Aurora-DB-Cluster werden die Rechen- und Speicherressourcen entkoppelt. Ohne die DB-Instance wird Ihnen die Rechenleistung nicht in Rechnung gestellt, sondern nur der Speicherplatz. Richtig eingerichtet, wird das sekundäre Headless-Speichervolume mit dem primären Aurora-DB-Cluster synchonisiert.

Sie fügen den sekundären Cluster wie gewohnt beim Erstellen einer globalen Aurora-Datenbank hinzu. Wenn Sie alle Cluster in der globalen Datenbank erstellen, gehen Sie wie unter Erstellen einer globalen Datenbank von Amazon Aurora beschrieben vor. Wenn Sie bereits über einen DB-Cluster verfügen, den Sie als primären Cluster verwenden können, gehen Sie wie unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank beschrieben vor.

Nachdem der primäre Aurora-DB-Cluster mit der Replikation zum sekundären Cluster begonnen hat, löschen Sie die schreibgeschützte Aurora-DB-Instance aus dem sekundären Aurora-DB-Cluster. Dieser sekundäre Cluster gilt jetzt als „Headless“, da er keine DB-Instance mehr hat. Selbst ohne DB-Instance im sekundären Aurora-DB-Cluster synchronisiert Aurora das Speicher-Volume mit dem primären Aurora-DB-Cluster.

> **Warning**
>
> Um mit Aurora PostgreSQL einen Headless-Cluster in einer sekundären AWS-Region zu erstellen, verwenden Sie die AWS CLI- oder RDS-API, um die sekundäre AWS-Region

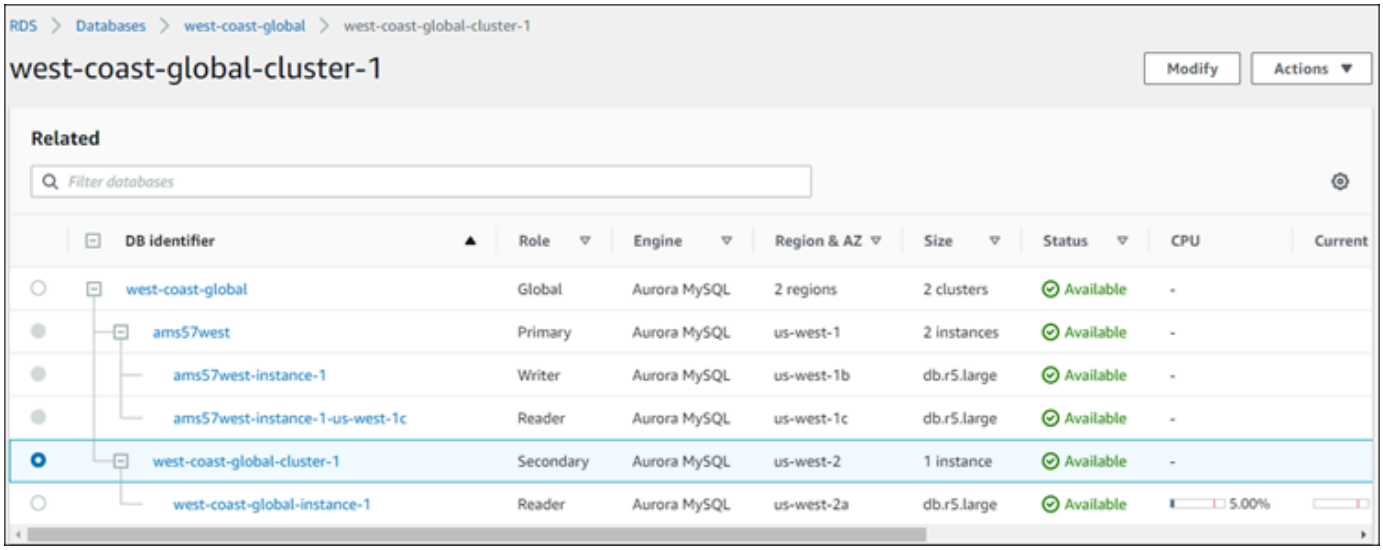
hinzuzufügen. Überspringen Sie den Schritt, um die Reader-DB-Instance für den sekundären Cluster zu erstellen. Derzeit wird das Erstellen eines Headless-Clusters in der RDS-Konsole nicht unterstützt. Informationen zur Verwendung der CLI- und API-Prozeduren finden Sie unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.
Wenn Ihre globale Datenbank eine niedrigere Engine-Version als 13.4, 12.8 oder 11.13 verwendet, kann das Erstellen einer Reader-DB-Instance in einer sekundären Region und das anschließende Löschen dieser Instance zu einem Aurora-PostgreSQL-Bereinigungsproblem auf der Writer-DB-Instance der primären Region führen. Wenn dieses Problem auftritt, starten Sie die Writer-DB-Instance der primären Region neu, nachdem Sie die Reader-DB-Instance der sekundären Region gelöscht haben.

So fügen Sie Ihrer globalen Aurora-Datenbank einen sekundären Aurora-Headless-DB-Cluster hinzu

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich der AWS-Managementkonsole die Option Datenbanken aus.
3. Wählen Sie die Aurora globale Datenbank aus, die einen sekundären Aurora-DB-Cluster benötigt. Stellen Sie sicher, dass der primäre Aurora-DB-Cluster is `Available`.
4. Wählen Sie für Aktionen die Option AWS-Region hinzufügen aus.
5. Wählen Sie auf der Seite Eine Region hinzufügen die sekundäre AWS-Region aus.

   Sie können keine AWS-Region auswählen, die bereits über einen sekundären Aurora-DB-Cluster für dieselbe globale Aurora-Datenbank verfügt. Außerdem kann dies nicht dieselbe Region sein wie die des primären Aurora-DB-Clusters.
6. Füllen Sie die restlichen Felder für den sekundären Aurora-Cluster in der neuen AWS-Region aus. Dies sind die gleichen Konfigurationsoptionen wie bei jeder Aurora-DB-Cluster-Instance.

   Bei einer Aurora MySQL–basierten globalen Aurora-Datenbank ignorieren Sie die Option Read Replica-Schreibweiterleitung aktivieren. Diese Option hat keine Funktion, nachdem Sie die Reader-Instance gelöscht haben.
7. Wählen Sie AWS-Region hinzufügen aus. Nachdem Sie die Region zu Ihrer globalen Aurora-Datenbank hinzugefügt haben, können Sie sie in der Liste der Datenbanken in AWS-Managementkonsole sehen, wie im Screenshot gezeigt.

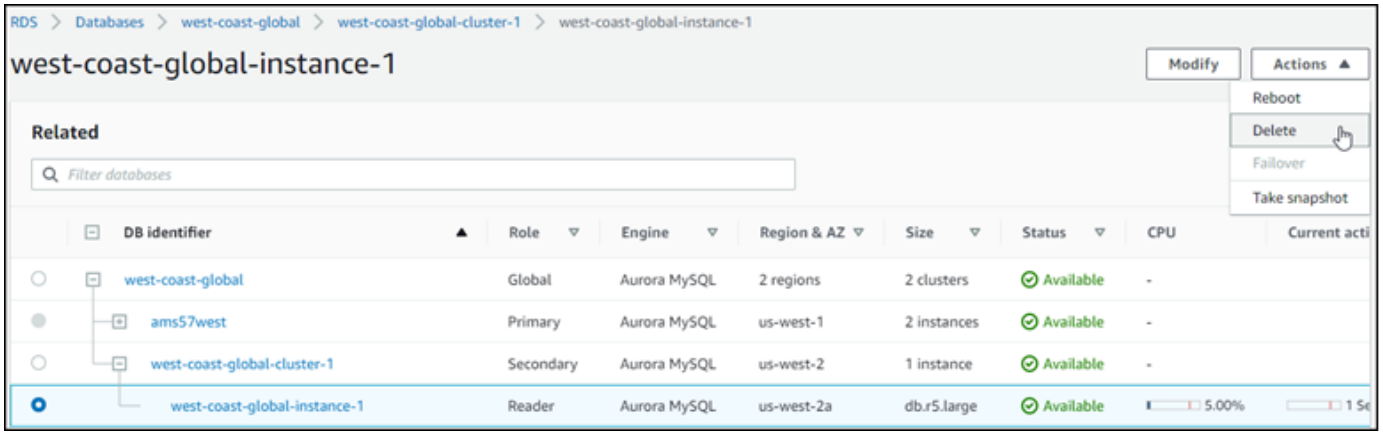


8. Überprüfen Sie den Status des sekundären Aurora-DB-Clusters und seiner Reader-Instance, bevor Sie fortfahren, indem Sie die AWS-Managementkonsole oder AWS CLI verwenden. Zum Beispiel:

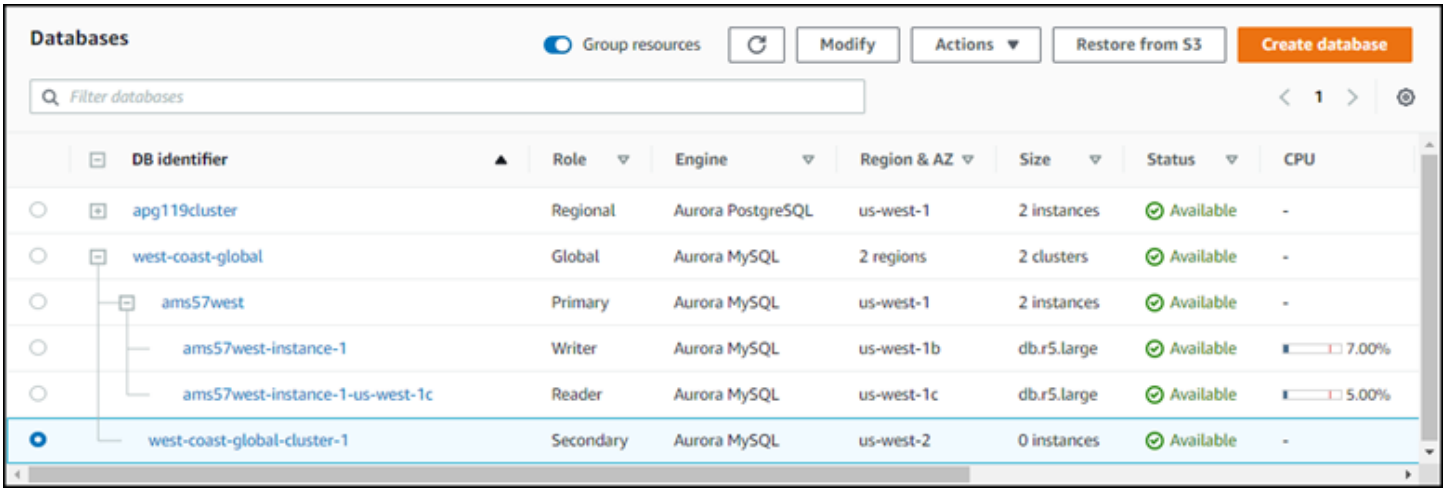
```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query
 '*[].[Status]' --output text
```

   Es kann mehrere Minuten dauern, bis der Status eines neu hinzugefügten sekundären Aurora-DB-Clusters von `creating` auf `available` wechselt. Wenn der Aurora-DB-Cluster verfügbar ist, können Sie die Reader-Instance löschen.

9. Wählen Sie die Reader-Instance im sekundären Aurora-DB-Cluster aus und klicken Sie dann auf Löschen.



Nach dem Löschen der Reader-Instance bleibt der sekundäre Cluster Teil der globalen Aurora-Datenbank. Ihm ist keine Instance zugeordnet, wie im Folgenden gezeigt.



Sie können diesen sekundären Aurora-Headless-DB-Cluster verwenden, um Ihre globale Amazon-Aurora-Datenbank nach einem ungeplanten Ausfall der primären AWS-Region manuell wiederherzustellen, wenn ein solcher Ausfall auftritt.

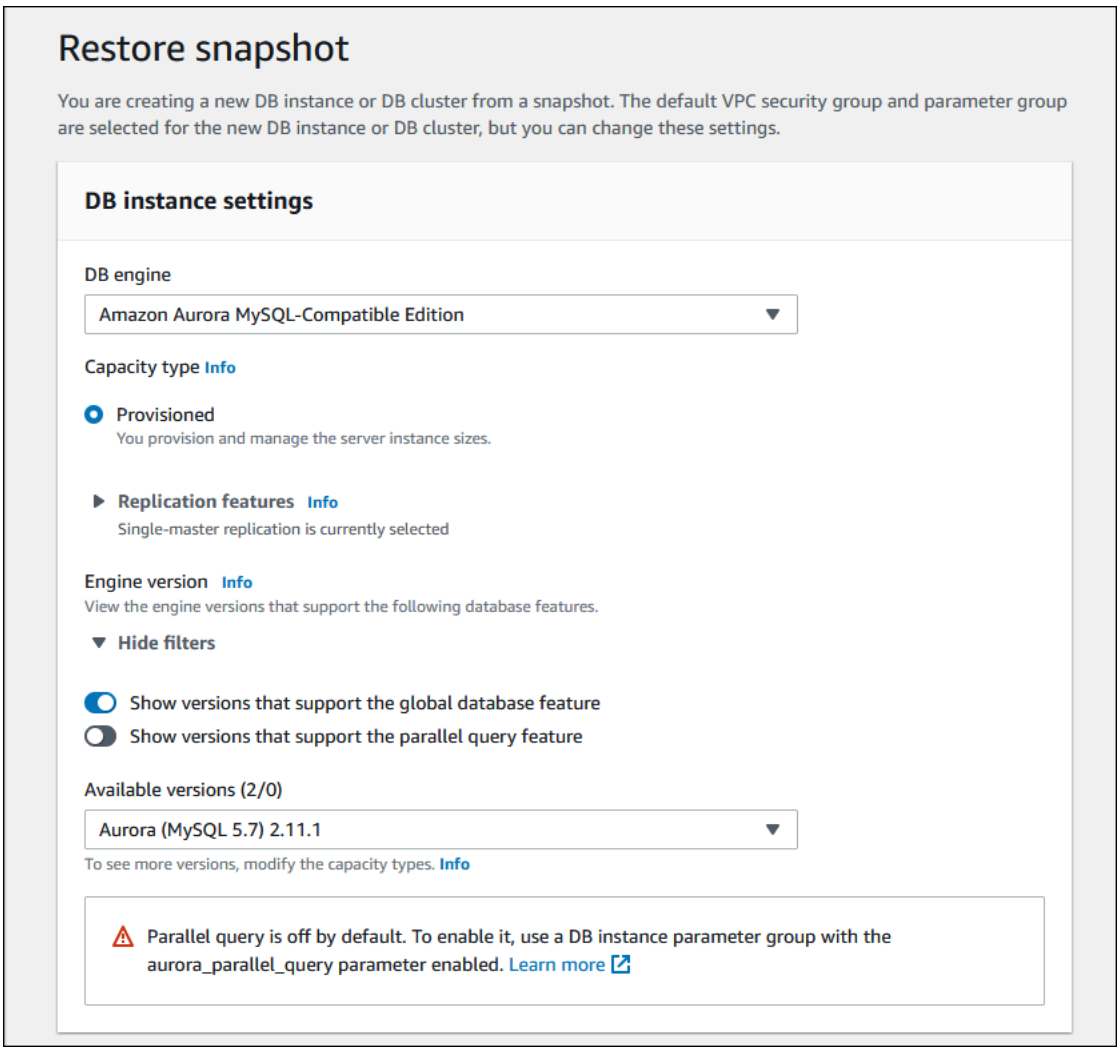
Erstellen einer globalen Amazon-Aurora-Datenbank aus einem Aurora- oder Amazon-RDS-Snapshot

Sie können einen Snapshot eines Aurora-DB-Clusters oder von einer Amazon-RDS-DB-Instance wiederherstellen, um ihn als Ausgangspunkt für Ihre globale Aurora-Datenbank zu verwenden. Sie stellen den Snapshot wieder her und erstellen gleichzeitig einen neuen von Aurora bereitgestellten DB-Cluster. Sie fügen dann dem wiederhergestellten DB-Cluster eine weitere AWS-Region hinzu und verwandeln sie so in eine globale Aurora-Datenbank. Jeder Aurora-DB-Cluster, den Sie auf diese Weise mit einem Snapshot erstellen, wird zum primären Cluster Ihrer globalen Aurora-Datenbank.

Der Snapshot, den Sie verwenden, kann von einem `provisioned`- oder einem `serverless` Aurora-DB-Cluster stammen.

Wählen Sie während des Wiederherstellungsvorgangs den gleichen DB-Engine-Typ wie den des Snapshots aus. Angenommen, Sie möchten einen Snapshot wiederherstellen, der aus einem Aurora Serverless-DB-Cluster mit Aurora PostgreSQL erstellt wurde. In diesem Fall erstellen Sie einen Aurora PostgreSQL-DB-Cluster mit derselben Aurora-DB-Engine und derselben Version.

Der wiederhergestellte DB-Cluster übernimmt die Rolle des primären Clusters für die globale Aurora-Datenbank, wenn Sie ihm eine AWS-Region hinzufügen. Alle in diesem primären Cluster enthaltenen Daten werden auf alle sekundären Cluster repliziert, die Sie Ihrer globalen Aurora-Datenbank hinzufügen.



Verwalten einer Amazon Aurora Global Database

Die meisten Verwaltungsvorgänge werden auf den einzelnen Clustern ausgeführt, aus denen eine globale Aurora-Datenbank besteht. Wenn Sie Group related resources (Gruppenbezogene Ressourcen) auf der Seite Databases (Datenbanken) in der Konsole auswählen, werden der primäre Cluster und der sekundäre Cluster unter der zugehörigen globalen Datenbank gruppiert. Um die AWS-Regionen zu finden, in denen die DB-Cluster einer globalen Datenbank ausgeführt werden sowie die zugehörige Aurora-DB-Engine, -Version und ihre Kennung, verwenden Sie die Registerkarte Konfiguration.

Die regionsübergreifenden Datenbank-Failover-Prozesse stehen nur globalen Aurora-Datenbankobjekten zur Verfügung, nicht einem einzelnen Aurora-DB-Cluster. Weitere Informationen hierzu finden Sie unter Verwenden von Umstellung oder Failover in Amazon Aurora Global Database.

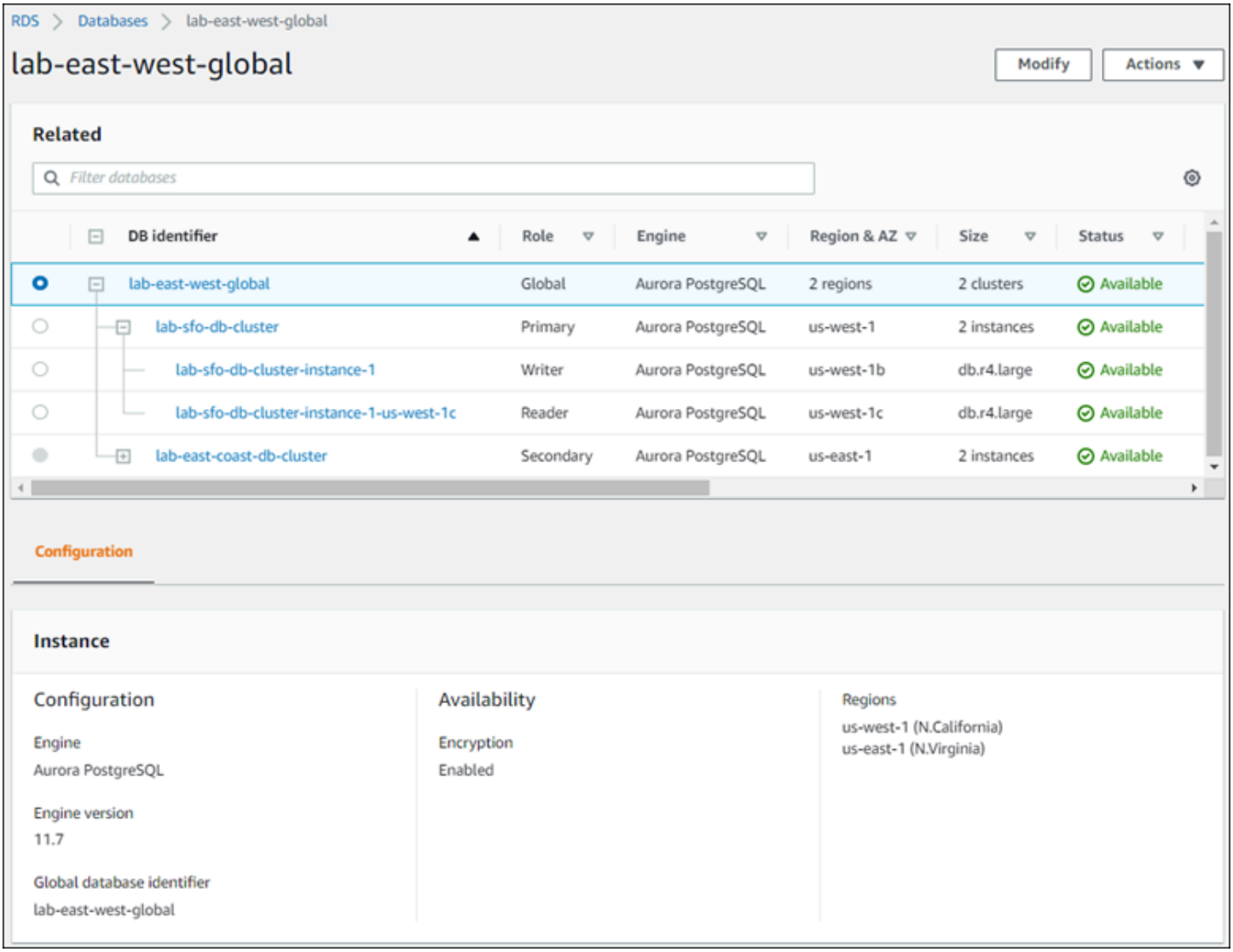
Informationen zum Wiederherstellen einer globalen Aurora-Datenbank nach einem ungeplanten Ausfall in ihrer primären Region finden Sie unter Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall.

Themen

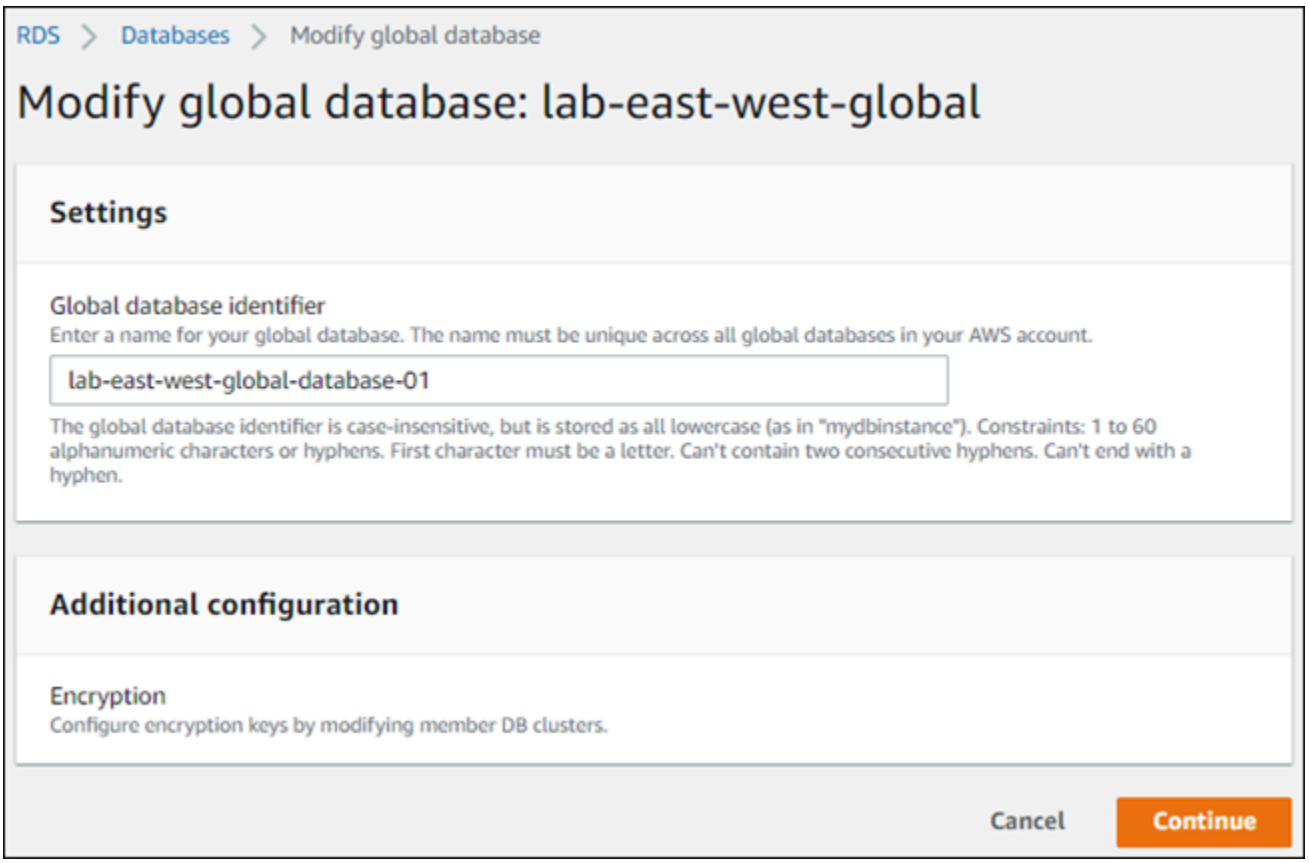
- Verändern einer Amazon Aurora Global Database
- Ändern von Parametern für eine Aurora globale Datenbank
- Entfernen eines Clusters aus einer Amazon Aurora Global Database
- Löschen einer Amazon Aurora Global Database
- Taggen von Ressourcen von Amazon Aurora Global Database

Verändern einer Amazon Aurora Global Database

In der AWS-Managementkonsole werden auf der Seite Datenbanken alle globalen Aurora-Datenbanken mit den jeweiligen primären und sekundären Clustern aufgeführt. Die globale Aurora-Datenbank hat ihre eigenen Konfigurationseinstellungen. Insbesondere verfügt sie über AWS-Regionen, die mit ihren primären und sekundären Clustern verknüpft sind, wie im folgenden Screenshot gezeigt.



Wenn Sie Änderungen an der Aurora globalen Datenbank vornehmen, haben Sie die Möglichkeit, Änderungen abzubrechen, wie im folgenden Screenshot gezeigt.



Wenn Sie Weiter wählen, bestätigen Sie die Änderungen.

Ändern von Parametern für eine Aurora globale Datenbank

Sie können die Parametergruppen jedes Aurora-DB-Cluster für jeden Aurora-Cluster in der globalen Aurora-Datenbank einzeln konfigurieren. Die meisten Parameter funktionieren wie bei anderen Aurora-Clusterarten. Wir empfehlen, dass Sie die Einstellungen zwischen allen Clustern in einer globalen Datenbank konsistent halten. Dies hilft, unerwartete Verhaltensänderungen zu vermeiden, wenn Sie einen sekundären Cluster zum primären Cluster hochstufen.

Sie sollten z. B. die gleichen Einstellungen für Zeitzonen und Zeichensätze verwenden, um inkonsistentes Verhalten zu vermeiden, wenn ein anderer Cluster die Rolle des primären Clusters übernimmt.

Die Konfigurationseinstellungen `aurora_enable_repl_bin_log_filtering` und `aurora_enable_replica_log_compression` haben keine Auswirkungen.

Entfernen eines Clusters aus einer Amazon Aurora Global Database

Sie können Aurora-DB-Cluster aus verschiedenen Gründen aus Ihrer Aurora globalen Datenbank entfernen. Beispielsweise möchten Sie möglicherweise einen Aurora-DB-Cluster aus einer Aurora globalen Datenbank entfernen, wenn der primäre Cluster herabgesetzt oder isoliert wird. Er wird dann zu einem eigenständigen bereitgestellten Aurora-DB-Cluster, der zum Erstellen einer neuen globalen Aurora-Datenbank verwendet werden könnte. Weitere Informationen hierzu finden Sie unter Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall.

Sie können bei Bedarf auch Aurora-DB-Cluster entfernen, wenn Sie eine globale Aurora-Datenbank löschen möchten, die Sie nicht mehr benötigen. Sie können die globale Aurora-Datenbank erst löschen, nachdem Sie alle zugehörigen Aurora-DB-Cluster entfernt haben, wobei der primäre Cluster zuletzt entfernt wird. Weitere Informationen finden Sie unter Löschen einer Amazon Aurora Global Database.

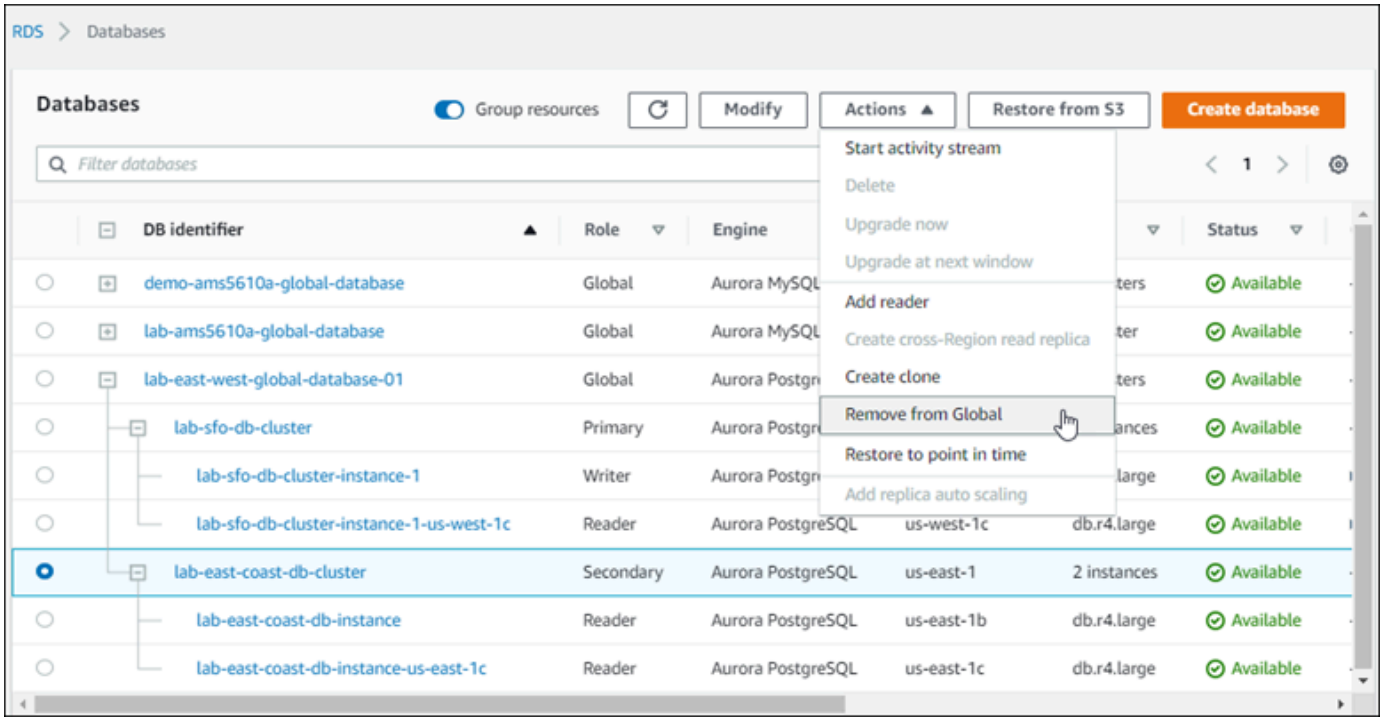
Wenn ein Aurora-DB-Cluster von der globalen Aurora-Datenbank getrennt wird, wird er nicht mehr mit dem primären Cluster synchronisiert. Er wird zu einem eigenständigen bereitgestellten Aurora-DB-Cluster mit vollen Lese-/Schreibfähigkeiten.

Sie können Aurora-DB-Cluster aus Ihrer globalen Aurora-Datenbank mithilfe von AWS-Managementkonsole, AWS CLI oder der RDS-API entfernen.

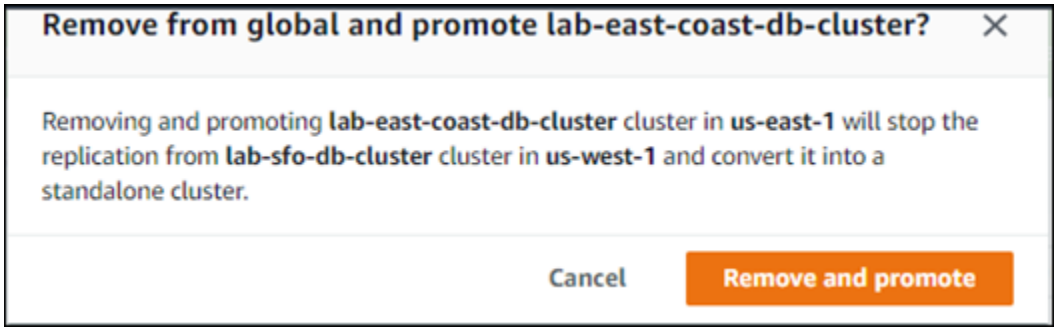
Konsole

So entfernen Sie einen Aurora-Cluster aus einer globalen Aurora-Datenbank:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie den Cluster auf der Seite Databases (Datenbanken) aus.
3. Wählen Sie unter Actions (Aktionen) die Option Remove from Global (Aus 'Global' entfernen) aus.

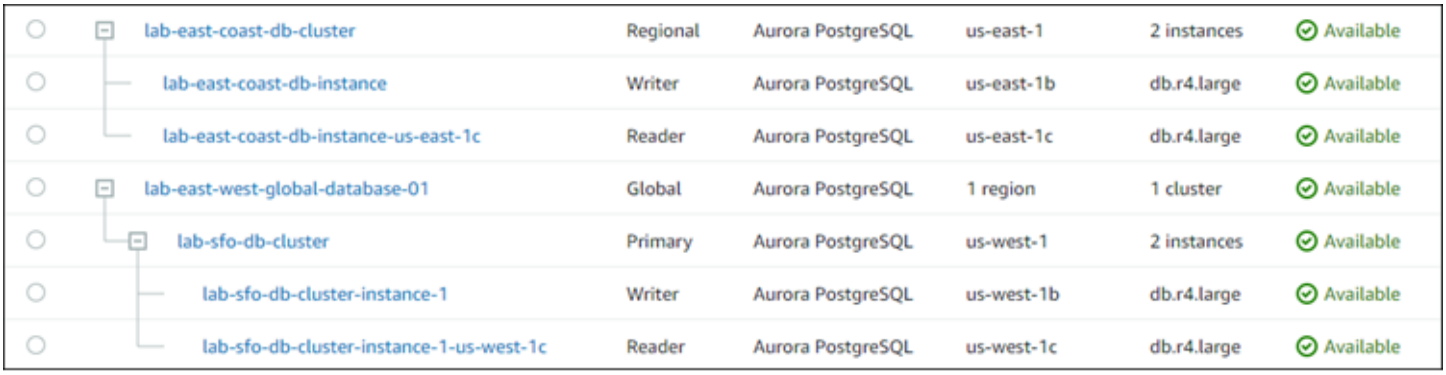


Sie sehen eine Aufforderung, um zu bestätigen, dass Sie den sekundären Cluster von der Aurora globalen Datenbank trennen möchten.



4. Wählen Sie Entfernen und hochstufen , um den Cluster aus der globalen Datenbank zu entfernen.

Der Aurora-DB-Cluster dient nicht mehr als sekundärer Cluster in der Aurora globalen Datenbank und ist nicht mehr mit dem primären DB-Cluster synchronisiert. Er ist ein eigenständiger Aurora-DB-Cluster mit voller Lese-/Schreibfähigkeit.



Nachdem Sie alle sekundären Cluster entfernt oder gelöscht haben, können Sie den primären Cluster auf die gleiche Weise entfernen. Sie können den primären Aurora-DB-Cluster erst von einer Aurora globalen Datenbank trennen (entfernen), nachdem Sie alle sekundären Cluster entfernt haben.

Die globale Aurora-Datenbank kann in der Datenbankliste mit null Regionen und AZs verbleiben. Sie können sie löschen, wenn Sie diese Aurora globale Datenbank nicht mehr verwenden möchten. Weitere Informationen finden Sie unter Löschen einer Amazon Aurora Global Database.

AWS CLI

Führen Sie den CLI-Befehl remove-from-global-cluster mit den Parametern im Folgenden aus, um einen Aurora-Cluster von einer globalen Aurora-Datenbank zu entfernen:

- `--global-cluster-identifier` – Der Name (identifier) Ihrer Aurora globalen Datenbank.
- `--db-cluster-identifier` – Der Name jedes Aurora-DB-Clusters, der aus der Aurora globalen Datenbank entfernt werden soll. Entfernen Sie alle sekundären Aurora-DB-Cluster, bevor Sie den primären Cluster entfernen.

Bei den folgenden Beispielen wird zuerst ein sekundärer Cluster und dann der primäre Cluster aus einer globalen Aurora-Datenbank entfernt.

Für Linux, macOS oder Unix:

```
aws rds --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
  remove-from-global-cluster \
```

```
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_database_id
```

Wiederholen Sie den `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` -Befehl für jede sekundäre AWS-Region in Ihrer globalen Aurora-Datenbank.

Für Windows:

```
aws rds --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id

aws rds --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_database_id
```

Wiederholen Sie den `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN`-Befehl für jede sekundäre AWS-Region in Ihrer globalen Aurora-Datenbank.

RDS-API

Führen Sie die Aktion RemoveFromGlobalCluster aus, um einen Aurora-Cluster mit der RDS-API aus einer globalen Aurora-Datenbank zu entfernen.

Löschen einer Amazon Aurora Global Database

Da eine Aurora globale Datenbank üblicherweise geschäftskritische Daten enthält, ist es nicht möglich, die globale Datenbank und ihre zugeordneten Cluster in einem einzigen Schritt zu löschen. Um eine globale Aurora-Datenbank zu löschen, führen Sie die folgenden Schritte aus:

- Entfernen Sie alle sekundären DB-Cluster aus der Aurora globalen Datenbank. Jeder Cluster wird zu einem eigenständigen Aurora-DB-Cluster. Um zu erfahren wie, siehe Entfernen eines Clusters aus einer Amazon Aurora Global Database.
- Löschen Sie von jedem eigenständigen Aurora-DB-Cluster alle Aurora-Replicas.
- Entfernen Sie den primären DB-Cluster aus der globalen Aurora-Datenbank. Dieser Cluster wird zu einem eigenständigen Aurora-DB-Cluster.

- Löschen Sie vom primären Aurora-DB-Cluster zuerst alle Aurora-Replicas und löschen Sie dann die Writer-DB-Instance.

Durch das Löschen der Writer-Instance aus dem neu eigenständigen Aurora-DB-Cluster werden normalerweise auch der Aurora-DB-Cluster und die Aurora globale Datenbank entfernt.

Weitere allgemeine Informationen finden Sie unter Löschen einer DB-Instance aus einem Aurora-DB-Cluster.

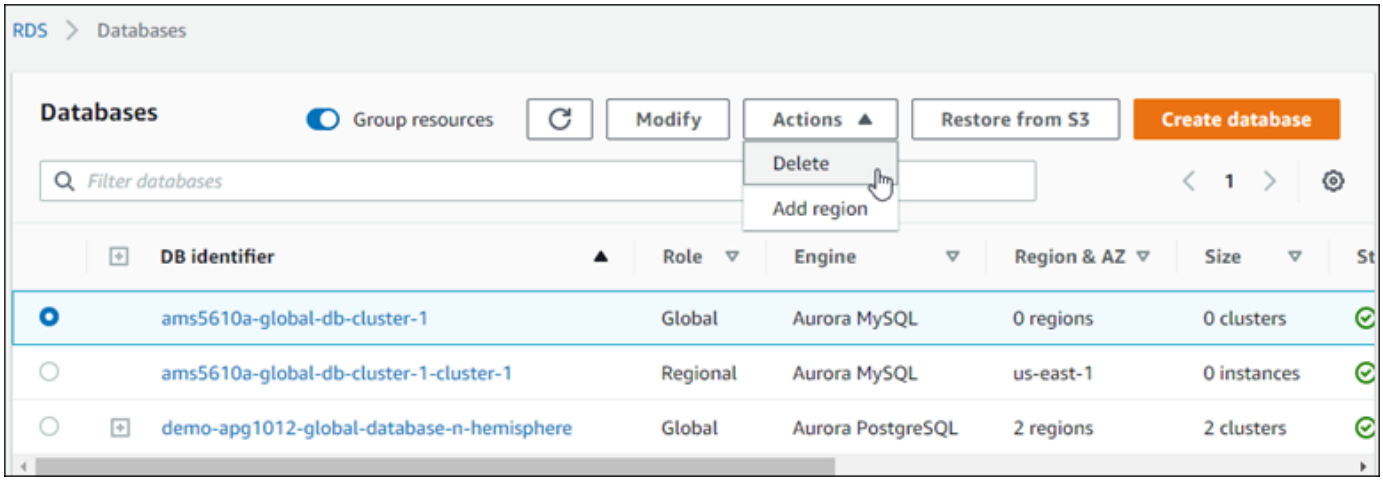
Um eine globale Aurora-Datenbank zu löschen, können Sie AWS-Managementkonsole, AWS CLI oder die RDS-API verwenden.

Konsole

So löschen Sie eine globale Aurora-Datenbank

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Datenbanken und suchen Sie die Aurora globale Datenbank, die Sie löschen möchten, in der Liste.
3. Bestätigen Sie, dass alle Cluster aus der globalen Aurora-Datenbank entfernt sind. Die globale Aurora-Datenbank sollte 0 Regionen und AZs sowie eine Größe von 0 Clustern enthalten.

   Wenn die Aurora globale Datenbank Aurora-DB-Cluster enthält, können Sie sie nicht löschen. Trennen Sie bei Bedarf die primären und sekundären Aurora-DB-Cluster von der Aurora globalen Datenbank. Weitere Informationen finden Sie unter Entfernen eines Clusters aus einer Amazon Aurora Global Database.
4. Wählen Sie Ihre globale Aurora-Datenbank in der Liste aus, und wählen Sie dann Löschen im Menü Aktionen aus.



AWS CLI

Um eine globale Aurora-Datenbank zu löschen, führen Sie den CLI-Befehl delete-global-cluster mit dem Namen der AWS-Region und der globalen Aurora-Datenbank-ID aus, wie im folgenden Beispiel gezeigt.

Für Linux, macOS oder Unix:

```
aws rds --region primary_region delete-global-cluster \
   --global-cluster-identifier global_database_id
```

Für Windows:

```
aws rds --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_database_id
```

RDS-API

Führen Sie die API-Operation DeleteGlobalCluster aus, um einen Cluster, der Teil einer globalen Aurora-Datenbank ist, zu löschen.

Taggen von Ressourcen von Amazon Aurora Global Database

Mit der Funktion von Aurora Global Database können Sie RDS-Tags auf Ressourcen auf verschiedenen Ebenen innerhalb einer globalen Datenbank anwenden. Wenn Sie nicht wissen, wie Tags bei AWS- oder Aurora-Ressourcen verwendet werden, informieren Sie sich entsprechend unter

Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen, ehe Sie Tags innerhalb Ihrer globalen Datenbank anwenden.

> **Note**
>
> Da AWS Tag-Daten im Rahmen des Mechanismus der Kostenberichterstattung verarbeitet, sollten Sie keine sensiblen Daten oder persönlich identifizierbare Informationen (PII) in die Tag-Namen oder -Werte aufnehmen.

Sie können Tags auf die folgenden Ressourcenarten innerhalb einer globalen Datenbank anwenden:

- Das Container-Objekt für Ihre gesamte globale Datenbank. Diese Ressource gilt als globaler Cluster.

  Nachdem Sie den globalen Cluster erstellt haben, indem Sie in der Konsole den Vorgang AWS-Region hinzufügen ausgeführt haben, können Sie mithilfe der Detailseite für den globalen Cluster Tags hinzufügen. Auf der Registerkarte Tags der Detailseite für den globalen Cluster können Sie Tags und ihre zugehörigen Werte hinzufügen, entfernen oder ändern, indem Sie Tags verwalten auswählen.

  Mit der AWS CLI und der RDS-API können Sie dem globalen Cluster gleichzeitig mit dessen Erstellung Tags hinzufügen. Sie können auch Tags für einen vorhandenen globalen Cluster hinzufügen, entfernen oder ändern.
- Den primären Cluster. Sie verwenden hier dieselben Verfahren für die Arbeit mit Tags wie für eigenständige Aurora-Cluster. Sie können die Tags einrichten, bevor Sie den ursprünglichen Aurora-Cluster in eine globale Datenbank umwandeln. Sie können auch Tags und die zugehörigen Werte hinzufügen, entfernen oder ändern, indem Sie auf der Detailseite des DB-Clusters die Option Tags verwalten auf der Registerkarte Tags auswählen.
- Alle sekundären Cluster. Sie verwenden hier dieselben Verfahren für die Arbeit mit Tags wie für eigenständige Aurora-Cluster. Sie können die Tags gleichzeitig mit der Erstellung eines sekundären Aurora-Clusters einrichten, indem Sie die Aktion AWS-Region hinzufügen in der Konsole verwenden. Sie können auch Tags und die zugehörigen Werte hinzufügen, entfernen oder ändern, indem Sie auf der Detailseite des DB-Clusters die Option Tags verwalten auf der Registerkarte Tags auswählen.
- Einzelne DB-Instances innerhalb des primären oder sekundären Clusters. Sie verwenden hier dieselben Verfahren für die Arbeit mit Tags wie für Aurora- oder RDS-DB-Instances. Sie können

die Tags gleichzeitig mit dem Hinzufügen einer neuen DB-Instance zum Aurora-Cluster einrichten, indem Sie die Aktion Reader hinzufügen in der Konsole verwenden. Sie können auch Tags und die zugehörigen Werte hinzufügen, entfernen oder ändern, indem Sie auf der Detailseite der DB-Instance die Option Tags verwalten auf der Registerkarte Tags auswählen.

Nachfolgend einige Beispiele für die Arten von Tags, die Sie innerhalb einer globalen Datenbank zuweisen können:

- Sie können dem globalen Cluster Tags hinzufügen, um allgemeine Informationen über Ihre Anwendung zu erfassen, z. B. anonymisierte Kennungen, die Eigentümer und Kontakte innerhalb Ihrer Organisation repräsentieren. Sie können Tags verwenden, um Eigenschaften der Anwendung zu kennzeichnen, die die globale Datenbank verwendet.
- Sie können dem primären Cluster und den sekundären Clustern Tags hinzufügen, um die Kosten für Ihre Anwendung auf AWS-Regionsebene nachzuverfolgen. Einzelheiten zu diesem Verfahren finden Sie unter So funktioniert die AWS Abrechnung mit Tags in Amazon RDS.
- Sie können bestimmten DB-Instances mit den Aurora-Clustern Tags hinzufügen, um deren speziellen Zweck anzugeben. Sie könnten zum Beispiel innerhalb des primären Clusters eine Reader-Instance mit einer niedrigen Failover-Priorität haben, die ausschließlich für die Berichtsgenerierung verwendet wird. Durch ein Tag hebt sich diese spezielle DB-Instance von anderen Instances ab, die für eine hohe Verfügbarkeit innerhalb des primären Clusters vorgesehen sind.
- Sie können Tags auf allen Ebenen Ihrer globalen Datenbankressourcen verwenden, um den Zugriff über IAM-Richtlinien zu steuern. Weitere Informationen finden Sie unter Steuern des Zugriffs auf AWS-Ressourcen im AWS-Identity-and-Access-Management-Benutzerhandbuch.

  > **Tip**
  >
  > In der AWS-Managementkonsole fügen Sie Tags zum globalen Cluster-Container in einem separaten Schritt hinzu, nachdem Sie ihn erstellt haben. Wenn Sie Zeitintervalle vermeiden möchten, in denen der globale Cluster ohne Tags für die Zugriffskontrolle existiert, können Sie die Tags während der `CreateGlobalCluster`-Operation anwenden, indem Sie die Ressource mit der AWS CLI, der RDS-API oder einer CloudFormation-Vorlage erstellen.

- Sie können Tags auf Cluster-Ebene oder für den globalen Cluster verwenden, um Informationen zur Qualitätssicherung und zum Testen Ihrer Anwendung zu erfassen. Sie können beispielsweise ein Tag auf einem DB-Cluster angeben, um zu erfassen, wann Sie das letzte Mal eine Umstellung

zu diesem Cluster durchgeführt haben. Sie können ein Tag für den globalen Cluster angeben, um den Zeitpunkt der letzten Notfallwiederherstellungsübung für die gesamte Anwendung zu erfassen.

AWS CLI-Beispiele für das Taggen globaler Datenbanken

Die folgenden AWS CLI-Beispiele zeigen, wie Sie Tags für alle Arten von Aurora-Ressourcen in Ihrer globalen Datenbank angeben und untersuchen können.

Sie können Tags für den globalen Cluster-Container mit dem `create-global-cluster`-Befehl angeben. Der folgende Befehl erstellt einen globalen Cluster und weist diesem 2 Tags zu. Die Tags haben die Schlüssel `tag1` und `tag2`.

```
$  aws rds create-global-cluster --global-cluster-identifier my_global_cluster_id \
  --engine aurora-mysql --tags Key=tag1,Value=val1 Key=tag2,Value=val2
```

Sie können die Tags im globalen Cluster-Container mit dem `describe-global-clusters`-Befehl auflisten. Wenn Sie mit Tags arbeiten, führen Sie diesen Befehl normalerweise zuerst aus, um den Amazon-Ressourcennamen (ARN) des globalen Clusters abzurufen. Sie verwenden den ARN als Parameter in nachfolgenden Befehlen für die Arbeit mit Tags. Mit dem folgenden Befehl werden die Tag-Informationen im `TagList`-Attribut angezeigt. Er zeigt auch den ARN, der in den späteren Beispielen als Parameter verwendet wird.

```
$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "GlobalClusterArn": "my_global_cluster_arn",
            ...
            "TagList": [
                {
                    "Value": "val1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                }
            ]
```

```
        }
    ]
}
```

Sie können mit dem `add-tags-to-resource`-Befehl neue Tags hinzufügen. Mit diesem Befehl geben Sie anstelle der ID den Amazon-Ressourcennamen (ARN) des globalen Clusters an. Wenn Sie ein neues Tag hinzufügen, dessen Name mit dem eines vorhandenen Tags identisch ist, wird der vorherige Tag-Wert überschrieben. Wenn Sie Leerzeichen oder Sonderzeichen in die Tag-Werte aufnehmen, setzen Sie die Werte entsprechend Ihrem Betriebssystem oder Ihrer Befehls-Shell in Anführungszeichen. Im folgenden Beispiel werden die Tags des globalen Clusters aus dem vorherigen Beispiel geändert. Ursprünglich hatte der Cluster Tags mit den Schlüsseln `tag1` und `tag2`. Nach Ausführung des Befehls hat der globale Cluster ein neues Tag mit dem Schlüssel `tag3` und das Tag mit dem Schlüssel `tag1` hat einen anderen Wert.

```
$  aws rds add-tags-to-resource --resource-name my_global_cluster_arn \
  --tags Key=tag1,Value="new value for tag1" Key=tag3,Value="entirely new tag"

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                },
                {
                    "Value": "val2",
                    "Key": "tag2"
                },
                {
                    "Value": "entirely new tag",
                    "Key": "tag3"
                }
            ]
        }
    ]
}
```

Mit dem `remove-tags-from-resource`-Befehl können Sie ein Tag aus dem globalen Cluster löschen. Mit diesem Befehl geben Sie nur einen Satz von Tag-Schlüsseln ohne Tag-Werte an. Im folgenden Beispiel werden die Tags des globalen Clusters aus dem vorherigen Beispiel geändert. Ursprünglich hatte der Cluster Tags mit den Schlüsseln `tag1`, `tag2` und `tag3`. Nach Ausführung des Befehls bleibt nur das Tag mit dem Schlüssel `tag1` übrig.

```
$  aws rds remove-tags-from-resource --resource-name my_global_cluster_arn --tag-keys
 tag2 tag3

$  aws rds describe-global-clusters --global-cluster-identifier my_global_cluster_id
{
    "GlobalClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            ...
            "TagList": [
                {
                    "Value": "new value for tag1",
                    "Key": "tag1"
                }
            ]
        }
    ]
}
```

Verbinden mit Amazon Aurora Global Database

Jede globale Aurora-Datenbank verfügt über einen Writer-Endpunkt, der automatisch von Aurora aktualisiert wird, um Anfragen an die aktuelle Writer-Instance des primären DB-Clusters weiterzuleiten. Mit dem Writer-Endpunkt müssen Sie Ihre Verbindungszeichenfolge nicht ändern, nachdem Sie den Standort der primären Region mithilfe der verwalteten Umstellungs- und Failover-Funktionen von Aurora Global Database geändert haben. Weitere Informationen zur Verwendung des Writer-Endpunkts zusammen mit den Umstellungs- und Failover-Funktionen von Aurora Global Database finden Sie unter Verwenden von Umstellung oder Failover in Amazon Aurora Global Database. Informationen zum Verbinden mit einer globalen Aurora-Datenbank über RDS-Proxy finden Sie unter Verwenden von RDS-Proxy mit globalen Aurora-Datenbanken.

Themen

- Auswählen des Endpunkts, der Ihren Anwendungsanforderungen entspricht

- Anzeigen der Endpunkte einer globalen Amazon-Aurora-Datenbank
- Überlegungen zur Verwendung globaler Writer-Endpunkte

Auswählen des Endpunkts, der Ihren Anwendungsanforderungen entspricht

Die Verbindung zu einer globalen Aurora-Datenbank hängt davon ab, ob Sie aus der Datenbank lesen oder in diese schreiben müssen sowie von der AWS-Region, an die Sie Ihre Anfragen weiterleiten möchten. Nachfolgend ein paar typische Anwendungsfälle:

- Weiterleiten von Anfragen an die Writer-Instance: Stellen Sie eine Verbindung zum Writer-Endpunkt von Aurora Global Database her, wenn Sie Data Manipulation Language (DML)- und Data Definition Language (DDL)-Anweisungen ausführen müssen oder wenn Sie eine hohe Konsistenz zwischen Lese- und Schreibvorgängen benötigen. Dieser Endpunkt leitet Anfragen an die Writer-Instance im primären Cluster Ihrer globalen Datenbank weiter. Dieser Endpunkt wird automatisch aktualisiert, um Anfragen an die Writer-Instance weiterzuleiten, sodass Sie Ihre Anwendung nicht jedes Mal aktualisieren müssen, wenn Sie den Writer-Standort in Ihrem globalen Cluster ändern. Sie können den globalen Endpunkt auch verwenden, um regionsübergreifende Lese-/Schreibanfragen an Ihren Writer zu senden.

  > **Note**
  >
  > Wenn Sie Ihre globale Datenbank eingerichtet haben, bevor der Writer-Endpunkt von Aurora Global Database verfügbar war, stellt Ihre Anwendung möglicherweise eine Verbindung zum Cluster-Endpunkt des primären Clusters her. In diesem Fall empfehlen wir, Ihre Verbindungseinstellungen so zu ändern, dass stattdessen der globale Writer-Endpunkt verwendet wird. Dadurch müssen Sie Ihre Verbindungseinstellungen nicht nach jeder Umstellung oder jedem Failover von Aurora Global Database ändern.
  > Der erste Teil des Namens des Writer-Endpunkts ist der Name Ihrer globalen Aurora-Datenbank. Wenn Sie also Ihre globale Aurora-Datenbank umbenennen, ändert sich der Name des Writer-Endpunkts, und jeder Code, der ihn verwendet, muss mit dem neuen Namen aktualisiert werden.

- Skalieren von Lesevorgängen näher an der Region Ihrer Anwendung: Um schreibgeschützte Anfragen in derselben AWS-Region wie Ihre Anwendung oder in einer nahegelegenen zu skalieren, stellen Sie eine Verbindung zum Reader-Endpunkt des primären oder sekundären Aurora-Clusters her.

- Skalieren von Lesevorgängen mit gelegentlichen regionsübergreifenden Schreibvorgängen: Stellen Sie für gelegentliche DML-Anweisungen, z. B. zur Wartung und Datenbereinigung, eine Verbindung zum Reader-Endpunkt eines sekundären Clusters her, für den die Schreibweiterleitung aktiviert ist. Mit der Schreibweiterleitung leitet Aurora die Schreibanweisungen automatisch an den Writer in der primären Region Ihrer globalen Aurora-Datenbank weiter. Die Schreibweiterleitung bietet die folgenden Vorteile:
  - Sie müssen sich nicht die Mühe machen, eine Konnektivität zwischen dem sekundären und dem primären Cluster herzustellen, um regionsübergreifende Schreibvorgänge zu senden.
  - Sie müssen Lese- und Schreibanforderungen in der Anwendung nicht aufteilen.
  - Sie müssen keine komplexe Logik entwickeln, um die Konsistenz von Read-After-Write-Anfragen zu gewährleisten.

  Bei der Schreibweiterleitung müssen Sie jedoch Ihren Anwendungscode oder Ihre Konfiguration aktualisieren, um nach einem regionsübergreifenden Failover oder einer solchen Umstellung eine Verbindung zum Reader-Endpunkt der neu hochgestuften primären Region herzustellen. Es wird empfohlen, die Latenz von Vorgängen, die über die Schreibweiterleitung ausgeführt werden, zu überwachen, um den Overhead bei der Verarbeitung der Schreibanforderungen zu überprüfen. Schließlich unterstützt die Schreibweiterleitung bestimmte MySQL- oder PostgreSQL-Operationen nicht, wie beispielsweise DDL-Änderungen oder `SELECT FOR UPDATE`-Anweisungen.

  Weitere Informationen zur Verwendung der Schreibweiterleitung innerhalb von AWS-Regionen finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

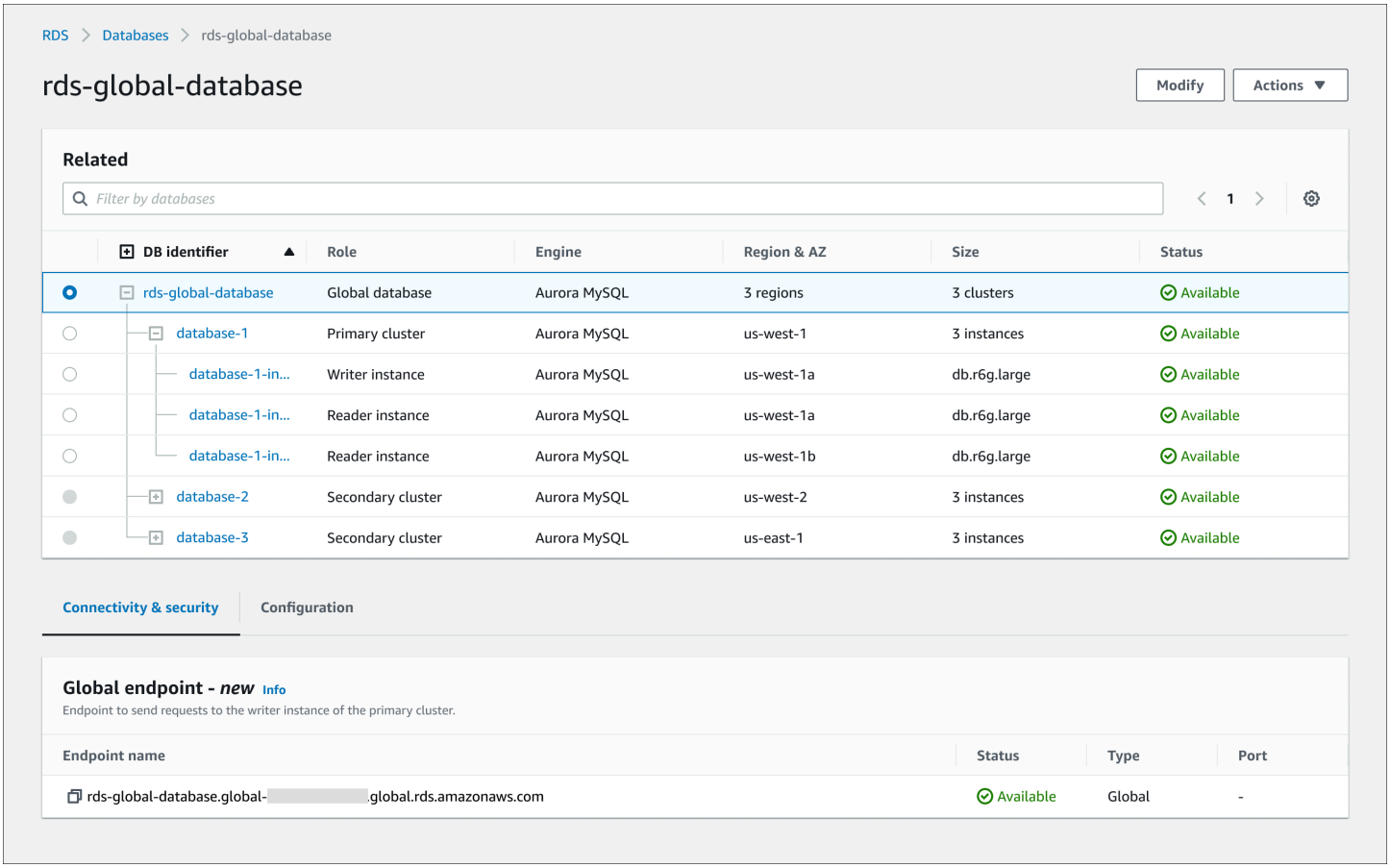
Einzelheiten zu den verschiedenen Arten von Aurora-Endpunkten finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

Anzeigen der Endpunkte einer globalen Amazon-Aurora-Datenbank

Wenn Sie eine globale Aurora-Datenbank in der Konsole anzeigen, können Sie alle Endpunkte sehen, die mit allen ihren Clustern verknüpft sind. Die folgende Abbildung zeigt ein Beispiel für die Arten von Endpunkten, die Sie sehen, wenn Sie sich die Details für Ihren primären DB-Cluster ansehen:

- Globaler Writer – Dies ist der einzelne Lese-/Schreibendpunkt, der immer auf die aktuelle Writer-DB-Instance für den globalen Datenbank-Cluster verweist.
- Writer – Dies ist der Verbindungsendpunkt für Lese-/Schreibanforderungen an den primären DB-Cluster im globalen Datenbankcluster.

- Reader – Dies ist der Verbindungsendpunkt für schreibgeschützte Anfragen an einen primären oder sekundären DB-Cluster im globalen Datenbank-Cluster. Zum Minimieren der Latenzzeit wählen Sie denjenigen Reader-Endpunkt aus, der sich in Ihrer AWS-Region oder der Ihnen am nächsten gelegenen AWS-Region befindet.



Konsole

So zeigen Sie die Endpunkte einer globalen Datenbank an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie in der Liste die globale Datenbank oder den primären oder sekundären DB-Cluster aus, dessen Endpunkte Sie anzeigen möchten.
4. Wählen Sie die Registerkarte Konnektivität und Sicherheit aus, um die Endpunktdetails zu sehen. Die angezeigten Endpunkte hängen vom Typ des Clusters ab, den Sie ausgewählt haben:

- Globale Datenbank – der globale Writer-Endpunkt
- Primärer DB-Cluster – der globale Writer-Endpunkt sowie der Cluster-Endpunkt und der Reader-Endpunkt für den primären Cluster
- Sekundärer DB-Cluster – der Cluster-Endpunkt und der Reader-Endpunkt für den sekundären Cluster Auf einem sekundären Cluster zeigt der Cluster-Endpunkt den Status inaktiv an, da er keine Schreibanforderungen bearbeitet. Sie können weiterhin eine Verbindung zum Cluster-Endpunkt herstellen, jedoch nur für Leseabfragen.

AWS CLI

Um den Writer-Endpunkt des globalen Clusters anzuzeigen, verwenden Sie den AWS CLI-Befehl describe-global-clusters wie im folgenden Beispiel gezeigt.

```
aws rds describe-global-clusters --region aws_region
{
    "GlobalClusters": [
        {
            "GlobalClusterIdentifier": "global_cluster_id",
            "GlobalClusterResourceId": "cluster-unique_string",
            "GlobalClusterArn": "arn:aws:rds::123456789012:global-
cluster:global_cluster_id",
            "Status": "available",
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",
            "GlobalClusterMembers": [

              ...
            ],
            "Endpoint":
 "global_cluster_id.global-unique_string.global.rds.amazonaws.com"
        }
    ]
}
```

Um die Cluster- und Reader-Endpunkte für Mitglieds-DB-Cluster des globalen Clusters anzuzeigen, verwenden Sie den AWS CLI-Befehl describe-db-clusters wie im folgenden Beispiel gezeigt. Bei den für `Endpoint` und `ReaderEndpoint` zurückgegebenen Werten handelt es sich um die Endpunkte für den Cluster bzw. Reader.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier db_cluster_id
{
    "DBClusters": [
        {
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "az_1",
                "az_2",
                "az_3"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "db_cluster_id",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "DBSubnetGroup": "default",
            "Status": "available",
            "EarliestRestorableTime": "2023-08-01T18:21:11.301Z",
            "Endpoint":
 "db_cluster_id.cluster-unique_string.primary_region.rds.amazonaws.com",
            "ReaderEndpoint": "db_cluster_id.cluster-
ro-unique_string.primary_region.rds.amazonaws.com",
            "MultiAZ": false,
            "Engine": "aurora-mysql",
            "EngineVersion": "5.7.mysql_aurora.2.11.2",

            "ReadReplicaIdentifiers": [
                "arn:aws:rds:secondary_region:123456789012:cluster:db_cluster_id"
            ],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "db_instance_id",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                }
            ],

            ...
            "TagList": [],
            "GlobalWriteForwardingRequested": false
        }
    ]
}
```

RDS-API

Verwenden Sie die RDS-API-Operation DescribeGlobalClusters, um den Writer-Endpunkt des globalen Clusters anzuzeigen. Verwenden Sie die RDS-API-Operation DescribeDBClusters, um die Cluster- und Reader-Endpunkte für Mitglieds-DB-Cluster des globalen Clusters anzuzeigen.

Überlegungen zur Verwendung globaler Writer-Endpunkte

Sie können Writer-Endpunkte von Aurora Global Database effektiv nutzen, indem Sie sich an die folgenden Richtlinien und bewährten Methoden halten:

- Um Unterbrechungen nach einem regionsübergreifenden Failover oder einer solchen Umstellung zu minimieren, können Sie eine VPC-Konnektivität zwischen Ihrer Anwendungsdatenverarbeitung und Ihren primären und sekundären AWS-Regionen einrichten. Angenommen, Sie haben Anwendungen oder Clientsysteme, die in derselben VPC wie der primäre Cluster ausgeführt werden. Wenn der sekundäre Cluster hochgestuft wird, ändert sich der globale Writer-Endpunkt automatisch so, dass er auf diesen Cluster verweist. Mit dem globalen Writer-Endpunkt können Sie zwar vermeiden, dass Sie die Verbindungseinstellungen für Ihre Anwendung ändern müssen, Ihre Anwendungen können aber erst dann auf die IP-Adressen in der VPC der neu hochgestuften, primären AWS-Region zugreifen, wenn Sie ein Netzwerk zwischen den beiden VPCs eingerichtet haben. Unter Amazon VPC-to-Amazon VPC connectivity options finden Sie Informationen zur Bewertung verschiedener Optionen für die Einrichtung dieser Konnektivität.
- Die Aktualisierung des globalen Writer-Endpunkts nach einem globalen Datenbank-Failover oder einer solchen Umstellung kann je nach Dauer der Zwischenspeicherung Ihres Domain Name Service (DNS) lange dauern. Weitere Informationen finden Sie im Administratorhandbuch zu Amazon Aurora-MySQL-Datenbanken. Aurora Global Database gibt ein RDS-Ereignis aus, wenn es eine DNS-Änderung auf dem globalen Writer-Endpunkt feststellt. Sie können das Ereignis verwenden, um Strategien zu entwickeln, die dafür sorgen, dass die Dauer der DNS-Zwischenspeicherung nicht über die Zeit nach der Generierung des Ereignisses hinausgeht. Weitere Informationen finden Sie unter DB-Cluster-Ereignisse.
- Aurora Global Database repliziert Daten asynchron. Die regionsübergreifenden Failover-Methoden können dazu führen, dass einige Schreibtransaktionsdaten vor dem Eintreten des Failovers nicht zur gewählten sekundären Region repliziert wurden. Obwohl Aurora nach besten Kräften versucht, Schreibvorgänge in der ursprünglichen primären AWS-Region zu blockieren, kann ein Failover Split-Brain-Probleme nach sich ziehen. Die Überlegungen zur Minimierung von Datenverlusten und Split-Brain-Risiken gelten auch für Writer-Endpunkte von Aurora Global Database. Weitere

Informationen finden Sie unter Ausführen von verwalteten geplanten Failovers für globale Aurora-Datenbanken.

Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank

Sie können die Anzahl der Endpunkte reduzieren, die Sie für Anwendungen verwalten müssen, die in Ihrer Aurora globalen Datenbank ausgeführt werden, indem Sie Schreibweiterleitung verwenden. Wenn die Schreibweiterleitung aktiviert ist, leiten sekundäre Cluster in einer Aurora Global Database SQL-Anweisungen, die Schreibvorgänge ausführen, an den primären Cluster weiter. Der primäre Cluster aktualisiert die Quelle und überträgt dann die resultierenden Änderungen an alle sekundären AWS-Regionen zurück.

Die Konfiguration der Schreibweiterleitung erspart Ihnen die Implementierung Ihres eigenen Mechanismus zum Senden von Schreibvorgängen von einer sekundären AWS-Region an die primäre Region. Aurora übernimmt die regionsübergreifende Netzwerk-Einrichtung. Aurora überträgt auch den gesamten erforderlichen Sitzungs- und Transaktionskontext für jede Anweisung. Die Daten werden stets zuerst im primären Cluster geändert und anschließend zu den sekundären Clustern in der Aurora globalen Datenbank repliziert. Daher ist der primäre Cluster die autoritative Quelle und enthält stets die jeweils aktuellen Versionen all Ihrer Daten.

Themen

- Verwenden der Schreibweiterleitung in einer globalen Aurora-MySQL-Datenbank
- Verwenden der Schreibweiterleitung in einer globalen Aurora-PostgreSQL-Datenbank

Verwenden der Schreibweiterleitung in einer globalen Aurora-MySQL-Datenbank

Themen

- Verfügbarkeit von Regionen und Versionen der Schreibweiterleitung in Aurora MySQL
- Aktivieren der Schreibweiterleitung in Aurora MySQL
- Überprüfen, ob die Schreibweiterleitung für einen sekundären Cluster in Aurora MySQL aktiviert ist
- Anwendungs- und SQL-Kompatibilität mit Schreibweiterleitung in Aurora MySQL
- Isolation und Konsistenz für die Schreibweiterleitung in Aurora MySQL

- Ausführen von Multipart-Anweisungen mit Schreibweiterleitung in Aurora MySQL
- Transaktionen mit Schreibweiterleitung in Aurora MySQL
- Konfigurationsparameter für die Schreibweiterleitung in Aurora MySQL
- Amazon-CloudWatch-Metriken für die Schreibweiterleitung in Aurora MySQL
- Aurora-MySQL-Statusvariablen für die Schreibweiterleitung

Verfügbarkeit von Regionen und Versionen der Schreibweiterleitung in Aurora MySQL

Die Schreibweiterleitung wird für Aurora MySQL 2.08.1 und höhere Versionen in jeder Region unterstützt, in der Aurora-MySQL-basierte globale Datenbanken verfügbar sind.

Informationen zur Verfügbarkeit von Versionen und Regionen von globalen Aurora-MySQL-Datenbanken finden Sie unter Globale Aurora-Datenbanken mit Aurora MySQL.

Aktivieren der Schreibweiterleitung in Aurora MySQL

Standardmäßig ist die Schreibweiterleitung nicht aktiviert, wenn Sie einen sekundären Cluster zu einer Aurora Global Database hinzufügen.

Um die Schreibweiterleitung über die AWS-Managementkonsole zu aktivieren, wählen Sie das Kontrollkästchen Globale Schreibweiterleitung aktivieren unter Read-Replica-Schreibweiterleitung aus, wenn Sie eine Region für eine globale Datenbank hinzufügen. Ändern Sie den Cluster für einen vorhandenen sekundären Cluster in Globale Schreibweiterleitung aktivieren. Wenn Sie die Schreibweiterleitung ausschalten möchten, deaktivieren Sie beim Hinzufügen der Region oder beim Ändern des sekundären Clusters das Kontrollkästchen Read-Replica-Schreibweiterleitung aktivieren.

Um die Schreibweiterleitung über die AWS CLI zu aktivieren, verwenden Sie die Option `--enable-global-write-forwarding`. Diese Option funktioniert, wenn Sie über den Befehl `create-db-cluster` einen neuen sekundären Cluster erstellen. Sie funktioniert auch, wenn Sie einen vorhandenen sekundären Cluster über den Befehl `modify-db-cluster` ändern. Sie erfordert, dass die globale Datenbank eine Aurora-Version verwendet, die eine Schreibweiterleitung unterstützt. Sie können die Schreibweiterleitung deaktivieren, indem Sie die Option `--no-enable-global-write-forwarding` mit denselben CLI-Befehlen verwenden.

Um die Schreibweiterleitung über die Amazon-RDS-API zu aktivieren, legen Sie den Parameter `EnableGlobalWriteForwarding` auf `true` fest. Dieser Parameter funktioniert, wenn Sie über

die Operation `CreateDBCluster` einen neuen sekundären Cluster erstellen. Sie funktioniert auch, wenn Sie einen vorhandenen sekundären Cluster über die Operation `ModifyDBCluster` ändern. Sie erfordert, dass die globale Datenbank eine Aurora-Version verwendet, die eine Schreibweiterleitung unterstützt. Sie können die Schreibweiterleitung deaktivieren, indem Sie den Parameter `EnableGlobalWriteForwarding` auf `false` festlegen.

> **Note**
>
> Damit eine Datenbanksitzung die Schreibweiterleitung verwenden kann, geben Sie eine Einstellung für den Konfigurationsparameter `aurora_replica_read_consistency` an. Sie führen dies in jeder Sitzung aus, in der die Schreibweiterleitungsfunktion verwendet wird. Informationen zu diesem Parameter finden Sie unter Isolation und Konsistenz für die Schreibweiterleitung in Aurora MySQL.
>
> Die RDS-Proxy-Funktion unterstützt den `SESSION`-Wert für die Variable `aurora_replica_read_consistency` nicht. Durch das Festlegen dieses Werts kann unerwartetes Verhalten auftreten.

Die folgenden CLI-Beispiele zeigen, wie Sie eine Aurora Global Database mit aktivierter oder deaktivierter Schreibweiterleitung einrichten. Die markierten Elemente stellen die Befehle und Optionen dar, die bei der Einrichtung der Infrastruktur für eine Aurora Global Database Datenbank angegeben werden müssen und konsistent sein müssen.

Im folgenden Beispiel werden eine Aurora Global Database, ein primärer Cluster und ein sekundärer Cluster mit aktivierter Schreibweiterleitung erstellt. Geben Sie Ihre eigenen Daten für Benutzername, Passwort und primäre und sekundäre AWS-Regionen ein.

```
# Create overall global database.
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create primary cluster, in the same AWS Region as the global database.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-1 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username user_name --master-user-password password \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
```

```
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create secondary cluster, in a different AWS Region than the global database,
# with write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2 \
  --enable-global-write-forwarding

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2
```

Das folgende Beispiel setzt das vorherige Beispiel fort. In ihm werden ein sekundärer Cluster ohne aktivierte Schreibweiterleitung erstellt und anschließend die Schreibweiterleitung aktiviert. Nach Abschluss dieses Beispiels ist die Schreibweiterleitung für alle sekundären Cluster in der globalen Datenbank aktiviert.

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1
```

```
aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
  --region us-east-2 \
  --enable-global-write-forwarding
```

Überprüfen, ob die Schreibweiterleitung für einen sekundären Cluster in Aurora MySQL aktiviert ist

Um festzustellen, ob Sie die Schreibweiterleitung aus einem sekundären Cluster verwenden können, können Sie prüfen, ob der Cluster das Attribut besitz `"GlobalWriteForwardingStatus": "enabled"`.

In der AWS-Managementkonsole sehen Sie auf der Registerkarte Konfiguration der Detailseite des Clusters den StatusAktiviert für Globale Lesereplikat-Schreibweiterleitung.

Um den Status der globalen Einstellung für die Schreibweiterleitung für alle Cluster anzuzeigen, führen Sie den folgenden AWS CLI-Befehl aus.

Ein sekundärer Cluster zeigt die Werte `"enabled"` oder `"disabled"` an, um anzugeben, ob die Schreibweiterleitung aktiviert oder deaktiviert ist. Der Wert `null` gibt an, dass die Schreibweiterleitung für diesen Cluster nicht verfügbar ist. Entweder ist der Cluster nicht Teil einer globalen Datenbank oder ist der primäre Cluster und nicht kein sekundärer Cluster. Der Wert kann auch `"enabling"` oder `"disabling"` sein, wenn die Schreibweiterleitung gerade aktiviert oder deaktiviert wird.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
```

```
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

Führen Sie den folgenden Befehl aus, um alle sekundären Cluster zu suchen, für die eine globale Schreibweiterleitung aktiviert ist. Dieser Befehl gibt auch den Reader-Endpunkt des Clusters aus. Sie verwenden den Reader-Endpunkt des sekundären Clusters, wenn Sie die Schreibweiterleitung vom sekundären zum primären Cluster in Ihrer globalen Aurora-Datenbank verwenden.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Anwendungs- und SQL-Kompatibilität mit Schreibweiterleitung in Aurora MySQL

Sie können die folgenden Arten von SQL-Anweisungen mit Schreibweiterleitung verwenden:

- Data Manipulation Language (DM)-Anweisungen wie `INSERT`, wie`DELETE` und `UPDATE`.
  Es gibt einige Einschränkungen für die Eigenschaften dieser Anweisungen, die Sie bei der Schreibweiterleitung verwenden können, wie im Folgenden beschrieben.

- `SELECT ... LOCK IN SHARE MODE`- und `SELECT FOR UPDATE`-Anweisungen.
- `PREPARE`- und `EXECUTE`-Anweisungen.

Bestimmte Anweisungen sind nicht zulässig oder können veraltete Ergebnisse erzeugen, wenn Sie sie in einer globalen Datenbank mit Schreibweiterleitung verwenden. Daher ist die Einstellung `EnableGlobalWriteForwarding` für sekundäre Cluster standardmäßig deaktiviert. Stellen Sie vor einer Aktivierung sicher, dass der Anwendungscode von keiner dieser Einschränkungen betroffen ist.

Die folgenden Einschränkungen gelten für die SQL-Anweisungen, die Sie für die Schreibweiterleitung verwenden. In einigen Fällen können Sie die Anweisungen auf sekundären Clustern verwenden, bei denen eine Schreibweiterleitung auf Cluster-Ebene aktiviert ist. Dieser Ansatz funktioniert, wenn die Schreibweiterleitung nicht innerhalb der Sitzung durch den Konfigurationsparameter `aurora_replica_read_consistency` aktiviert wird. Der Versuch, eine Anweisung zu verwenden, wenn sie aufgrund der Schreibweiterleitung nicht zulässig ist, verursacht eine Fehlermeldung im folgendem Format.

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

Data Definition Language (DDL)

Stellen Sie eine Verbindung zum primären Cluster her, um DDL-Anweisungen auszuführen. Sie können sie nicht von Reader-DB-Instances aus ausführen.

Aktualisieren einer permanenten Tabelle mit Daten aus einer temporären Tabelle

Sie können in sekundären Clustern mit aktivierter Schreibweiterleitung temporäre Tabellen verwenden. Sie können jedoch keine DML-Anweisung verwenden, um eine permanente Tabelle zu ändern, wenn sich die Anweisung auf eine temporäre Tabelle bezieht. Beispielsweise können Sie keine `INSERT ... SELECT`-Anweisung verwenden, die die Daten aus einer temporären Tabelle übernimmt. Die temporäre Tabelle ist auf dem sekundären Cluster vorhanden und nicht verfügbar, wenn die Anweisung auf dem primären Cluster ausgeführt wird.

XA-Transaktionen

Sie können die folgenden Anweisungen nicht in einem sekundären Cluster verwenden, wenn die Schreibweiterleitung innerhalb der Sitzung aktiviert wird. Sie können diese Anweisungen in sekundären Clustern verwenden, bei denen die Schreibweiterleitung nicht aktiviert ist, oder innerhalb von Sitzungen mit leerer Einstellung `aurora_replica_read_consistency`. Bevor

Sie die Schreibweiterleitung innerhalb einer Sitzung aktivieren, müssen Sie überprüfen, ob Ihr Code diese Anweisungen verwendet.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

LOAD-Anweisungen für permanente Tabellen

Sie können die folgenden Anweisungen nicht in einem sekundären Cluster mit aktivierter Schreibweiterleitung verwenden.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
        LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

Sie können in einem sekundären Cluster Daten in eine temporäre Tabelle laden. Sie dürfen jedoch alle LOAD-Anweisungen, die sich auf permanente Tabellen beziehen, nur im primären Cluster ausführen.

Plugin-Anweisungen

Sie können die folgenden Anweisungen nicht in einem sekundären Cluster mit aktivierter Schreibweiterleitung verwenden.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT-Anweisungen

Sie können die folgenden Anweisungen nicht in einem sekundären Cluster verwenden, wenn die Schreibweiterleitung innerhalb der Sitzung aktiviert wird. Sie können diese Anweisungen in sekundären Clustern ohne aktivierte Schreibweiterleitung oder in Sitzungen mit leerer Einstellung `aurora_replica_read_consistency` verwenden. Sie müssen überprüfen, ob Ihr Code diese Anweisungen verwendet, bevor Sie die Schreibweiterleitung innerhalb einer Sitzung aktivieren.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Isolation und Konsistenz für die Schreibweiterleitung in Aurora MySQL

In Sitzungen, die Schreibweiterleitung verwenden, können Sie nur die Isolationsstufe `REPEATABLE READ` verwenden. Obwohl Sie die Isolationsstufe `READ COMMITTED` auch mit schreibgeschützten Clustern in sekundären AWS-Regionen verwenden können, funktioniert diese Isolationsstufe nicht mit Schreibweiterleitung. Weitere Informationen zu den Isolationsstufen `REPEATABLE READ` und `READ COMMITTED` finden Sie unter Aurora MySQL-Isolierungsstufen.

Sie können den Grad der Lesekonsistenz in einem sekundären Cluster steuern. Die Lesekonsistenzstufe legt fest, wie viel Wartezeit der sekundäre Cluster vor jeder Leseoperation ausführt, um sicherzustellen, dass einige oder alle Änderungen aus dem primären Cluster repliziert werden. Sie können die Lesekonsistenzstufe anpassen, um sicherzustellen, dass alle aus Ihrer Sitzung weitergeleiteten Schreiboperationen im sekundären Cluster vor nachfolgenden Abfragen angezeigt werden. Sie können diese Einstellung auch verwenden, um sicherzustellen, dass Abfragen auf dem sekundären Cluster stets die aktuellsten Updates des primären Clusters angezeigt werden. Dies gilt auch für diejenigen, die von anderen Sitzungen oder anderen Clustern übermittelt wurden. Um diese Art von Verhalten für Ihre Anwendung anzugeben, wählen Sie einen Wert für den Sitzungsebenen-Parameter au `aurora_replica_read_consistency`.

> **Important**
>
> Legen Sie immer den Parameter `aurora_replica_read_consistency` für jede Sitzung fest, für die Sie Schreibvorgänge weiterleiten möchten. Andernfalls aktiviert Aurora die Schreibweiterleitung für diese Sitzung nicht. Dieser Parameter hat standardmäßig einen leeren Wert, wählen Sie daher einen bestimmten Wert, wenn Sie diesen Parameter verwenden. Der Parameter `aurora_replica_read_consistency` wirkt sich nur auf sekundäre Cluster aus, in denen die Schreibweiterleitung aktiviert ist.
> Verwenden Sie für Aurora-MySQL-Version 2 und Version 3 unter 3.04 `aurora_replica_read_consistency` als Sitzungsvariable. Für Aurora-MySQL-Version 3.04 und höher können Sie `aurora_replica_read_consistency` entweder als Sitzungsvariable oder als DB-Cluster-Parameter verwenden.

Für den Parameter `aurora_replica_read_consistency` können Sie die Werte `EVENTUAL`, `SESSION` und `GLOBAL` angeben.

Wenn Sie das Konsistenzniveau erhöhen, verbringt Ihre Anwendung mehr Zeit mit dem Warten auf die Propagierung von Änderungen zwischen AWS-Regionen. Sie können das Verhältnis zwischen

schneller Reaktionszeit und der Gewährleistung festlegen, dass vor der Ausführung Ihrer Abfragen Änderungen an anderen Speicherorten vollständig verfügbar sind.

Wenn die Lesekonsistenz auf `EVENTUAL` festgelegt wird, werden Abfragen in einer sekundären AWS-Region, die Schreibweiterleitung verwendet, möglicherweise Daten angezeigt, die aufgrund von Replikationsverzögerungen leicht veraltet sind. Ergebnisse von Schreiboperationen in derselben Sitzung werden erst angezeigt, wenn die Schreiboperation in der primären Region ausgeführt und zur aktuellen Region repliziert wird. Die Abfrage wartet nicht, bis die aktualisierten Ergebnisse verfügbar sind. Daher könnte sie die älteren Daten oder die aktualisierten Daten abrufen, abhängig vom Zeitpunkt der Anweisungen und der Größe der Replikationsverzögerung.

Wenn die Lesekonsistenz auf `SESSION` festgelegt ist, werden allen Abfragen in einer sekundären AWS-Region, die Schreibweiterleitung verwendet, die Ergebnisse aller in dieser Sitzung ausgeführten Änderungen angezeigt. Die Änderungen werden unabhängig davon angezeigt, ob die Transaktion übergeben wurde. Wenn notwendig, wartet die Abfrage auf die Replikation der Ergebnisse weitergeleiteter Schreiboperationen zur aktuellen Region. Sie wartet nicht auf aktualisierte Ergebnisse aus Schreiboperationen, die in anderen Regionen oder in anderen Sitzungen innerhalb der aktuellen Region ausgeführt werden.

Wenn die Lesekonsistenz auf `GLOBAL` festgelegt ist, werden einer Sitzung in einer sekundären AWS-Region Änderungen angezeigt, die von dieser Sitzung ausgeführt wurden. Außerdem werden ihr alle übergebenen Änderungen sowohl aus der primären AWS-Region als auch aus anderen sekundären AWS-Regionen angezeigt. Jede Abfrage kann für einen bestimmten Zeitraum warten, abhängig von der Sitzungsverzögerung. Die Abfrage wird fortgesetzt, wenn der sekundäre Cluster mit allen übergebenen Daten aus dem primären Cluster mit Stand zu dem Zeitpunkt, an dem die Abfrage gestartet wurde, aktualisiert ist.

Weitere Informationen zu allen mit der Schreibweiterleitung verbundenen Parametern finden Sie unter Konfigurationsparameter für die Schreibweiterleitung in Aurora MySQL.

Beispiele für die Verwendung der Schreibweiterleitung

In diesen Beispielen wird `aurora_replica_read_consistency` als Sitzungsvariable verwendet. Für Aurora-MySQL-Version 3.04 und höher können Sie `aurora_replica_read_consistency` entweder als Sitzungsvariable oder als DB-Cluster-Parameter verwenden.

Im folgenden Beispiel befindet sich der primäre Cluster in der Region US East (N. Virginia). Der sekundäre Cluster befindet sich in der Region USA Ost (Ohio). Das Beispiel zeigt die Auswirkungen

ausgeführter INSERT-Anweisungen, gefolgt von SELECT-Anweisungen. Abhängig vom Wert der Einstellung aurora_replica_read_consistency können sich die Ergebnisse abhängig vom Zeitpunkt der Anweisungen unterscheiden. Um eine höhere Konsistenz zu erreichen, können Sie kurz warten, bevor Sie die Anweisung SELECT ausführen. Oder Aurora kann automatisch warten, bis die Ergebnisse fertig repliziert sind, bevor mit fortgefahren wir SELECT.

In diesem Beispiel gibt es eine Lesekonsistenzeinstellung von eventual. Das Ausführen der Anweisung INSERT unmittelbar gefolgt von der Anweisung SELECT gibt immer noch den Wert von COUNT(*) zurück. Dieser Wert gibt die Anzahl der Zeilen an, bevor die neue Zeile eingefügt wird. Wenn Sie kurze Zeit später SELECT erneut ausführen, wird die aktualisierte Zeilenzahl zurückgegeben. Die SELECT-Anweisungen haben keine Wartezeit.

```
mysql> set aurora_replica_read_consistency = 'eventual';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Bei Festlegung der Lesekonsistenzeinstellung auf session wartet eine SELECT-Anweisung, die unmittelbar nach einer INSERT-Anweisung ausgeführt wird, bis die Änderungen aus der INSERT-Anweisung angezeigt werden. Nachfolgende SELECT-Anweisungen haben keine Wartezeit.

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
```

```
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

Wenn die Lesekonsistenzeinstellung weiter auf `session` festgelegt ist, wird durch die Einführung einer kurzen Wartezeit nach der Ausführung einer `INSERT`-Anweisung die aktualisierte Zeilenzahl zum Zeitpunkt der nächsten Ausführung der `SELECT`-Anweisung verfügbar.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Wenn die Leseksistenzeinstellung auf `global` festgelegt ist, wartet jede `SELECT`-Anweisung, um sicherzustellen, dass alle Datenänderungen mit Stand zur Startzeit der Anweisung angezeigt werden,

bevor die Abfrage ausgeführt wird. Die Wartezeit für die einzelnen `SELECT`-Anweisungen ist von der Replikationsverzögerung zwischen dem primären und den sekundären Clustern abhängig.

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

Ausführen von Multipart-Anweisungen mit Schreibweiterleitung in Aurora MySQL

Eine DML-Anweisung kann aus mehreren Teilen bestehen, z. B. einer `INSERT ... SELECT`-Anweisung oder einer `DELETE ... WHERE`-Anweisung. In diesem Fall wird die gesamte Anweisung an den primären Cluster weitergeleitet und dort ausgeführt.

Transaktionen mit Schreibweiterleitung in Aurora MySQL

Ob die Transaktion an den primären Cluster weitergeleitet wird, ist vom Zugriffsmodus der Transaktion abhängig. Sie können den Zugriffsmodus für die Transaktion durch Verwendung der Anweisungen `SET TRANSACTION` oder `START TRANSACTION` angeben. Sie können den Transaktionszugriffsmodus auch angeben, indem Sie den Wert der Sitzungsvariablen transaction_read_only ändern. Sie können diesen Sitzungswert nur ändern, wenn Sie mit einem DB-Cluster verbunden sind, für den die Schreibweiterleitung aktiviert ist.

Wenn eine Transaktion mit langer Laufzeit für einen beträchtlichen Zeitraum keine Anweisung ausgibt, kann sie Leerlaufzeitüberschreitung überschreiten. Dieser Zeitraum hat einen Standardwert von einer Minute. Sie können den Wert auf bis zu einem Tag erhöhen. Eine Transaktion, die die Leerlaufzeitüberschreitung überschreitet, wird vom primären Cluster abgebrochen. Die nächste nachfolgende Anweisung, die Sie übermitteln, empfängt einen Zeitüberschreitungsfehler. In diesem Fall rollt Aurora die Transaktion zurück.

Diese Art von Fehler kann in anderen Fällen auftreten, wenn die Schreibweiterleitung nicht mehr verfügbar ist. Beispielsweise bricht Aurora alle Transaktionen ab, die Schreibweiterleitung verwenden, wenn Sie den primären Cluster neu starten oder die Konfigurationseinstellung für die Schreibweiterleitung deaktivieren.

Konfigurationsparameter für die Schreibweiterleitung in Aurora MySQL

Die Aurora-Cluster-Parametergruppen enthalten Einstellungen für die Schreibweiterleitung. Da es sich um Cluster-Parameter handelt, haben alle DB-Instances in allen Clustern die gleichen Werte für diese Variablen. Details zu diesen Parametern sind in der folgenden Tabelle zusammengefasst, mit Nutzungshinweisen unterhalb der Tabelle.

Name	Scope	Typ	Standardwert	Zulässige Werte
`aurora_fwd_master_idle_timeout` (Aurora-MySQL-Version 2)	Global	Ganzzahl ohne Vorzeichen	60	1 – 86.400
`aurora_fwd_master_max_connections_pct` (Aurora-MySQL-Version 2)	Global	Lange Ganzzahl ohne Vorzeichen	10	0 – 90
`aurora_fwd_writer_idle_timeout` (Aurora-MySQL-Version 3)	Global	Ganzzahl ohne Vorzeichen	60	1 – 86.400
`aurora_fwd_writer_max_connections_pct` (Aurora-MySQL-Version 3)	Global	Lange Ganzzahl	10	0 – 90

Name	Scope	Typ	Standardwert	Zulässige Werte
		ohne Vorzeichen		
`aurora_replica_read_consistency`	Sitzung für Version 2 und Version 3 niedriger als 3.04, global für Version 3.04 und höher	Enum	" (null)	`EVENTUAL`, `SESSION`, `GLOBAL`

Um eingehende Schreibanforderungen aus sekundären Clustern zu steuern, verwenden Sie die folgenden Einstellungen auf dem primären Cluster:

- `aurora_fwd_master_idle_timeout`, `aurora_fwd_writer_idle_timeout`: Die Anzahl der Sekunden, die der primäre Cluster auf Aktivität für eine Verbindung wartet, die aus einem sekundären Cluster weitergeleitet wird, bevor er sie schließt. Wenn die Sitzung über diesen Zeitraum hinaus im Leerlauf ist, bricht Aurora die Sitzung ab.
- `aurora_fwd_master_max_connections_pct`, `aurora_fwd_writer_max_connections_pct`: Die obere Grenze für Datenbankverbindungen, die in einer Writer-DB-Instance für die Verarbeitung von Abfragen verwendet werden können, die von Lesern weitergeleitet werden. Sie wird als Prozentsatz der Einstellung `max_connections` für die Writer-DB-Instance im primären Cluster ausgedrückt. Wenn beispielsweise `max_connections` 800 ist und `aurora_fwd_master_max_connections_pct` oder `aurora_fwd_writer_max_connections_pct` 10 ist, lässt der Writer maximal 80 weitergeleitete Sitzungen gleichzeitig zu. Diese Verbindungen stammen aus demselben, von der Einstellung `max_connections` verwalteten Verbindungspool.

  Diese Einstellung gilt nur für den primären Cluster, wenn für mindestens einen sekundären Cluster die Schreibweiterleitung aktiviert ist. Wenn Sie den Wert verringern, sind vorhandene Verbindungen nicht betroffen. Aurora berücksichtigt den neuen Wert der Einstellung, wenn versucht wird, eine neue Verbindung aus einem sekundären Cluster zu erstellen. Der Standardwert

ist 10, was 10 % des Werts für `max_connections` entspricht. Wenn Sie die Abfrageweiterleitung für einen der sekundären Cluster aktivieren, muss diese Einstellung einen Wert ungleich Null haben, damit Schreiboperationen aus sekundären Clustern erfolgreich ausgeführt werden können. Wenn der Wert null ist, empfangen die Schreiboperationen den Fehlercode `ER_CON_COUNT_ERROR` mit der Meldung `Not enough connections on writer to handle your request`.

Der `aurora_replica_read_consistency`-Parameter aktiviert die Schreibweiterleitung. Sie benutzen ihn in jeder Sitzung. Sie können `EVENTUAL`, `SESSION` oder `GLOBAL` als Lesekonsistenzstufe angeben. Weitere Informationen über Konsistenzebenen finden Sie unter Isolation und Konsistenz für die Schreibweiterleitung in Aurora MySQL. Für diesen Parameter gelten die folgenden Regeln:

- Der Standardwert ist '' (leer).
- Die Schreibweiterleitung ist in einer Sitzung nur verfügbar, wenn `aurora_replica_read_consistency` auf `EVENTUAL`, `SESSION` oder `GLOBAL` festgelegt ist. Dieser Parameter ist nur in Reader-Instances sekundärer Cluster relevant, für die Schreibweiterleitung aktiviert ist und die sich in einer Aurora Global Database befinden.
- Sie können diese Variable (wenn leer) oder nicht eingestellt (wenn sie bereits festgelegt ist) nicht innerhalb einer Multistatement-Transaktion setzen. Sie können sie jedoch während einer solchen Transaktion von einem gültigen Wert (`EVENTUAL`, `SESSION` oder `GLOBAL`) in einen anderen gültigen Wert (`EVENTUAL`, `SESSION` oder `GLOBAL`) ändern.
- Die Variable darf nicht `SET` sein, wenn die Schreibweiterleitung auf dem sekundären Cluster nicht aktiviert ist.

Amazon-CloudWatch-Metriken für die Schreibweiterleitung in Aurora MySQL

Die folgenden Amazon CloudWatch-Metriken gelten für den primären Cluster, wenn Sie die Schreibweiterleitung auf einem oder mehreren sekundären Clustern verwenden. Diese Metriken werden alle auf der Writer-DB-Instance im primären Cluster gemessen.

CloudWatch-Metrik	Einheit	Beschreibung
`AuroraDMLRejectedMasterFull`	Anzahl	Die Anzahl der weitergeleiteten Abfragen, die abgelehnt wurden, weil die

CloudWatch-Metrik	Einheit	Beschreibung
		Sitzung auf der Writer-DB-Instance ausgelastet ist Für Aurora-MySQL-Version 2.
`AuroraDMLRejectedWriterFull`	Anzahl	Die Anzahl der weitergeleiteten Abfragen, die abgelehnt wurden, weil die Sitzung auf der Writer-DB-Instance ausgelastet ist Für Aurora-MySQL-Version 3.
`ForwardingMasterDMLLatency`	Millisekunden	Durchschnittliche Zeit für die Verarbeitung jeder weitergeleiteten DML-Anweisung auf der Writer-DB-Instance. Nicht enthalten ist die Zeit, die der sekundäre Cluster benötigt, um die Schreibanforderung weiterzuleiten, oder die Zeit, um Änderungen zurück an den sekundären Cluster zu replizieren. Für Aurora-MySQL-Version 2.
`ForwardingMasterDMLThroughput`	Anzahl pro Sekunde	Anzahl der weitergeleiteten DML-Anweisungen, die pro Sekunde von dieser Writer-DB-Instance verarbeitet werden. Für Aurora-MySQL-Version 2.

CloudWatch-Metrik	Einheit	Beschreibung
`ForwardingMasterOpenSessions`	Anzahl	Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance. Für Aurora-MySQL-Version 2.
`ForwardingWriterDMLLatency`	Millisekunden	Durchschnittliche Zeit für die Verarbeitung jeder weitergeleiteten DML-Anweisung auf der Writer-DB-Instance. Nicht enthalten ist die Zeit, die der sekundäre Cluster benötigt, um die Schreibanforderung weiterzuleiten, oder die Zeit, um Änderungen zurück an den sekundären Cluster zu replizieren. Für Aurora-MySQL-Version 3.
`ForwardingWriterDMLThroughput`	Anzahl pro Sekunde	Anzahl der weitergeleiteten DML-Anweisungen, die pro Sekunde von dieser Writer-DB-Instance verarbeitet werden. Für Aurora-MySQL-Version 3.
`ForwardingWriterOpenSessions`	Anzahl	Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance. Für Aurora-MySQL-Version 3.

Die folgenden CloudWatch-Metriken gelten für jeden sekundären Cluster. Diese Metriken werden auf jeder Reader-DB-Instance in einem sekundären Cluster mit aktivierter Schreibweiterleitung gemessen.

CloudWatch-Metrik	Einheit	Beschreibung
`ForwardingReplicaDMLLatency`	Millisekunden	Durchschnittliche Reaktionszeit weitergeleiteter DML-Anweisungen auf Replikaten.
`ForwardingReplicaDMLThroughput`	Anzahl pro Sekunde	Anzahl der weitergeleiteten DML-Anweisungen, die pro Sekunde verarbeitet werden.
`ForwardingReplicaOpenSessions`	Anzahl	Die Anzahl der Sitzungen, die die Schreibweiterleitung für eine Reader-DB-Instance verwenden.
`ForwardingReplicaReadWaitLatency`	Millisekunden	Durchschnittliche Wartezeit, die eine `SELECT`-Anweisung in einer Reader-DB-Instance darauf wartet, zum primären Cluster aufzuholen. Der Grad, in dem die Reader-DB-Instance vor der Verarbeitung einer Abfrage wartet, ist von der Einstellung `aurora_replica_read_consistency` abhängig.
`ForwardingReplicaReadWaitThroughput`	Anzahl pro Sekunde	Gesamtzahl der pro Sekunde in allen Sitzungen verarbeiteten `SELECT`-Anweisungen, die Schreibvorgänge weiterleiten.
`ForwardingReplicaSelectLatency`	Millisekunden	Weitergeleitete `SELECT`-Latenz, Durchschnitt über alle weitergeleiteten `SELECT`-

CloudWatch-Metrik	Einheit	Beschreibung
		Anweisungen innerhalb des Überwachungszeitraums.
`ForwardingReplicaSelectThroughput`	Anzahl pro Sekunde	Weitergeleiteter `SELECT`-Durchsatz, pro Sekunde, als Durchschnitt innerhalb des Überwachungszeitraums.

Aurora-MySQL-Statusvariablen für die Schreibweiterleitung

Die folgenden Aurora-MySQL-Statusvariablen gelten für den primären Cluster, wenn Sie die Schreibweiterleitung auf einem oder mehreren sekundären Clustern verwenden. Diese Metriken werden alle auf der Writer-DB-Instance im primären Cluster gemessen.

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
`Aurora_fwd_master_dml_stmt_count`	Anzahl	Gesamtzahl der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Für Aurora-MySQL-Version 2.
`Aurora_fwd_master_dml_stmt_duration`	Mikrosekunden	Gesamtdauer der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Für Aurora-MySQL-Version 2.
`Aurora_fwd_master_open_sessions`	Anzahl	Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance. Für Aurora-MySQL-Version 2.

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
`Aurora_fwd_master_select_stmt_count`	Anzahl	Gesamtzahl der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Für Aurora-MySQL-Version 2.
`Aurora_fwd_master_select_stmt_duration`	Mikrosekunden	Gesamtdauer der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Für Aurora-MySQL-Version 2.
`Aurora_fwd_writer_dml_stmt_count`	Anzahl	Gesamtzahl der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Für Aurora-MySQL-Version 3.
`Aurora_fwd_writer_dml_stmt_duration`	Mikrosekunden	Gesamtdauer der DML-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden.
`Aurora_fwd_writer_open_sessions`	Anzahl	Anzahl der weitergeleiteten Sitzungen auf der Writer-DB-Instance. Für Aurora-MySQL-Version 3.
`Aurora_fwd_writer_select_stmt_count`	Anzahl	Gesamtzahl der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Für Aurora-MySQL-Version 3.

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
`Aurora_fwd_writer_select_stmt_duration`	Mikrosekunden	Gesamtdauer der `SELECT`-Anweisungen, die an diese Writer-DB-Instance weitergeleitet werden. Für Aurora-MySQL-Version 3.

Die folgenden Aurora-MySQL-Statusvariablen gelten für jeden sekundären Cluster. Diese Metriken werden auf jeder Reader-DB-Instance in einem sekundären Cluster mit aktivierter Schreibweiterleitung gemessen.

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
`Aurora_fwd_replica_dml_stmt_count`	Anzahl	Gesamtzahl der DML-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.
`Aurora_fwd_replica_dml_stmt_duration`	Mikrosekunden	Gesamtdauer aller DML-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.
`Aurora_fwd_replica_errors_session_limit`	Anzahl	Anzahl der Sitzungen, die vom primären Cluster aufgrund einer der folgenden Fehlerbedingungen zurückgewiesen wurden: • Writer voll • Es sind zu viele weitergeleitete Anweisungen in Bearbeitung.

Aurora-MySQL-Statusvariable	Einheit	Beschreibung
`Aurora_fwd_replica_open_sessions`	Anzahl	Die Anzahl der Sitzungen, die die Schreibweiterleitung für eine Reader-DB-Instance verwenden.
`Aurora_fwd_replica_read_wait_count`	Anzahl	Gesamtzahl der Lesen-Nach-Schreiben-Wartezeiten auf dieser Reader-DB-Instance.
`Aurora_fwd_replica_read_wait_duration`	Mikrosekunden	Gesamtdauer der Wartezeiten aufgrund der Lesekonsistenzeinstellung auf dieser Reader-DB-Instance.
`Aurora_fwd_replica_select_stmt_count`	Anzahl	Gesamtzahl der `SELECT`-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.
`Aurora_fwd_replica_select_stmt_duration`	Mikrosekunden	Gesamtdauer der `SELECT`-Anweisungen, die aus dieser Reader-DB-Instance weitergeleitet werden.

Verwenden der Schreibweiterleitung in einer globalen Aurora-PostgreSQL-Datenbank

Themen

- Region und Versionsverfügbarkeit der Schreibweiterleitung in Aurora PostgreSQL
- Aktivieren der Schreibweiterleitung in Aurora PostgreSQL
- Überprüfen auf die Aktivierung der Schreibweiterleitung für einen sekundären Cluster in Ausrora PostgreSQL
- Anwendungs- und SQL-Kompatibilität mit Schreibweiterleitung in Aurora PostgreSQL
- Isolation und Konsistenz für die Schreibweiterleitung in Aurora PostgreSQL

- Transaktionszugriffsmodi mit Schreibweiterleitung
- Ausführen von Multipart-Anweisungen mit Schreibweiterleitung in Aurora PostgreSQL
- Konfigurationsparameter für die Schreibweiterleitung in Aurora PostgreSQL
- Amazon-CloudWatch-Metriken für die Schreibweiterleitung in Aurora PostgreSQL
- Warte-Ereignisse für die Schreibweiterleitung in Aurora PostgreSQL

Region und Versionsverfügbarkeit der Schreibweiterleitung in Aurora PostgreSQL

Bei Aurora-PostgreSQL-Version 16 und höheren Hauptversionen wird die globale Schreibweiterleitung in allen Nebenversionen unterstützt. In früheren Versionen von Aurora PostgreSQL wird die globale Schreibweiterleitung ab Version 15.4 und höheren Nebenversionen sowie ab Version 14.9 und höheren Nebenversionen unterstützt. Die Schreibweiterleitung ist in jeder AWS-Region verfügbar, in der auf Aurora PostgreSQL basierende globale Datenbanken verfügbar sind.

Weitere Informationen zur Verfügbarkeit von Versionen und Regionen im Zusammenhang mit globalen Aurora-PostgreSQL-Datenbanken finden Sie unter Globale Aurora-Datenbanken unterstützen Aurora PostgreSQL.

Aktivieren der Schreibweiterleitung in Aurora PostgreSQL

Standardmäßig ist die Schreibweiterleitung nicht aktiviert, wenn Sie einen sekundären Cluster zu einer Aurora Global Database hinzufügen. Sie können die Schreibweiterleitung für Ihren sekundären DB-Cluster aktivieren, während Sie ihn erstellen oder jederzeit danach. Bei Bedarf können Sie es später deaktivieren. Das Aktivieren oder Deaktivieren der Schreibweiterleitung führt nicht zu Ausfallzeiten oder einem Neustart.

> **Note**
>
> Sie können die lokale Schreibweiterleitung für Anwendungen verwenden, bei denen gelegentlich Schreibvorgänge auftreten und die eine Lesen-nach-Schreiben-Konsistenz erfordern. Dies ist die Fähigkeit, den letzten Schreibvorgang in einer Transaktion zu lesen.

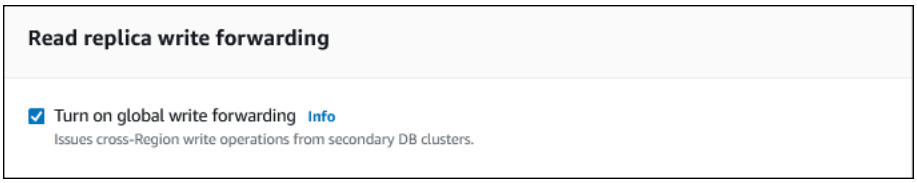
Konsole

In der Konsole können Sie die Schreibweiterleitung aktivieren oder deaktivieren, wenn Sie einen sekundären DB-Cluster erstellen oder ändern.

Aktivierung oder Deaktivierung der Schreibweiterleitung bei der Erstellung eines sekundären DB-Clusters

Wenn Sie einen neuen sekundären DB-Cluster erstellen, aktivieren Sie die Schreibweiterleitung, indem Sie unter Lesereplikat-Schreibweiterleitung das Kontrollkästchen Globale Schreibweiterleitung aktivieren markieren. Oder entfernen Sie die Markierung in dem Kontrollkästchen, um das Feature zu deaktivieren. Befolgen Sie die Anweisungen für Ihre DB-Engine unter Erstellen eines Amazon Aurora-DB Clusters, um einen sekundärem DB-Cluster zu erstellen.

Der folgende Screenshot zeigt den Abschnitt Lesereplikat-Schreibweiterleitung, bei dem das Kontrollkästchen Globale Schreibweiterleitung aktivieren markiert ist.



Aktivierung oder Deaktivierung der Schreibweiterleitung bei der Änderung eines sekundären DB-Clusters

In der Konsole können Sie einen sekundären DB-Cluster ändern, um die Schreibweiterleitung zu aktivieren oder zu deaktivieren.

So aktivieren oder deaktivieren Sie die Schreibweiterleitung für einen vorhandenen sekundären DB-Cluster mit der Konsole

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus.
3. Wählen Sie den sekundären DB-Cluster aus und klicken Sie dann auf Ändern.
4. Markieren oder löschen Sie im Abschnitt Lesereplikat-Schreibweiterleitung das Kontrollkästchen Globale Schreibweiterleitung aktivieren.
5. Klicken Sie auf Weiter.

6. Wählen Sie für Einplanung von Änderungen die Option Sofort anwenden aus. Wenn Sie Während des nächsten geplanten Wartungsfensters anwenden wählen, ignoriert Aurora diese Einstellung und aktiviert die Schreibweiterleitung sofort.
7. Wählen Sie Cluster bearbeiten aus.

AWS CLI

Um die Schreibweiterleitung über die AWS CLI zu aktivieren, verwenden Sie die Option `--enable-global-write-forwarding`. Diese Option funktioniert, wenn Sie über den Befehl create-db-cluster einen neuen sekundären Cluster erstellen. Sie funktioniert auch, wenn Sie einen vorhandenen sekundären Cluster über den Befehl modify-db-cluster ändern. Sie erfordert, dass die globale Datenbank eine Aurora-Version verwendet, die eine Schreibweiterleitung unterstützt. Sie können die Schreibweiterleitung deaktivieren, indem Sie die Option `--no-enable-global-write-forwarding` mit denselben CLI-Befehlen verwenden.

In den folgenden Verfahren wird beschrieben, wie Sie die Schreibweiterleitung für einen sekundären DB-Cluster in Ihrem globalen Cluster mithilfe von AWS CLI aktivieren oder deaktivieren.

So aktivieren oder deaktivieren Sie die Schreibweiterleitung für einen vorhandenen sekundären DB-Cluster

- Rufen Sie den AWS CLI-Befehl modify-db-cluster auf und geben Sie folgende Werte an:
  - `--db-cluster-identifier` – der Name des DB-Clusters.
  - `--enable-global-write-forwarding` zum Aktivieren oder `--no-enable-global-write-forwarding` zum Deaktivieren.

  Das folgende Beispiel aktiviert die Schreibweiterleitung für den DB-Cluster `sample-secondary-db-cluster`.

  Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-secondary-db-cluster \
    --enable-global-write-forwarding
```

  Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-secondary-db-cluster ^
    --enable-global-write-forwarding
```

RDS-API

Um die Schreibweiterleitung über die Amazon-RDS-API zu aktivieren, legen Sie den Parameter `EnableGlobalWriteForwarding` auf `true` fest. Dieser Parameter funktioniert, wenn Sie über die Operation CreateDBCluster einen neuen sekundären Cluster erstellen. Er funktioniert auch, wenn Sie einen vorhandenen sekundären Cluster über die Operation ModifyDBCluster ändern. Sie erfordert, dass die globale Datenbank eine Aurora-Version verwendet, die eine Schreibweiterleitung unterstützt. Sie können die Schreibweiterleitung deaktivieren, indem Sie den Parameter `EnableGlobalWriteForwarding` auf `false` festlegen.

Überprüfen auf die Aktivierung der Schreibweiterleitung für einen sekundären Cluster in Ausrora PostgreSQL

Um festzustellen, ob Sie die Schreibweiterleitung aus einem sekundären Cluster verwenden können, können Sie prüfen, ob der Cluster das Attribut besitz `"GlobalWriteForwardingStatus": "enabled"`.

In der AWS-Managementkonsole wird Ihnen `Read replica write forwarding` auf der Registerkarte Configuration (Konfiguration) der Detailseite für den Cluster angezeigt. Um den Status der globalen Einstellung für die Schreibweiterleitung für alle Cluster anzuzeigen, führen Sie den folgenden AWS CLI-Befehl aus.

Ein sekundärer Cluster zeigt die Werte `"enabled"` oder `"disabled"` an, um anzugeben, ob die Schreibweiterleitung aktiviert oder deaktiviert ist. Der Wert `null` gibt an, dass die Schreibweiterleitung für diesen Cluster nicht verfügbar ist. Entweder ist der Cluster nicht Teil einer globalen Datenbank oder ist der primäre Cluster und nicht kein sekundärer Cluster. Der Wert kann auch `"enabling"` oder `"disabling"` sein, wenn die Schreibweiterleitung gerade aktiviert oder deaktiviert wird.

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
[
```

```
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

Führen Sie den folgenden Befehl aus, um alle sekundären Cluster zu suchen, für die eine globale Schreibweiterleitung aktiviert ist. Dieser Befehl gibt auch den Reader-Endpunkt des Clusters aus. Sie verwenden den Reader-Endpunkt des sekundären Clusters, wenn Sie die Schreibweiterleitung vom sekundären zum primären Cluster in Ihrer globalen Aurora-Datenbank verwenden.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Anwendungs- und SQL-Kompatibilität mit Schreibweiterleitung in Aurora PostgreSQL

Bestimmte Anweisungen sind nicht zulässig oder können veraltete Ergebnisse erzeugen, wenn Sie sie in einer globalen Datenbank mit Schreibweiterleitung verwenden. Darüber hinaus werden benutzerdefinierte Funktionen und benutzerdefinierte Prozeduren nicht unterstützt. Daher ist die Einstellung `EnableGlobalWriteForwarding` für sekundäre Cluster standardmäßig deaktiviert. Stellen Sie vor einer Aktivierung sicher, dass der Anwendungscode von keiner dieser Einschränkungen betroffen ist.

Sie können die folgenden Arten von SQL-Anweisungen mit Schreibweiterleitung verwenden:

- Data Manipulation Language (DML)-Anweisungen wie `INSERT`, `DELETE` und `UPDATE`.
- `SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }`-Anweisungen.
- `PREPARE`- und `EXECUTE`-Anweisungen.
- `EXPLAIN`-Anweisungen mit den Anweisungen in dieser Liste

Die folgenden Arten von SQL-Anweisungen werden bei Schreibweiterleitung nicht unterstützt:

- DDL-Anweisungen (Data Definition Language)
- `ANALYZE`
- `CLUSTER`
- `COPY`
- Cursor – Cursors werden nicht unterstützt. Stellen Sie daher sicher, dass Sie sie schließen, bevor Sie die Schreibweiterleitung verwenden.
- `GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL`
- `LOCK`
- `SAVEPOINT`-Anweisungen.
- `SELECT INTO`
- `SET CONSTRAINTS`
- `TRUNCATE`
- `VACUUM`

Isolation und Konsistenz für die Schreibweiterleitung in Aurora PostgreSQL

In Sitzungen, die Schreibweiterleitung verwenden, können Sie nur die Isolationsstufen `REPEATABLE READ` und `READ COMMITTED` verwenden. Die `SERIALIZABLE`-Isolationsstufe wird jedoch nicht unterstützt.

Sie können den Grad der Lesekonsistenz in einem sekundären Cluster steuern. Die Lesekonsistenzstufe legt fest, wie viel Wartezeit der sekundäre Cluster vor jeder Leseoperation ausführt, um sicherzustellen, dass einige oder alle Änderungen aus dem primären Cluster repliziert werden. Sie können die Lesekonsistenzstufe anpassen, um sicherzustellen, dass alle aus Ihrer Sitzung weitergeleiteten Schreiboperationen im sekundären Cluster vor nachfolgenden Abfragen

angezeigt werden. Sie können diese Einstellung auch verwenden, um sicherzustellen, dass Abfragen auf dem sekundären Cluster stets die aktuellsten Updates des primären Clusters angezeigt werden. Dies gilt auch für diejenigen, die von anderen Sitzungen oder anderen Clustern übermittelt wurden. Um diese Art von Verhalten für Ihre Anwendung anzugeben, wählen Sie den entsprechenden Wert für den `apg_write_forward.consistency_mode`-Parameter aus. Der Parameter `apg_write_forward.consistency_mode` wirkt sich nur auf sekundäre Cluster aus, in denen die Schreibweiterleitung aktiviert ist.

> **Note**
>
> Für den Parameter `apg_write_forward.consistency_mode` können Sie die Werte `SESSION`, `EVENTUAL`, `GLOBAL` oder `OFF` angeben. Standardmäßig ist der Wert auf `SESSION` festgelegt. Wenn Sie den Wert auf `OFF` festlegen, wird die Schreibweiterleitung in der Sitzung deaktiviert.

Wenn Sie das Konsistenzniveau erhöhen, verbringt Ihre Anwendung mehr Zeit mit dem Warten auf die Propagierung von Änderungen zwischen AWS-Regionen. Sie können das Verhältnis zwischen schneller Reaktionszeit und der Gewährleistung festlegen, dass vor der Ausführung Ihrer Abfragen Änderungen an anderen Speicherorten vollständig verfügbar sind.

Bei jeder verfügbaren Konsistenzmodus-Einstellung hat dies folgende Auswirkungen:

- `SESSION` – Allen Abfragen in einer sekundären AWS-Region, die die Schreibweiterleitung verwendet, werden die Ergebnisse aller in dieser Sitzung ausgeführten Änderungen angezeigt. Die Änderungen werden unabhängig davon angezeigt, ob die Transaktion übergeben wurde. Wenn notwendig, wartet die Abfrage auf die Replikation der Ergebnisse weitergeleiteter Schreiboperationen zur aktuellen Region. Sie wartet nicht auf aktualisierte Ergebnisse aus Schreiboperationen, die in anderen Regionen oder in anderen Sitzungen innerhalb der aktuellen Region ausgeführt werden.
- `EVENTUAL` – Abfragen in einer sekundären AWS-Region, die die Schreibweiterleitung verwendet, werden möglicherweise Daten angezeigt, die aufgrund von Replikationsverzögerungen leicht veraltet sind. Ergebnisse von Schreiboperationen in derselben Sitzung werden erst angezeigt, wenn die Schreiboperation in der primären Region ausgeführt und zur aktuellen Region repliziert wird. Die Abfrage wartet nicht, bis die aktualisierten Ergebnisse verfügbar sind. Daher könnte sie die älteren Daten oder die aktualisierten Daten abrufen, abhängig vom Zeitpunkt der Anweisungen und der Größe der Replikationsverzögerung.

- `GLOBAL` – In einer Sitzung in einer sekundären AWS -Region werden Änderungen angezeigt, die von dieser Sitzung durchgeführt wurden. Außerdem werden ihr alle übergebenen Änderungen sowohl aus der primären AWS-Region als auch aus anderen sekundären AWS-Regionen angezeigt. Jede Abfrage kann für einen bestimmten Zeitraum warten, abhängig von der Sitzungsverzögerung. Die Abfrage wird fortgesetzt, wenn der sekundäre Cluster mit allen übergebenen Daten aus dem primären Cluster mit Stand zu dem Zeitpunkt, an dem die Abfrage gestartet wurde, aktualisiert ist.
- `OFF` – Die Schreibweiterleitung ist in der Sitzung deaktiviert.

Weitere Informationen zu allen mit der Schreibweiterleitung verbundenen Parametern finden Sie unter Konfigurationsparameter für die Schreibweiterleitung in Aurora PostgreSQL.

Transaktionszugriffsmodi mit Schreibweiterleitung

Wenn der Transaktionszugriffsmodus schreibgeschützt ist, wird die Schreibweiterleitung nicht verwendet. Sie können den Zugriffsmodus nur dann auf Lese- und Schreibzugriff festlegen, wenn Sie mit einem DB-Cluster und einer Sitzung verbunden sind, in der die lokale Schreibweiterleitung aktiviert ist.

Weitere Informationen zu den Transaktionszugriffsmodi finden Sie unter SET TRANSACTION.

Ausführen von Multipart-Anweisungen mit Schreibweiterleitung in Aurora PostgreSQL

Eine DML-Anweisung kann aus mehreren Teilen bestehen, z. B. einer `INSERT ... SELECT`-Anweisung oder einer `DELETE ... WHERE`-Anweisung. In diesem Fall wird die gesamte Anweisung an den primären Cluster weitergeleitet und dort ausgeführt.

Konfigurationsparameter für die Schreibweiterleitung in Aurora PostgreSQL

Die Aurora-Cluster-Parametergruppen enthalten Einstellungen für die Schreibweiterleitung. Da es sich um Cluster-Parameter handelt, haben alle DB-Instances in allen Clustern die gleichen Werte für diese Variablen. Details zu diesen Parametern sind in der folgenden Tabelle zusammengefasst, mit Nutzungshinweisen unterhalb der Tabelle.

Name	Scope	Typ	Standardwert	Zulässige Werte
apg_write_forward.connect_timeout	Sitzung	Sekunden	30	0 – 2147483647
apg_write_forward.consistency_mode	Sitzung	enum	Sitzung	SESSION, EVENTUAL, GLOBAL, OFF
apg_write_forward.idle_in_transaction_session_timeout	Sitzung	Millisekunden	86400000	0 – 2147483647
apg_write_forward.idle_session_timeout	Sitzung	Millisekunden	300000	0 – 2147483647
apg_write_forward.max_forwarding_connections_percent	Global	int	25	1-100

Der apg_write_forward.max_forwarding_connections_percent-Parameter definiert die Obergrenze der Datenbankverbindungsslots, die zum Umgang mit von Readern weitergeleiteten Abfragen verwendet werden können. Er wird als Prozentsatz der Einstellung max_connections für die Writer-DB-Instance im primären Cluster ausgedrückt. Wenn beispielsweise max_connections 800 und apg_write_forward.max_forwarding_connections_percent 10 ist, dann lässt der Writer maximal 80 weitergeleitete Sitzungen gleichzeitig zu. Diese Verbindungen stammen aus demselben, von der Einstellung max_connections verwalteten Verbindungspool. Diese Einstellung gilt nur für den primären Cluster, wenn für mindestens einen sekundären Cluster die Schreibweiterleitung aktiviert ist.

Verwenden Sie die folgenden Einstellungen auf dem sekundären Cluster:

- apg_write_forward.consistency_mode – Ein Parameter auf Sitzungsebene, der den Grad der Lesekonsistenz auf dem sekundären Cluster steuert. Gültige Werte sind SESSION, EVENTUAL,

`GLOBAL` oder `OFF`. Standardmäßig ist der Wert auf `SESSION` festgelegt. Wenn Sie den Wert auf `OFF` festlegen, wird die Schreibweiterleitung in der Sitzung deaktiviert. Weitere Informationen über Konsistenzebenen finden Sie unter Isolation und Konsistenz für die Schreibweiterleitung in Aurora PostgreSQL. Dieser Parameter ist nur in Reader-Instances sekundärer Cluster relevant, für die Schreibweiterleitung aktiviert ist und die sich in einer Aurora Global Database befinden.

- `apg_write_forward.connect_timeout` – Die maximale Anzahl von Sekunden, die der sekundäre Cluster beim Herstellen einer Verbindung zum primären Cluster wartet, bevor er aufgibt. Ein Wert von `0` bedeutet, dass auf unbestimmte Zeit gewartet werden soll.
- `apg_write_forward.idle_in_transaction_session_timeout` – Die Anzahl der Millisekunden, die der primäre Cluster auf Aktivität für eine Verbindung wartet, die aus einem sekundären Cluster mit einer offenen Transaktion weitergeleitet wird, bevor er sie schließt. Wenn die Sitzung über diesen Zeitraum hinaus inaktiv ist, bricht Aurora die Sitzung ab. Durch einen Wert des Typs `0` wird der Timout deaktiviert.
- `apg_write_forward.idle_session_timeout` – Die Anzahl der Millisekunden, die der primäre Cluster auf Aktivität für eine Verbindung wartet, die aus einem sekundären Cluster weitergeleitet wird, bevor er sie schließt. Wenn die Sitzung über diesen Zeitraum hinaus inaktiv bleibt, beendet Aurora die Sitzung. Durch einen Wert des Typs `0` wird der Timout deaktiviert.

Amazon-CloudWatch-Metriken für die Schreibweiterleitung in Aurora PostgreSQL

Die folgenden Amazon CloudWatch-Metriken gelten für den primären Cluster, wenn Sie die Schreibweiterleitung auf einem oder mehreren sekundären Clustern verwenden. Diese Metriken werden alle auf der Writer-DB-Instance im primären Cluster gemessen.

CloudWatch-Metrik	Einheiten und Beschreibung
`AuroraForwardingWriterDMLThroughput`	Anzahl pro Sekunde Anzahl der weitergeleiteten DML-Anweisungen, die pro Sekunde von dieser Writer-DB-Instance verarbeitet werden.
`AuroraForwardingWriterOpenSessions`	Anzahl Anzahl der offenen Sitzungen auf dieser Writer-DB-Instance, die weitergeleitete Anfragen verarbeiten.

CloudWatch-Metrik	Einheiten und Beschreibung
`AuroraForwardingWriterTotalSessions`	Anzahl Anzahl der weitergeleiteten Sitzungen auf dieser Writer-DB-Instance.

Die folgenden CloudWatch-Metriken gelten für jeden sekundären Cluster. Diese Metriken werden auf jeder Reader-DB-Instance in einem sekundären Cluster mit aktivierter Schreibweiterleitung gemessen.

CloudWatch-Metrik	Einheit und Beschreibung
`AuroraForwardingReplicaCommitThroughput`	Anzahl pro Sekunde Anzahl der Commits in Sitzungen, die von diesem Replikat pro Sekunde weitergeleitet werden.
`AuroraForwardingReplicaDMLLatency`	Millisekunden Durchschnittliche Reaktionszeit weitergeleiteter DML-Anweisungen in Millisekunden auf Replikaten.
`AuroraForwardingReplicaDMLThroughput`	Anzahl pro Sekunde Anzahl der weitergeleiteten DML-Anweisungen, die in diesem Replikat pro Sekunde verarbeitet werden.
`AuroraForwardingReplicaErrorSessionsLimit`	Anzahl Anzahl der Sitzungen, die vom primären Cluster zurückgewiesen wurden, weil das Limit für die maximale Zahl von Verbindungen oder die maximale Schreibweiterleitung erreicht wurde.
`AuroraForwardingReplicaOpenSessions`	Anzahl Die Anzahl der Sitzungen, die die Schreibweiterleitung für eine Replikat-Instance verwenden.
`AuroraForwardingReplicaReadWaitLatency`	Millisekunden Durchschnittliche Wartezeit in Millisekunden, die das Replikat darauf wartet, mit der LSN des primären Clusters konsistent zu sein. Das Ausmaß, in dem die

CloudWatch-Metrik	Einheit und Beschreibung
	Reader-DB-Instance vor der Verarbeitung einer Abfrage wartet, ist von der Einstellung `apg_write_forward.consistency_mode` abhängig. Weitere Informationen zu dieser Einstellung finden Sie unter the section called “Konfigurationsparameter für die Schreibweiterleitung in Aurora PostgreSQL”.

Warte-Ereignisse für die Schreibweiterleitung in Aurora PostgreSQL

Amazon Aurora generiert die folgenden Warteereignisse, wenn Sie die Schreibweiterleitung mit Aurora PostgreSQL verwenden.

Themen

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort
- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

Das `IPC:AuroraWriteForwardConnect`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem sekundären DB-Cluster darauf wartet, dass eine Verbindung zum Writer-Knoten des primären DB-Clusters geöffnet wird.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Dieses Ereignis nimmt zu, wenn die Anzahl der Verbindungsversuche vom Leseknoten einer sekundären Region zum Writer-Knoten des primären DB-Clusters zunimmt.

Aktionen

Reduzieren Sie die Anzahl gleichzeitiger Verbindungen von einem sekundären Knoten zum Writer-Knoten der primären Region.

IPC:AuroraWriteForwardConsistencyPoint

Das `IPC:AuroraWriteForwardConsistencyPoint`-Ereignis beschreibt, wie lange eine Abfrage von einem Knoten im sekundären-DB-Cluster darauf wartet, dass die Ergebnisse weitergeleiteter Schreiboperationen in die aktuelle Region repliziert werden. Dieses Ereignis wird nur generiert, wenn der Parameter `apg_write_forward.consistency_mode` auf Sitzungsebene auf einen der folgenden Werte gesetzt ist:

- `SESSION` – Abfragen auf einem sekundären Knoten warten auf die Ergebnisse aller in dieser Sitzung ausgeführten Änderungen.
- `GLOBAL` – Abfragen auf einem sekundären Knoten warten auf die Ergebnisse der in dieser Sitzung vorgenommenen Änderungen sowie auf alle festgeschriebenen Änderungen sowohl aus der primären Region als auch aus anderen sekundären Regionen im globalen Cluster.

Informationen zum Einstellen des Parameters `apg_write_forward.consistency_mode` finden Sie unter the section called “Konfigurationsparameter für die Schreibweiterleitung in Aurora PostgreSQL”.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten gehören:

- Erhöhte Replikatverzögerung, gemessen anhand der Amazon CloudWatch-Metrik `ReplicaLag`. Weitere Informationen zu dieser Metrik finden Sie unter Überwachung einer Aurora PostgreSQL-Replikation.
- Erhöhte Last auf dem Writer-Knoten der primären Region oder auf dem sekundären Knoten.

Aktionen

Ändern Sie Ihren Konsistenzmodus je nach den Anforderungen Ihrer Anwendung.

IPC:AuroraWriteForwardExecute

Das `IPC:AuroraWriteForwardExecute`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem sekundären DB-Cluster darauf wartet, dass eine weitergeleitete Abfrage abgeschlossen wird und Ergebnisse vom Writer-Knoten des primären DB-Clusters abgerufen werden.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Abrufen einer großen Zahl von Zeilen aus dem Writer-Knoten der primären Region.
- Eine erhöhte Netzwerklatenz zwischen dem sekundären Knoten und dem Writer-Knoten der primären Region erhöht die Zeit, die der sekundäre Knoten benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Belastung des sekundären Knotens kann die Übertragung der Abfrageanforderung vom sekundären Knoten zum Writer-Knoten der primären Region verzögern.
- Eine erhöhte Belastung des Writer-Knotens der primären Region kann die Übertragung der Daten vom Writer-Knoten zum sekundären Knoten verzögern.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

- Optimieren Sie Abfragen, um nur die erforderlichen Daten abzurufen.
- Optimieren Sie DML-Operationen (Data Manipulation Language), um nur die erforderlichen Daten zu ändern.
- Wenn der sekundäre Knoten oder der Writer-Knoten der primären Region durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, ihn auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

Das `IPC:AuroraWriteForwardGetGlobalConsistencyPoint`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem sekundären DB-Cluster, der den Konsistenzmodus GLOBAL verwendet, darauf wartet, den globalen Konsistenzpunkt vom Writer-Knoten zu erhalten, bevor er eine Abfrage ausführt.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem sekundären Knoten und dem Writer-Knoten der primären Region erhöht die Zeit, die der sekundäre Knoten benötigt, um Daten vom Writer-Knoten zu empfangen.

- Eine erhöhte Belastung des sekundären Knotens kann die Übertragung der Abfrageanforderung vom sekundären Knoten zum Writer-Knoten der primären Region verzögern.
- Eine erhöhte Belastung des Writer-Knotens der primären Region kann die Übertragung der Daten vom Writer-Knoten zum sekundären Knoten verzögern.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

- Ändern Sie Ihren Konsistenzmodus je nach den Anforderungen Ihrer Anwendung.
- Wenn der sekundäre Knoten oder der Writer-Knoten der primären Region durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, ihn auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardXactAbort

Das `IPC:AuroraWriteForwardXactAbort`-Ereignis tritt ein, wenn ein Back-End-Prozess auf dem sekundären DB-Cluster auf das Ergebnis einer Remote-Bereinigungsabfrage wartet. Bereinigungsabfragen werden ausgegeben, um den Prozess nach dem Abbruch einer per Schreibweiterleitung weitergeleiteten Transaktion wieder in den entsprechenden Zustand zu versetzen. Amazon Aurora führt diese entweder aus, weil ein Fehler gefunden wurde oder weil ein Benutzer einen expliziten `ABORT`-Befehl ausgegeben oder eine laufende Abfrage abgebrochen hat.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem sekundären Knoten und dem Writer-Knoten der primären Region erhöht die Zeit, die der sekundäre Knoten benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Belastung des sekundären Knotens kann die Übertragung der Bereinigungsabfrage-Anforderung vom sekundären Knoten zum Writer-Knoten der primären Region verzögern.
- Eine erhöhte Belastung des Writer-Knotens der primären Region kann die Übertragung der Daten vom Writer-Knoten zum sekundären Knoten verzögern.

Aktionen

Abhängig von den Ursachen Ihres Warteereignisses empfehlen wir verschiedene Aktionen.

- Untersuchen Sie die Ursache für die abgebrochene Transaktion.
- Wenn der sekundäre Knoten oder der Writer-Knoten der primären Region durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, ihn auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardXactCommit

Das `IPC:AuroraWriteForwardXactCommit`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem sekundären DB-Cluster auf das Ergebnis eines weitergeleiteten Commit-Transaktionsbefehls wartet.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem sekundären Knoten und dem Writer-Knoten der primären Region erhöht die Zeit, die der sekundäre Knoten benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Belastung des sekundären Knotens kann die Übertragung der Abfrageanforderung vom sekundären Knoten zum Writer-Knoten der primären Region verzögern.
- Eine erhöhte Belastung des Writer-Knotens der primären Region kann die Übertragung der Daten vom Writer-Knoten zum sekundären Knoten verzögern.

Aktionen

Wenn der sekundäre Knoten oder der Writer-Knoten der primären Region durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, ihn auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

IPC:AuroraWriteForwardXactStart

Das `IPC:AuroraWriteForwardXactStart`-Ereignis tritt ein, wenn ein Backend-Prozess auf dem sekundären DB-Cluster auf das Ergebnis eines weitergeleiteten Transaktionsstartbefehls wartet.

Wahrscheinliche Ursachen für erhöhte Wartezeiten

Zu den häufigsten Ursachen für längere Wartezeiten zählen auch die Folgenden:

- Eine erhöhte Netzwerklatenz zwischen dem sekundären Knoten und dem Writer-Knoten der primären Region erhöht die Zeit, die der sekundäre Knoten benötigt, um Daten vom Writer-Knoten zu empfangen.
- Eine erhöhte Belastung des sekundären Knotens kann die Übertragung der Abfrageanforderung vom sekundären Knoten zum Writer-Knoten der primären Region verzögern.
- Eine erhöhte Belastung des Writer-Knotens der primären Region kann die Übertragung der Daten vom Writer-Knoten zum sekundären Knoten verzögern.

Aktionen

Wenn der sekundäre Knoten oder der Writer-Knoten der primären Region durch die CPU- oder Netzwerkbandbreite eingeschränkt ist, sollten Sie erwägen, ihn auf einen Instance-Typ mit mehr CPU-Kapazität oder mehr Netzwerkbandbreite umzustellen.

Verwenden von Umstellung oder Failover in Amazon Aurora Global Database

Die Funktion von Aurora Global Database bietet einen höheren Business Continuity und Disaster Recovery (BCDR)-Schutz als die standardmäßige Hochverfügbarkeit, die von einem Aurora-DB-Cluster in einer einzelnen AWS-Region bereitgestellt wird. Mithilfe von Aurora Global Database können Sie für eine schnellere Wiederherstellung nach raren, nicht geplanten regionalen Notfällen planen oder vollständige Service-Level-Ausfälle schnell beheben.

Sie können die folgenden Anleitungen und Verfahren nutzen, um Ihre BCDR-Strategie mit der Funktion von Aurora Global Database zu planen, zu testen und zu implementieren.

Themen

- Planen in Bezug auf Business Continuity und Disaster Recovery
- Durchführen von Umstellungen für Amazon Aurora Global Databases
- Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall
- Verwalten von RPOs für Aurora PostgreSQL–basierte globale Datenbanken
- Regionsübergreifende Resilienz für sekundäre Cluster globaler Datenbanken

Planen in Bezug auf Business Continuity und Disaster Recovery

Um Ihre BCDR-Strategie zu planen, sollten Sie die folgende branchenspezifische Terminologie kennen und wissen, wie diese Begriffe im Zusammenhang mit den Funktionen von Aurora Global Database zu verstehen sind.

Die Wiederherstellung nach einem Notfall wird in der Regel von den folgenden beiden Geschäftszielen bestimmt:

- Recovery Time Objective (RTO) – Die Zeit, die ein System benötigt, um nach einem Notfall in einen arbeitsfähigen Zustand zurückzukehren. Mit anderen Worten: RTO misst die Ausfallzeit. Bei Aurora Global Database kann sich die RTO in einer Größenordnung von Minuten bewegen.
- Recovery Point Objective (RPO) – Die Datenmenge, die nach einer Katastrophe oder einem Serviceausfall verloren gehen kann (gemessen in Zeit). Dieser Datenverlust ist in der Regel auf asynchrone Replikationsverzögerungen zurückzuführen. Für eine globale Aurora-Datenbank wird die RPO in der Regel in Sekunden gemessen. Mit einer Aurora PostgreSQL–basierten globalen Datenbank können Sie den Parameter `rds.global_db_rpo` verwenden, um die RPO-Obergrenze festzulegen und zu überwachen, dies könnte jedoch die Transaktionsverarbeitung auf dem Writer-Knoten des primären Clusters beeinträchtigen. Weitere Informationen finden Sie unter Verwalten von RPOs für Aurora PostgreSQL–basierte globale Datenbanken.

Die Durchführung einer Umstellung oder eines Failovers mit Aurora Global Database beinhaltet die Heraufstufung eines sekundären DB-Clusters zum primären DB-Cluster. Der Begriff „regionaler Ausfall“ wird häufig verwendet, um eine Vielzahl von Ausfallszenarien zu beschreiben. Ein schlimmstmögliches Szenario könnte ein großflächiger Ausfall aufgrund eines katastrophalen Ereignisses sein, das Hunderte von Quadratkilometern betrifft. Die meisten Ausfälle sind jedoch viel stärker lokal begrenzt und betreffen nur einen kleinen Teil der Cloud-Dienste oder Kundensysteme. Berücksichtigen Sie den gesamten Umfang des Ausfalls, um sicherzustellen, dass regionsübergreifendes Failover die richtige Lösung ist, und wählen Sie die für die jeweilige Situation geeignete Failover-Methode aus. Ob Sie den Failover- oder den Umstellungs-Ansatz verwenden sollten, hängt vom jeweiligen Ausfallszenario ab:

- Failover – Verwenden Sie diesen Ansatz, um die Daten nach einem ungeplanten Ausfall wiederherzustellen. Damit führen Sie ein regionsübergreifendes Failover auf einen der sekundären DB-Cluster in Ihrer globalen Aurora-Datenbank durch. Das RPO für diesen Ansatz ist in der Regel ein Wert ungleich Null, der in Sekunden gemessen wird. Das Ausmaß des Datenverlusts hängt von der Verzögerung der Replikation der globalen Aurora-Datenbank über die AWS-Regionen zum

Zeitpunkt des Ausfalls ab. Weitere Informationen hierzu finden Sie unter Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall.

- Umstellung – Dieser Vorgang wurde zuvor als „verwaltetes geplantes Failover“ bezeichnet. Verwenden Sie diesen Ansatz für kontrollierte Szenarien, z. B. für die betriebliche Wartung und andere geplante betriebliche Verfahren, bei denen alle Aurora-Cluster und andere Services, mit denen sie interagieren, in einem fehlerfreien Zustand sind. Da diese Funktion die sekundären DB-Cluster mit dem primären synchronisiert, bevor andere Änderungen vorgenommen werden, ist der RPO 0 (kein Datenverlust). Weitere Informationen hierzu finden Sie unter Durchführen von Umstellungen für Amazon Aurora Global Databases.

> **Note**
>
> Vor einer Umstellung auf einen sekundären Headless-DB-Cluster von Aurora bzw. vor einem Failover zu diesem müssen Sie ihm eine DB-Instance hinzufügen. Weitere Informationen zu kopflosen DB-Clustern finden Sie unter Erstellen eines Aurora-Headless-DB-Clusters in einer sekundären Region.

Durchführen von Umstellungen für Amazon Aurora Global Databases

> **Note**
>
> Umstellungen wurden früher als verwaltete geplante Failover bezeichnet.

Mithilfe von Umstellungen können Sie die Region Ihres primären Clusters routinemäßig ändern. Dieser Ansatz ist für kontrollierte Szenarien gedacht, z. B. für betriebliche Wartungen und andere geplante Betriebsverfahren.

Es gibt drei gängige Anwendungsfälle für die Verwendung von Umstellungen.

- Für Anforderungen an die „regionale Rotation“, die in bestimmten Branchen gelten. Beispielsweise könnten die Vorschriften für Finanzdienstleistungen vorschreiben, dass Tier-0-Systeme für mehrere Monate in eine andere Region wechseln müssen, um sicherzustellen, dass die Notfallwiederherstellungsverfahren regelmäßig geübt werden.

- Für regionsübergreifende „Follow-the-Sun“-Anwendungen. Beispielsweise möchte ein Unternehmen möglicherweise Schreibvorgänge mit niedrigerer Latenz in verschiedenen Regionen bereitstellen, basierend auf den Geschäftszeiten in verschiedenen Zeitzonen.
- Als Methode ohne Datenverlust, um nach einem Failover zur ursprünglichen Primärregion zurückzukehren.

> **Note**
>
> Umstellungen sind für die Verwendung bei einer globalen Aurora-Datenbank konzipiert, bei der sich alle Aurora-Cluster sowie andere Services, mit denen sie agieren, in einem fehlerfreien Zustand befinden. Folgen Sie zur Wiederherstellung nach einem ungeplanten Ausfall dem entsprechenden Verfahren unter Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall.
> Sie können verwaltete regionsübergreifende Umstellungen mit Aurora Global Database nur durchführen, wenn die primären und sekundären DB-Cluster dieselben Engine-Haupt- und Nebenversionen haben. Je nach Engine und Engine-Versionen müssen die Patch-Level möglicherweise identisch sein oder können unterschiedlich sein. Eine Liste der Engines und Engine-Versionen, die diese Operationen zwischen primären und sekundären Clustern mit unterschiedlichen Patch-Leveln unterstützen, finden Sie unter Patch-Level-Kompatibilität für verwaltete regionsübergreifende Umstellungen und Failover. Bevor Sie mit der Umstellung beginnen, überprüfen Sie die Engine-Versionen in Ihrem globalen Cluster, um sicherzustellen, dass diese verwaltete regionsübergreifende Umstellungen unterstützen, und aktualisieren Sie sie bei Bedarf.

Während einer Umstellung macht Aurora den Cluster in der von Ihnen ausgewählten sekundären Region zu Ihrem primären Cluster. Der Umstellungsmechanismus behält die bestehende Replikationstopologie Ihrer globalen Datenbank bei: Sie hat immer noch dieselbe Anzahl von Aurora-Clustern in denselben Regionen. Ehe der Umstellungsprozess gestartet wird, wartet Aurora darauf, das alle Ziel-Cluster der sekundären Region vollständig mit dem Cluster der primären Region synchronisiert sind. Dann ist der DB-Cluster in der primären Region schreibgeschützt. Der gewählte sekundäre Cluster stuft einen seiner schreibgeschützten Knoten auf den Status eines vollständigen Writers hoch, sodass der sekundäre Cluster die Rolle des primären Clusters übernehmen kann. Da der sekundäre Ziel-Cluster zu Beginn des Prozesses mit dem primären Cluster synchronisiert wurde, setzt die neue primäre Version den Betrieb für die globale Aurora-Datenbank fort, ohne dass

Daten verloren gehen. Ihre Datenbank ist für kurze Zeit nicht verfügbar, während der primäre und der ausgewählte sekundäre Cluster ihre neuen Rollen übernehmen.

> **Note**
>
> Informationen zur Verwaltung von Replikations-Slots für Aurora PostgreSQL nach einer Umstellung finden Sie unter Verwalten logischer Slots für Aurora PostgreSQL .

Um die Anwendungsverfügbarkeit zu optimieren, empfehlen wir Ihnen, vor der Verwendung dieser Funktion Folgendes zu tun:

- Führen Sie diesen Vorgang außerhalb der Spitzenzeiten oder zu einem anderen Zeitpunkt durch, wenn die Schreibvorgänge auf den primären DB-Clustern minimal sind.
- Überprüfen Sie die Verzögerungszeiten für alle sekundären Aurora-DB-Cluster in der globalen Aurora-Datenbank. Verwenden Sie für alle Aurora PostgreSQL-basierten globalen Datenbanken und für Aurora MySQL-basierte globale Datenbanken ab Engine-Versionen 3.04.0 und höher oder 2.12.0 und höher Amazon CloudWatch, um die `AuroraGlobalDBRPOLag`-Metrik für alle sekundären DB-Cluster anzuzeigen. Zeigen Sie für niedrigere Nebenversionen von Aurora MySQL-basierten globalen Datenbanken stattdessen die `AuroraGlobalDBReplicationLag`-Metrik an. Diese Metriken geben an, wie weit (in Millisekunden) ein sekundärer Cluster gegenüber dem primären DB-Cluster verzögert ist. Der Wert verhält sich direkt proportional zu der Zeit, die Aurora zum Abschließen der Umstellung benötigt. Je größer der Verzögerungswert, desto länger dauert die Umstellung. Wenn Sie diese Metriken untersuchen, sollten Sie beim aktuellen primären Cluster beginnen.

  Weitere Informationen zu den CloudWatch-Metriken für Aurora finden Sie unter Metriken auf Cluster-Ebene für Amazon Aurora.
- Der sekundäre DB-Cluster, der während einer Umstellung hochgestuft wird, hat möglicherweise andere Konfigurationseinstellungen als der alte primäre DB-Cluster. Wir empfehlen, dass Sie die folgenden Arten von Konfigurationseinstellungen für alle Cluster in Ihren globalen Aurora-DB-Clustern konsistent halten. Auf diese Weise können Leistungsprobleme, Workload-Inkompatibilitäten und anderes anomales Verhalten nach einer Umstellung minimiert werden.
  - Konfigurieren einer Parametergruppe für Aurora-DB-Cluster für die neue primäre Version, falls erforderlich – Wenn Sie einen sekundären DB-Cluster zur Übernahme der primären Rolle hochstufen, kann die Parametergruppe des sekundären Clusters im Vergleich zum primären Cluster möglicherweise anders konfiguriert sein. Ist dies der Fall, ändern Sie die

Parametergruppe des hochgestuften sekundären DB-Clusters so, dass sie den Einstellungen des primären Clusters entspricht. Um zu erfahren wie, siehe Ändern von Parametern für eine Aurora globale Datenbank.

- Konfigurieren Sie Überwachungstools und -optionen wie Amazon CloudWatch Events und Alarme – Konfigurieren Sie den hochgestuften DB-Cluster mit den gleichen Protokollierungsfunktionen, Alarmen usw. wie für die globale Datenbank erforderlich. Wie bei Parametergruppen wird die Konfiguration für diese Funktionen während der Umstellung nicht vom primären Cluster übernommen. Einige CloudWatch-Metriken, wie z. B. die Verzögerung bei der Replikation, sind nur für sekundäre Regionen verfügbar. Daher ändert sich bei einer Umstellung die Art und Weise, wie diese Metriken angezeigt und Alarme für sie eingerichtet werden, und es können Änderungen an allen vordefinierten Dashboards erforderlich sein. Weitere Informationen zu Aurora-DB-Cluster und zur Überwachung finden Sie unter Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch.
- Konfigurieren Sie Integrationen mit anderen AWS-Services – Wenn Ihre globale Aurora-Datenbank mit AWS-Services wie AWS Secrets Manager, AWS Identity and Access Management, Amazon S3 und AWS Lambda integriert ist, müssen Sie sicherstellen, dass diese bedarfsgerecht konfiguriert sind. Weitere Informationen zur Integration globaler Aurora-Datenbanken in IAM, Amazon S3 und Lambda finden Sie unter Verwenden von globalen Amazon-Aurora-Datenbanken mit anderen AWS-Services. Weitere Informationen zu Secrets Manager finden Sie unter So automatisieren Sie die Replikation von Secrets in AWS Secrets Manager über verschiedene AWS-Regionen hinweg.

Wenn Sie den Writer-Endpunkt von Aurora Global Database verwenden, müssen Sie die Verbindungseinstellungen in Ihrer Anwendung nicht ändern. Stellen Sie sicher, dass die DNS-Änderungen weitergegeben wurden und dass Sie eine Verbindung herstellen und Schreibvorgänge auf dem neuen primären Cluster ausführen können. Anschließend können Sie den vollen Betrieb Ihrer Anwendung wieder aufnehmen.

Angenommen, Ihre Anwendungsverbindungen verwenden den Cluster-Endpunkt des alten primären Clusters und nicht den globalen Writer-Endpunkt. Stellen Sie in diesem Fall sicher, dass Sie die Verbindungseinstellungen Ihrer Anwendung so ändern, dass der Cluster-Endpunkt des neuen primären Clusters verwendet wird. Wenn Sie die angegebenen Namen beim Erstellen der globalen Aurora-Datenbank akzeptiert haben, können Sie den Endpunkt ändern, indem Sie `-ro` aus der Endpunkt-Zeichenfolge des hochgestuften Clusters in Ihrer Anwendung entfernen. Beispielsweise wird der Endpunkt des sekundären Clusters `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` zu `my-`

`global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`, wenn dieser Cluster zum primären Cluster hochgestuft wird.

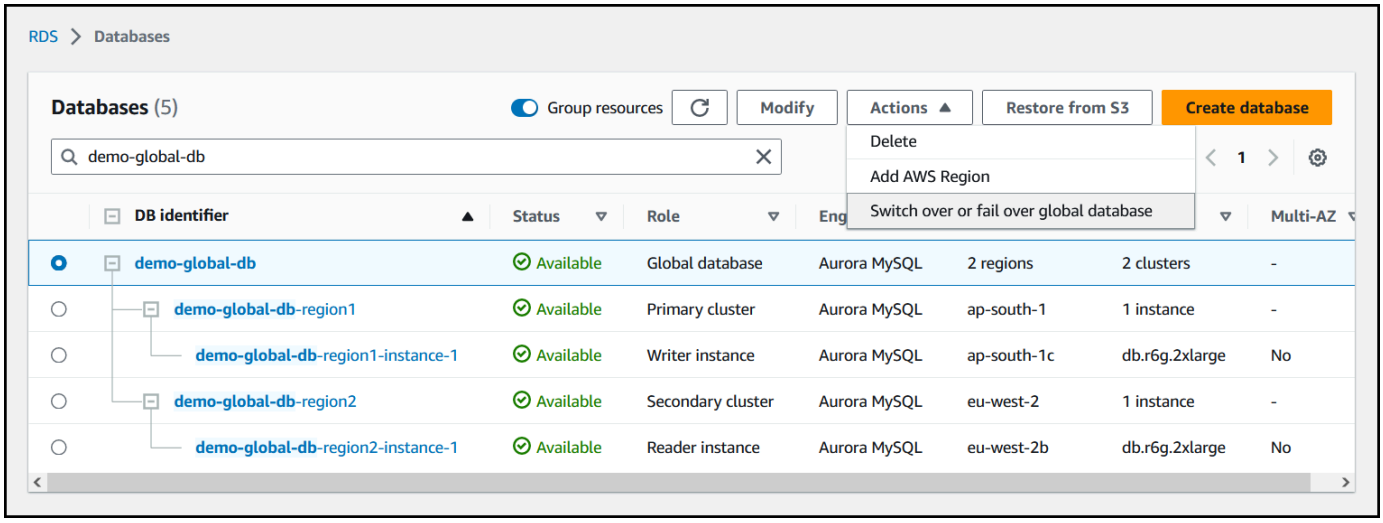
Stellen Sie, wenn Sie RDS-Proxy verwenden, sicher, dass Sie die Schreibvorgänge Ihrer Anwendung an den entsprechenden Lese-/Schreibendpunkt des Proxys umleiten, der dem neuen primären Cluster zugeordnet ist. Dieser Proxy-Endpunkt kann der Standardendpunkt oder ein benutzerdefinierter Lese-/Schreibendpunkt sein. Weitere Informationen finden Sie unter So funktionieren RDS-Proxy-Endpunkte mit globalen Datenbanken.

Sie können eine Umstellung einer globalen Aurora-Datenbank mit der AWS-Managementkonsole, der AWS CLI oder der RDS-API durchführen.

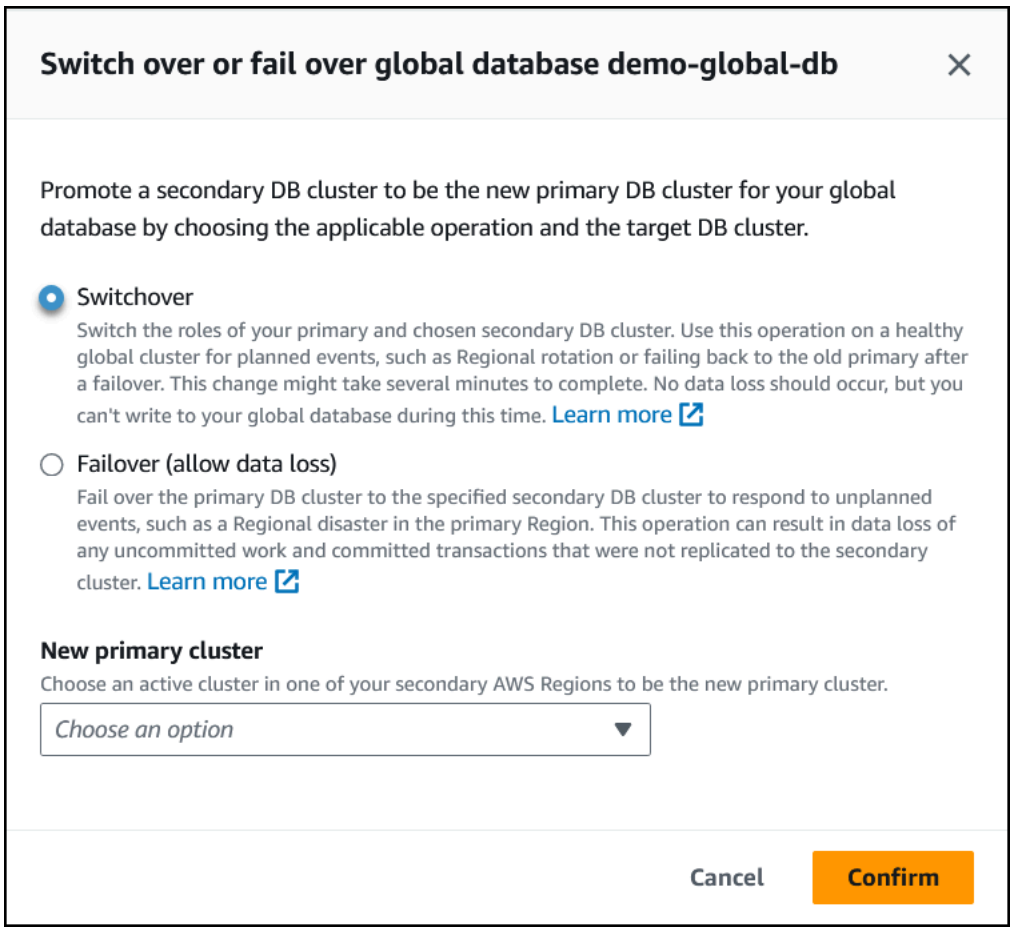
Konsole

So führen Sie eine Umstellung in Ihrer Aurora Global Database durch

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Datenbanken aus und suchen Sie die globale Aurora-Datenbank, für die Sie eine Umstellung ausführen möchten.
3. Wählen Sie im Menü Aktionen die Option Failover für globale Datenbank ausführen aus.

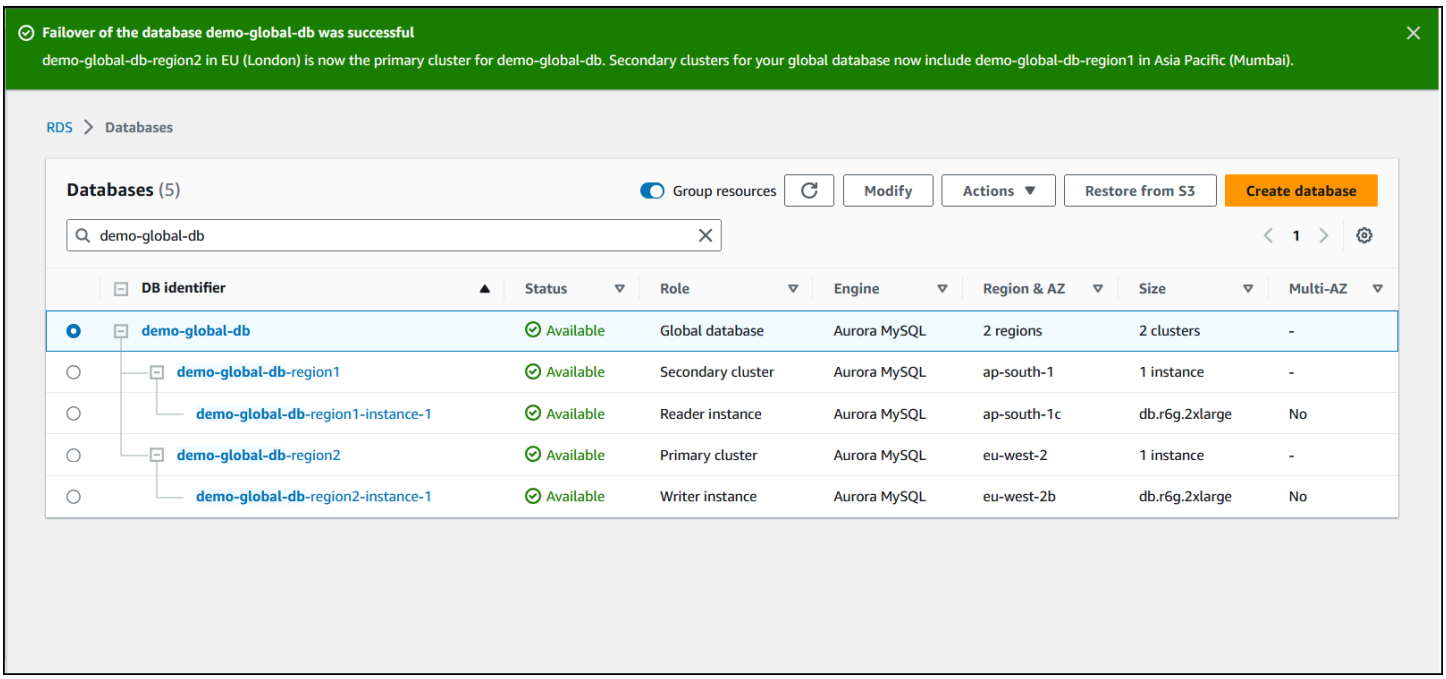


4. Wählen Sie Umschaltung.



5. Wählen Sie für Neuer primärer Cluster einen aktiven Cluster in einer Ihrer sekundären AWS-Regionen als neuen primären Cluster aus.
6. Wählen Sie Bestätigen aus.

Wenn die Umstellung abgeschlossen ist, werden die Aurora-DB-Cluster und ihr aktueller Status wie im Folgenden dargestellt in der Liste Datenbanken angezeigt.



AWS CLI

So führen Sie eine Umstellung auf einer Aurora Global Database durch

Verwenden Sie den CLI-Befehl `switchover-global-cluster`, um eine Umstellung für Aurora Global Database auszuführen. Mit dem Befehl können Sie Werte für die folgenden Parameter übertragen.

- `--region` – Geben Sie die AWS-Region an, in der der primäre DB-Cluster der globalen Aurora-Datenbank ausgeführt wird.
- `--global-cluster-identifier` – Geben Sie den Namen Ihrer globalen Aurora-Datenbank an.
- `--target-db-cluster-identifier` – Geben Sie den Amazon-Ressourcennamen (ARN) des Aurora-DB-Clusters an, den Sie als primäres Cluster für die globale Aurora-Datenbank hochstufen möchten.

Für Linux, macOS oder Unix:

```
aws rds --region region_of_primary \
   switchover-global-cluster --global-cluster-identifier global_database_id \
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

Für Windows:

```
aws rds --region region_of_primary ^
   switchover-global-cluster --global-cluster-identifier global_database_id ^
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

RDS-API

Um eine Umstellung für Aurora Global Database durchzuführen, führen Sie die API-Operation SwitchoverGlobalCluster aus.

Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall

In sehr seltenen Fällen kann es bei Ihrer globalen Aurora-Datenbank zu einem unerwarteten Ausfall in der primären AWS-Region kommen. In einem solchen Fall sind der primäre Aurora-DB-Cluster und sein Writer-Knoten nicht verfügbar, und die Replikation zwischen dem primären Cluster und den sekundären Clustern wird angehalten. Um Ausfallzeiten (RTO) und Datenverlust (RPO) zu minimieren, können Sie schnell ein regionsübergreifendes Failover durchführen.

Aurora Global Database bietet zwei Failover-Methoden, die Sie in einer Notfallwiederherstellungssituation verwenden können:

- Verwaltetes Failover — Diese Methode wird für die Notfallwiederherstellung empfohlen. Wenn Sie diese Methode verwenden, fügt Aurora die alte primäre Region automatisch wieder als sekundäre Region zur globalen Datenbank hinzu, sobald sie wieder verfügbar ist. Somit wird die ursprüngliche Topologie Ihres globalen Clusters beibehalten. Um zu erfahren, wie Sie diese Methode verwenden, vgl. Ausführen von verwalteten geplanten Failovers für globale Aurora-Datenbanken.
- Manuelles Failover – Diese alternative Methode kann verwendet werden, wenn ein verwaltetes Failover nicht in Frage kommt, z. B. wenn in Ihren primären und sekundären Regionen inkompatible Engine-Versionen ausgeführt werden. Um zu erfahren, wie Sie diese Methode verwenden, vgl. Ausführen von manuellen geplanten Failovers für globale Aurora-Datenbanken.

> **Important**
>
> Beide Failover-Methoden können zum Verlust von Schreibtransaktionsdaten führen, die vor dem Eintreten des Failover-Ereignisses nicht zur gewählten sekundären Region repliziert wurden. Der Wiederherstellungsprozess, der eine DB-Instance auf dem ausgewählten sekundären DB-Cluster zur primären Writer-DB-Instance hochstuft, garantiert jedoch, dass

sich die Daten in einem transaktionskonsistenten Zustand befinden. Bei einem Failover kann es auch zu Split-Brain-Problemen kommen.

Ausführen von verwalteten geplanten Failovers für globale Aurora-Datenbanken

Dieser Ansatz dient der Geschäftskontinuität im Falle einer echten regionalen Katastrophe oder eines kompletten Service-Level-Ausfalls.

Bei einem verwalteten Failover wird der sekundäre Cluster in Ihrer ausgewählten sekundären Region zum neuen primären Cluster. Der gewählte sekundäre Cluster stuft einen seiner schreibgeschützten Knoten auf vollen Writer-Status herauf. In diesem Schritt kann der Cluster die Rolle des primären Clusters übernehmen. Ihre Datenbank ist für kurze Zeit nicht verfügbar, während dieser Cluster seine neue Rolle annimmt. Sobald diese alte Region fehlerfrei und wieder verfügbar ist, fügt Aurora sie automatisch wieder als sekundäre Region zum globalen Cluster hinzu. Somit wird die bestehende Replikationstopologie Ihrer globalen Aurora-Datenbank beibehalten.

> **Note**
>
> Informationen zur Verwaltung von Replikations-Slots für Aurora PostgreSQL nach einem Failover finden Sie unter Verwalten logischer Slots für Aurora PostgreSQL .

> **Note**
>
> Sie können verwaltete regionsübergreifende Failover mit Aurora Global Database nur durchführen, wenn die primären und sekundären DB-Cluster dieselben Engine-Haupt- und Nebenversionen haben. Je nach Engine und Engine-Versionen müssen die Patch-Level möglicherweise identisch sein oder können unterschiedlich sein. Eine Liste der Engines und Engine-Versionen, die diese Operationen zwischen primären und sekundären Clustern mit unterschiedlichen Patch-Leveln unterstützen, finden Sie unter Patch-Level-Kompatibilität für verwaltete regionsübergreifende Umstellungen und Failover. Bevor Sie mit dem Failover beginnen, überprüfen Sie die Engine-Versionen in Ihrem globalen Cluster, um sicherzustellen, dass diese verwaltete regionsübergreifende Umstellungen unterstützen, und aktualisieren Sie sie bei Bedarf. Wenn Ihre Engine-Versionen identische Patch-Level erfordern, aber unterschiedliche Patch-Level ausführen, können Sie das Failover manuell

durchführen, indem Sie die Schritte unter Ausführen von manuellen geplanten Failovers für globale Aurora-Datenbanken ausführen.

Beim verwalteten Failover wird nicht darauf gewartet, dass Daten zwischen der ausgewählten sekundären Region und der aktuellen primären Region synchronisiert werden. Da Aurora Global Database Daten asynchron repliziert, ist es möglich, dass nicht alle Transaktionen in die gewählte sekundäre AWS-Region repliziert wurden, bevor eine Hochstufung auf volle Lese-/Schreibfunktionen erfolgte.

Um sicherzustellen, dass sich die Daten in einem konsistenten Zustand befinden, erstellt Aurora nach der Wiederherstellung ein neues Speicher-Volume für die alte primäre Region. Bevor das neue Speicher-Volume in der AWS-Region erstellt wird, versucht Aurora, zum Zeitpunkt des Fehlers einen Snapshot des alten Speicher-Volumes zu erstellen. Auf diese Weise können Sie den Snapshot wiederherstellen und somit auch alle fehlenden Daten. Wenn dieser Vorgang erfolgreich ist, platziert Aurora diesen Snapshot mit dem Namen `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster`*`-`*`timestamp`* im Snapshot-Bereich der AWS-Managementkonsole. Sie können auch den AWS CLI-Befehl `describe-db-cluster-snapshots` oder die `DescribeDBClusterSnapshots`-API-Operation verwenden, um Details zum Snapshot anzuzeigen.

Wenn Sie ein verwaltetes Failover initiieren, versucht Aurora auch, den Schreibverkehr über die hochverfügbare Aurora-Speicherschicht zu stoppen. Wir bezeichnen diesen Mechanismus als „Write-Fencing“. Wenn der Vorgang erfolgreich ist, gibt Aurora ein RDS-Ereignis aus, das Sie darüber informiert, dass Schreibvorgänge gestoppt wurden. Im unwahrscheinlichen Fall, dass mehrere AZ-Ausfälle in einer Region auftreten, ist es möglich, dass der Write-Fencing-Prozess nicht rechtzeitig erfolgreich durchgeführt wird. In diesem Fall gibt Aurora ein RDS-Ereignis aus, das Sie darüber informiert, dass das Zeitlimit für den Vorgang zum Stoppen von Schreibvorgängen überschritten wurde. Wenn der alte primäre Cluster im Netzwerk erreichbar ist, zeichnet Aurora diese Ereignisse dort auf. Wenn nicht, zeichnet Aurora die Ereignisse auf dem neuen primären Cluster auf. Weitere Informationen zu diesen Ereignissen finden Sie unter DB-Cluster-Ereignisse. Da ein Fencing (Umgrenzen) von Schreibvorgängen ein Best-Effort-Versuch ist, ist es möglich, dass Schreibvorgänge in der alten primären Region vorübergehend akzeptiert werden, was zu Split-Brain-Problemen führt.

Wir empfehlen, dass Sie die folgenden Aufgaben ausführen, bevor Sie ein Failover mit Aurora Global Database durchführen. Dadurch wird die Möglichkeit von Split-Brain-Problemen oder der Wiederherstellung nicht replizierter Daten aus dem Snapshot des alten primären Clusters minimiert.

- Schalten Sie Anwendungen offline, um zu verhindern, dass Schreibvorgänge an den primären Cluster von Aurora Global Database gesendet werden.
- Stellen Sie sicher, dass alle Anwendungen, die eine Verbindung zum primären DB-Cluster herstellen, den globalen Writer-Endpunkt verwenden. Dieser Endpunkt hat einen Wert, der auch dann gleich bleibt, wenn eine neue Region aufgrund einer Umstellung oder eines Failovers zum primären Cluster wird. Aurora implementiert zusätzliche Sicherheitsvorkehrungen, um die Möglichkeit von Datenverlusten bei Schreibvorgängen zu minimieren, die über den globalen Endpunkt übermittelt werden. Weitere Informationen zu globalen Writer-Endpunkten finden Sie unter Verbinden mit Amazon Aurora Global Database.
- Wenn Sie den globalen Writer-Endpunkt verwenden und Ihre Anwendungs- oder Netzwerkschichten DNS-Werte zwischenspeichern, setzen Sie die Gültigkeitsdauer (TTL) Ihres DNS-Caches auf einen niedrigen Wert, z B. auf 5 Sekunden. Auf diese Weise registriert Ihre Anwendung DNS-Änderungen schnell im globalen Writer-Endpunkt. Aurora versucht zwar, Schreibvorgänge in der alten primären Region zu blockieren, es ist jedoch nicht garantiert, dass die Aktion erfolgreich ist. Durch die Reduzierung der DNS-Cache-Lebensdauer wird die Wahrscheinlichkeit von Split-Brain-Problemen weiter minimiert. Alternativ können Sie nach dem RDS-Ereignis suchen, das Sie darüber informiert, wann Aurora DNS-Änderungen für den globalen Writer-Endpunkt beobachtet hat. Auf diese Weise können Sie überprüfen, ob Ihre Anwendung die DNS-Änderung auch registriert hat, bevor Sie den Schreibverkehr Ihrer Anwendung neu starten.
- Überprüfen Sie die Verzögerungszeiten für alle sekundären Aurora-DB-Cluster in der globalen Aurora-Datenbank. Durch die Auswahl der sekundären Region mit der geringsten Replikationsverzögerung kann der Datenverlust in der aktuell ausgefallenen primären Region minimiert werden.

  Verwenden Sie für alle Aurora-PostgreSQL-basierten Versionen globaler Datenbanken und für Aurora-MySQL-basierte globale Datenbanken ab Engine-Versionen 3.04.0 und höher oder 2.12.0 und höher Amazon CloudWatch, um die `AuroraGlobalDBRPOLag`-Metrik für alle sekundären DB-Cluster anzuzeigen. Zeigen Sie für niedrigere Nebenversionen von Aurora MySQL-basierten globalen Datenbanken stattdessen die `AuroraGlobalDBReplicationLag`-Metrik an. Diese Metriken geben an, wie weit (in Millisekunden) ein sekundärer Cluster gegenüber dem primären DB-Cluster verzögert ist.

  Weitere Informationen zu den CloudWatch-Metriken für Aurora finden Sie unter Metriken auf Cluster-Ebene für Amazon Aurora.

Während eines verwalteten Failovers wird der ausgewählte sekundäre DB-Cluster in seine neue Rolle als primärer Cluster hochgestuft. Er übernimmt jedoch nicht die verschiedenen Konfigurationsoptionen des primären DB-Clusters. Wenn die Konfiguration nicht übereinstimmt, kann dies Leistungsprobleme, Workload-Inkompatibilitäten und anderes anomales Verhalten zur Folge haben. Um solche Probleme zu vermeiden, empfehlen wir Ihnen, die Unterschiede zwischen den Clustern Ihrer globalen Aurora-Datenbank wie folgt aufzulösen:

- Konfigurieren einer Parametergruppe für Aurora-DB-Cluster für den neuen primären Cluster, falls erforderlich – Sie können Ihre Parametergruppen für Aurora-DB-Cluster für jeden Aurora-Cluster in Ihrer globalen Aurora-Datenbank separat konfigurieren. Wenn Sie also einen sekundären DB-Cluster zur Übernahme der primären Rolle hochstufen, kann die Parametergruppe des sekundären Clusters im Vergleich zum primären Cluster möglicherweise anders konfiguriert sein. Ist dies der Fall, ändern Sie die Parametergruppe des hochgestuften sekundären DB-Clusters so, dass sie den Einstellungen des primären Clusters entspricht. Um zu erfahren wie, siehe Ändern von Parametern für eine Aurora globale Datenbank.
- Konfigurieren Sie Überwachungstools und -optionen wie Amazon CloudWatch Events und Alarme – Konfigurieren Sie den hochgestuften DB-Cluster mit den gleichen Protokollierungsfunktionen, Alarmen usw. wie für die globale Datenbank erforderlich. Wie bei Parametergruppen wird die Konfiguration für diese Funktionen während des Failover-Prozesses nicht vom primären Cluster übernommen. Einige CloudWatch-Metriken, wie z. B. die Verzögerung bei der Replikation, sind nur für sekundäre Regionen verfügbar. Daher ändert ein Failover die Art und Weise, wie diese Metriken angezeigt und Alarme für sie eingerichtet werden, und es können Änderungen an allen vordefinierten Dashboards erforderlich sein. Weitere Informationen zur Überwachung von Aurora-DB-Clustern finden Sie unter Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch.
- Konfigurieren von Integrationen in andere AWS-Services – Wenn Ihre globale Aurora-Datenbank in andere AWS-Services wie AWS Secrets Manager, AWS Identity and Access Management, Amazon S3 und AWS Lambda integriert ist, müssen Sie sicherstellen, dass diese für den Zugriff aus sekundären Regionen konfiguriert sind. Weitere Informationen zur Integration globaler Aurora-Datenbanken in IAM, Amazon S3 und Lambda finden Sie unter Verwenden von globalen Amazon-Aurora-Datenbanken mit anderen AWS-Services. Weitere Informationen zu Secrets Manager finden Sie unter So automatisieren Sie die Replikation von Secrets in AWS Secrets Manager über verschiedene AWS-Regionen hinweg.

In der Regel übernimmt der gewählte sekundäre Cluster innerhalb weniger Minuten die primäre Rolle. Sobald die Writer-DB-Instance der neuen primären Region verfügbar ist, können Sie Ihre Anwendungen mit dieser verbinden und Ihre Workloads fortsetzen. Nachdem Aurora den neuen

primären Cluster hochgestuft hat, werden automatisch alle zusätzlichen Cluster der sekundären Region neu erstellt.

Da die globalen Aurora-Datenbanken die asynchrone Replikation verwenden, kann die Replikationsverzögerung in jeder sekundären Region variieren. Aurora baut diese sekundären Regionen neu auf, so dass sie genau dieselben Point-in-Time-Daten wie der neue Cluster der primären Region haben. Die Dauer der vollständigen Neuerstellungsaufgabe kann je nach Größe des Speichervolumens und der Entfernung zwischen den Regionen einige Minuten bis mehrere Stunden dauern. Wenn der Wiederaufbau der Cluster der sekundären Region aus der neuen primären Region abgeschlossen ist, stehen sie für den Lesezugriff zur Verfügung.

Sobald der neue primäre Writer hochgestuft und verfügbar ist, kann der Cluster der neuen primären Region Lese- und Schreibvorgänge für die globale Aurora-Datenbank abwickeln.

Wenn Sie den globalen Endpunkt verwenden, müssen Sie die Verbindungseinstellungen in Ihrer Anwendung nicht ändern. Stellen Sie sicher, dass die DNS-Änderungen weitergegeben wurden und dass Sie eine Verbindung herstellen und Schreibvorgänge auf dem neuen primären Cluster ausführen können. Anschließend können Sie den vollen Betrieb Ihrer Anwendung wieder aufnehmen.

Wenn Sie den globalen Endpunkt nicht verwenden, stellen Sie sicher, dass Sie den Endpunkt für Ihre Anwendung so ändern, dass er den Cluster-Endpunkt für den neu hochgestuften primären DB-Cluster verwendet. Wenn Sie die angegebenen Namen beim Erstellen der globalen Aurora-Datenbank akzeptiert haben, können Sie den Endpunkt ändern, indem Sie `-ro` aus der Endpunkt-Zeichenfolge des hochgestuften Clusters in Ihrer Anwendung entfernen.

Beispielsweise wird der Endpunkt des sekundären Clusters `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` zu `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`, wenn dieser Cluster zum primären Cluster hochgestuft wird.

Wenn Sie RDS-Proxy verwenden, stellen Sie sicher, dass Sie die Schreibvorgänge Ihrer Anwendung an den entsprechenden Lese-/Schreibendpunkt des Proxys umleiten, der dem neuen primären Cluster zugeordnet ist. Dieser Proxy-Endpunkt kann der Standardendpunkt oder ein benutzerdefinierter Lese-/Schreibendpunkt sein. Weitere Informationen finden Sie unter So funktionieren RDS-Proxy-Endpunkte mit globalen Datenbanken.

Um die ursprüngliche Topologie des globalen Datenbank-Clusters wiederherzustellen, überwacht Aurora die Verfügbarkeit der alten primären Region. Sobald diese Region fehlerfrei und wieder verfügbar ist, fügt Aurora sie automatisch wieder als sekundäre Region zum globalen Cluster

hinzu. Bevor das neue Speichervolume in der alten Primärregion erstellt wird, versucht Aurora, zum Zeitpunkt des Fehlers einen Snapshot des alten Speichervolumes zu erstellen. Auf diese Weise können Sie alle fehlenden Daten wiederherstellen. Wenn dieser Vorgang erfolgreich ist, erstellt Aurora einen Snapshot mit dem Namen `rds:unplanned-global-failover-`*`name-of-old-primary-DB-cluster`*`-`*`timestamp`*. Sie finden diesen Snapshot im Bereich Snapshots der AWS-Managementkonsole. Sie können diesen Snapshot auch in den Informationen sehen, die von der API-Operation DescribeDBClusterSnapshots zurückgegeben wird.

> **Note**
>
> Der Snapshot des alten Speichervolumes ist ein System-Snapshot, der der auf dem alten primären Cluster konfigurierten Aufbewahrungsfrist für Backups unterliegt. Um diesen Snapshot außerhalb des Aufbewahrungszeitraums zu speichern, können Sie ihn kopieren, um ihn als manuellen Snapshot zu speichern. Weitere Informationen zum Kopieren von Snapshots, einschließlich der Preise, finden Sie unter Kopieren eines DB-Cluster-Snapshots.

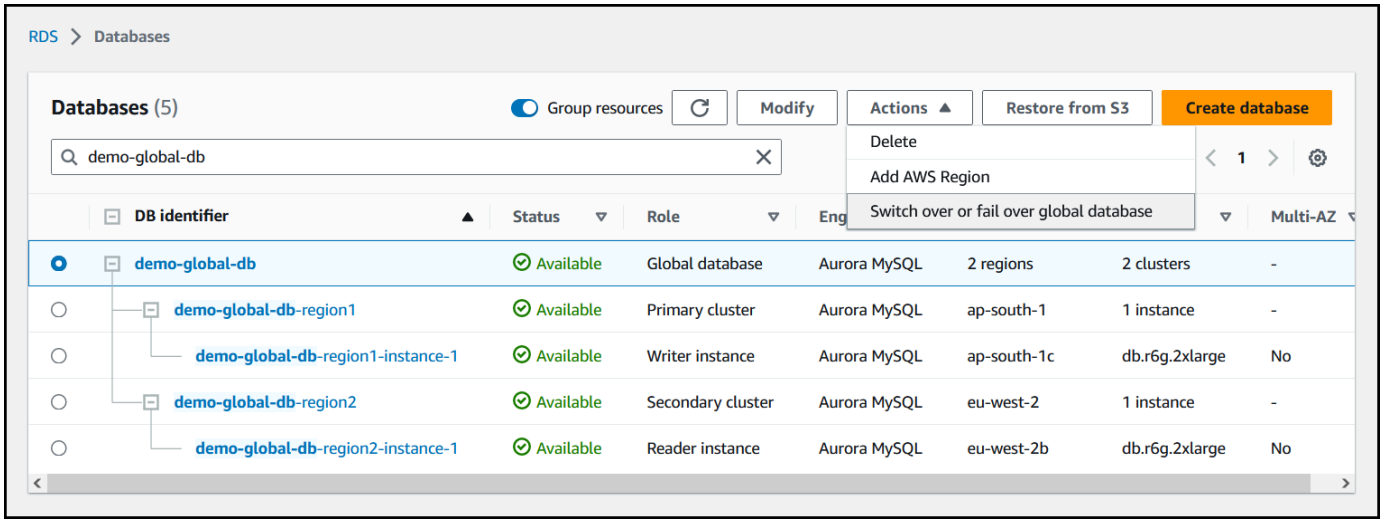
Nachdem die ursprüngliche Topologie wiederhergestellt ist, können Sie ein Failback Ihrer globalen Datenbank auf die ursprüngliche primäre Region durchführen, indem Sie eine Umstellung durchführen, wenn dies für Ihr Unternehmen und Ihren Workload am sinnvollsten ist. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Durchführen von Umstellungen für Amazon Aurora Global Databases.

Sie können ein Failover mit Aurora Global Datenbase mit der AWS-Managementkonsole, der AWS CLI oder der RDS-API ausführen.

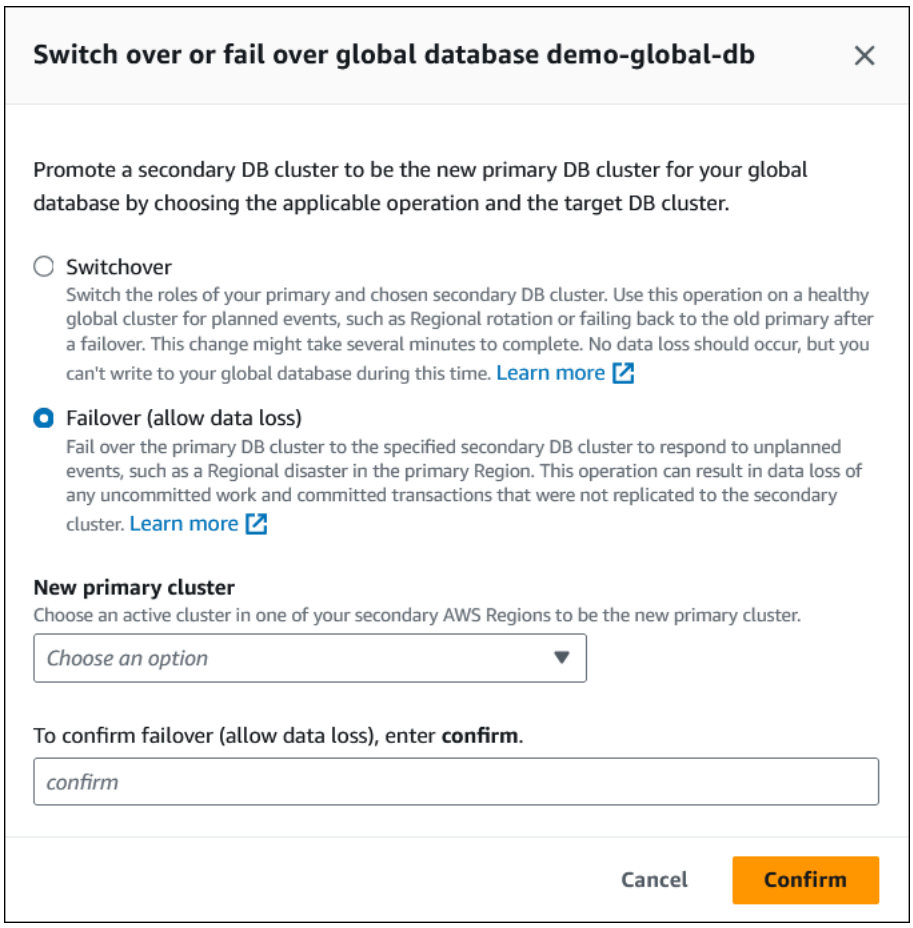
Konsole

So starten Sie den verwalteten Failover-Prozess in Ihrer globalen Aurora-Datenbank

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Datenbanken aus und suchen Sie die globale Aurora-Datenbank, für die Sie ein Failover ausführen möchten.
3. Wählen Sie im Menü Aktionen die Option Failover für globale Datenbank ausführen aus.

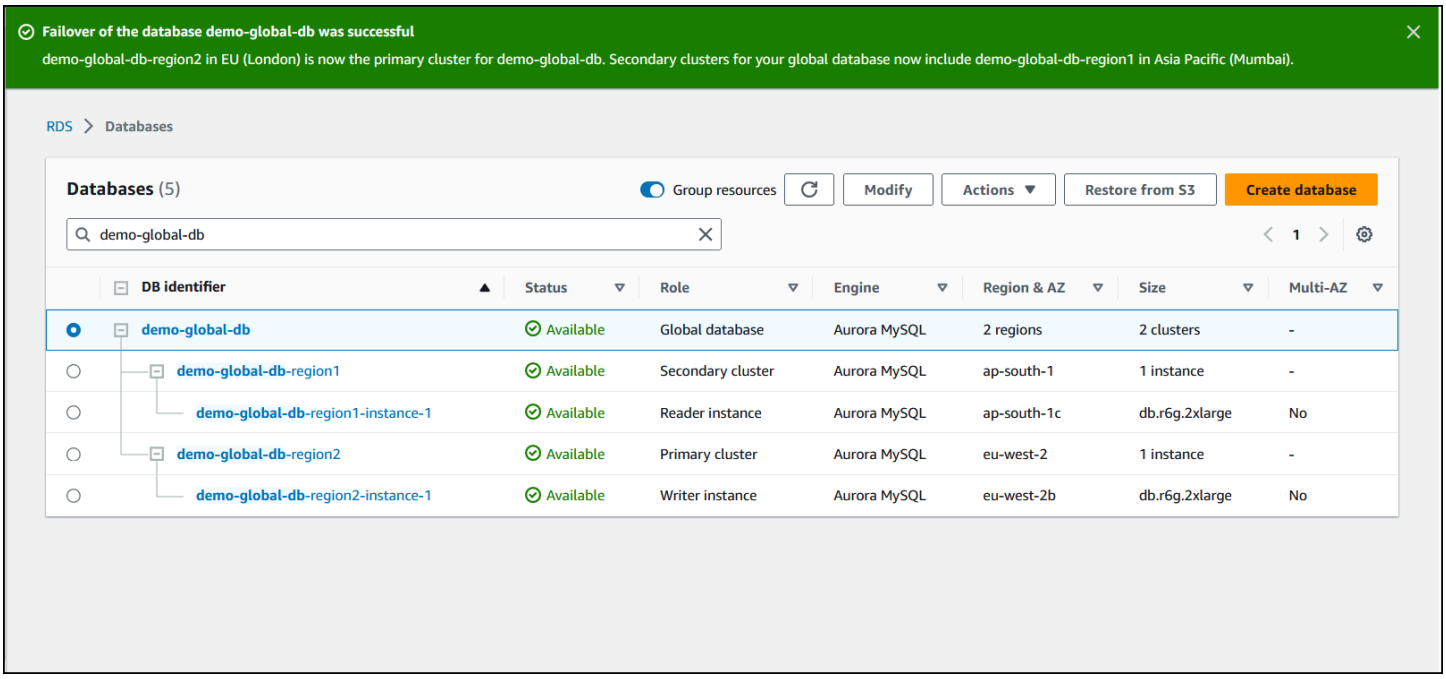


4. Wählen Sie Failover (Datenverlust zulassen).



5. Für Neuer primärer Cluster wählen Sie einen aktiven Cluster in einer Ihrer sekundären AWS-Regionen als neuen primären Cluster aus.
6. Geben Sie **`confirm`** ein, und wählen Sie dann Bestätigen.

Wenn das Failover abgeschlossen ist, werden die Aurora-DB-Cluster und ihr aktueller Status in der Liste Datenbanken angezeigt, wie im folgenden Bild zu sehen ist.



AWS CLI

So führen Sie das verwaltete Failover für eine globale Aurora-Datenbank durch

Verwenden Sie den CLI-Befehl `failover-global-cluster`, um ein Failover mit Aurora Global Database durchzuführen. Mit dem Befehl können Sie Werte für die folgenden Parameter übertragen.

- `--region` – Geben Sie die AWS-Region an, in der der sekundäre DB-Cluster ausgeführt wird, den Sie als neuen primären DB-Cluster für die globale Aurora-Datenbank verwenden möchten.
- `--global-cluster-identifier` – Geben Sie den Namen Ihrer globalen Aurora-Datenbank an.
- `--target-db-cluster-identifier` – Geben Sie den Amazon-Ressourcennamen (ARN) des Aurora-DB-Clusters an, den Sie als neuen primären Cluster für die globale Aurora-Datenbank hochstufen möchten.
- `--allow-data-loss` – Machen Sie dies ausdrücklich zu einer Failover- und nicht zu einer Umstellungs-Operation. Ein Failover-Vorgang kann zu Datenverlusten führen, wenn die asynchronen Replikationskomponenten das Senden aller replizierten Daten an die sekundäre Region nicht abgeschlossen haben.

Für Linux, macOS oder Unix:

```
aws rds --region region_of_selected_secondary \
   failover-global-cluster --global-cluster-identifier global_database_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

Für Windows:

```
aws rds --region region_of_selected_secondary ^
   failover-global-cluster --global-cluster-identifier global_database_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

RDS-API

Um ein Failover mit Aurora Global Database durchzuführen, führen Sie die API-Operation FailoverGlobalCluster aus.

Ausführen von manuellen geplanten Failovers für globale Aurora-Datenbanken

In einigen Szenarien können Sie den verwalteten Failover-Prozess möglicherweise nicht verwenden. Ein Beispiel ist etwa, wenn auf Ihrem primären und sekundären DB-Cluster keine kompatiblen Engine-Versionen ausgeführt werden. In diesem Fall können Sie diesem manuellen Prozess folgen, um ein Failover in Ihre sekundäre Zielregion durchzuführen.

> **Tip**
>
> Bevor Sie diesen Prozess verwenden, sollten Sie sich damit vertraut machen. Halten Sie einen Plan bereit, um bei den ersten Anzeichen eines regionsweiten Problems schnell handeln zu können. Sie können bereit sein, die sekundäre Region mit der geringsten Replikationsverzögerung zu identifizieren, indem Sie Amazon CloudWatch verwenden, um Verzögerungszeiten für sekundäre Cluster zu verfolgen. Testen Sie Ihren Plan, um zu überprüfen, ob Ihre Verfahren vollständig und genau sind, und stellen Sie sicher, dass die Mitarbeiter darin geschult sind, ein Failover für die Notfallwiederherstellung durchzuführen, bevor es wirklich erforderlich ist.

So führen Sie ein manuelles Failover auf einen sekundären Cluster nach einem ungeplanten Ausfall in der primären Region durch:

1. Beenden Sie die Ausgabe von DML-Anweisungen und anderen Schreibvorgängen auf dem primären Aurora-DB-Cluster in der AWS-Region mit dem Ausfall.
2. Identifizieren Sie einen Aurora-DB-Cluster aus einer sekundären AWS-Region, der als neuer primärer DB-Cluster verwendet werden soll. Wenn Sie zwei oder mehr sekundäre AWS-Regionen in Ihrer globalen Aurora-Datenbank haben, wählen Sie den sekundären Cluster mit der geringsten Replikationsverzögerung.
3. Trennen Sie Ihren ausgewählten sekundären DB-Cluster von der globalen Aurora-Datenbank.

   Das Entfernen eines sekundären DB-Clusters aus einer globalen Aurora-Datenbank stoppt sofort die Replikation vom primären zu diesem sekundären Cluster und stuft ihn als ein eigenständiges bereitgestelltes Aurora-DB-Cluster mit uneingeschränkter Lese-/Schreibfunktion ein. Alle anderen sekundären Aurora-DB-Cluster, die mit dem primären Cluster in der Region mit dem Ausfall verknüpft sind, sind weiterhin verfügbar und können Aufrufe von Ihrer Anwendung annehmen. Sie verbrauchen auch Ressourcen. Da Sie die globale Aurora-Datenbank neu erstellen, entfernen Sie die anderen sekundären DB-Cluster, bevor Sie die neue globale Aurora-Datenbank in den folgenden Schritten erstellen. Dadurch werden Dateninkonsistenzen zwischen den DB-Clustern in der globalen Aurora-Datenbank vermieden (split-brain-Probleme).

   Ausführliche Schritte zum Trennen finden Sie unter Entfernen eines Clusters aus einer Amazon Aurora Global Database.
4. Konfigurieren Sie Ihre Anwendung neu, um alle Schreibvorgänge mit ihrem neuen Endpunkt an diesen eigenständigen Aurora-DB-Cluster zu senden. Wenn Sie die angegebenen Namen beim Erstellen der globalen Aurora-Datenbank akzeptiert haben, können Sie den Endpunkt ändern, indem Sie `-ro` aus der Endpunkt-Zeichenfolge des Clusters in Ihrer Anwendung entfernen.

   Beispielsweise wird der Endpunkt `my-global.cluster-**ro**-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` des sekundären Clusters zu `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com`, wenn dieser Cluster von der globalen Aurora-Datenbank getrennt wird.

   Dieser Aurora-DB-Cluster wird zum primären Cluster einer neuen globalen Aurora-Datenbank, wenn Sie im nächsten Schritt beginnen Regionen hinzuzufügen.

   Wenn Sie RDS-Proxy verwenden, stellen Sie sicher, dass Sie die Schreibvorgänge Ihrer Anwendung an den entsprechenden Lese-/Schreibendpunkt des Proxys umleiten, der dem

neuen primären Cluster zugeordnet ist. Dieser Proxy-Endpunkt kann der Standardendpunkt oder ein benutzerdefinierter Lese-/Schreibendpunkt sein. Weitere Informationen finden Sie unter So funktionieren RDS-Proxy-Endpunkte mit globalen Datenbanken.

5. Fügen Sie dem DB-Cluster eine AWS-Region hinzu. Wenn Sie dies tun, beginnt der Replikationsprozess vom primären zum sekundären Cluster. Ausführliche Schritte zum Hinzufügen einer Region finden Sie unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank.
6. Fügen Sie nach Bedarf weitere AWS-Regionen hinzu, um die Topologie wieder zu erstellen, die zur Unterstützung Ihrer Anwendung erforderlich ist.

Stellen Sie sicher, dass Schreibvorgänge der Anwendung vor, während und nach diesen Änderungen an den richtigen Aurora-DB-Cluster gesendet werden. Dadurch werden Dateninkonsistenzen zwischen den DB-Clustern in der globalen Aurora-Datenbank vermieden (split-brain-Probleme).

Wenn Sie aufgrund eines Ausfalls in einer AWS-Region eine Neukonfiguration vorgenommen haben, können Sie diese AWS-Region auf primär zurücksetzen, nachdem der Ausfall behoben wurde. Dazu fügen Sie die alte AWS-Region Ihrer neuen globalen Datenbank hinzu und verwenden dann den Umstellungs-Prozess, um die Rolle zu wechseln. Ihre Aurora Global Database muss eine Version von Aurora PostgreSQL oder Aurora MySQL verwenden, die Umstellungen unterstützt. Weitere Informationen finden Sie unter Durchführen von Umstellungen für Amazon Aurora Global Databases.

Verwalten von RPOs für Aurora PostgreSQL–basierte globale Datenbanken

Mit einer Aurora PostgreSQL–basierten globalen Datenbank können Sie den Recovery Point Objective (RPO)-Wert für Ihre globale Aurora-Datenbank mithilfe des Parameters `rds.global_db_rpo` verwalten. RPO gibt die maximale Datenmenge an, die im Falle eines Ausfalls verloren gehen kann.

Wenn Sie eine RPO für Ihre Aurora PostgreSQL–basierte globale Datenbank festlegen, überwacht Aurora die RPO-Verzögerungszeit aller sekundären Cluster, um sicherzustellen, dass mindestens ein sekundärer Cluster innerhalb des angestrebten RPO-Bereichs liegt. Die RPO-Verzögerungszeit ist eine weitere zeitbasierte Metrik.

Der RPO wird verwendet, wenn Ihre Datenbank nach einem Failover den Betrieb in einer neuen AWS-Region wieder aufnimmt. Aurora bewertet RPO- und RPO-Verzögerungszeiten, um Transaktionen auf dem primären Netzwerk wie folgt zu übertragen (oder zu blockieren):

- Die Transaktion wird durchgeführt, wenn mindestens ein sekundärer DB-Cluster eine RPO-Verzögerungszeit hat, die kleiner ist als die RPO.
- Die Transaktion wird blockiert, wenn alle sekundären DB-Cluster RPO-Verzögerungszeiten haben, die größer sind als die RPO. Außerdem wird das Ereignis in der PostgreSQL-Protokolldatei erfasst und es werden Warteereignisse ausgegeben, die die blockierten Sessions anzeigen.

Wenn sich also alle sekundären Cluster hinter der Ziel-RPO befinden, pausiert Aurora die Transaktionen im primären Cluster, bis mindestens einer der sekundären Cluster den Rückstand aufholt. Pausierte Transaktionen werden fortgesetzt und übergeben, sobald die Verzögerungszeit mindestens eines sekundären DB-Clusters geringer ist als die RPO. Das Ergebnis ist, dass keine Transaktionen übertragen werden können, bis die RPO erreicht ist.

Der `rds.global_db_rpo`-Parameter ist dynamisch. Wenn Sie nicht möchten, dass alle Schreibtransaktionen angehalten werden, bis die Verzögerung ausreichend abnimmt, können Sie ihn schnell zurücksetzen. In diesem Fall erkennt Aurora die Änderung und implementiert sie nach einer kurzen Verzögerung.

> **Important**
>
> Bei einer globalen Datenbank mit nur zwei AWS-Regionen empfehlen wir, den Standardwert des Parameters `rds.global_db_rpo` in der Parametergruppe der sekundären Region beizubehalten. Andernfalls könnte ein Failover aufgrund eines Verlusts der primären AWS-Region dazu führen, dass Aurora Transaktionen unterbricht. Warten Sie stattdessen, bis Aurora den Neuaufbau des Clusters in der alten ausgefallenen AWS-Region abgeschlossen hat, bevor Sie diesen Parameter ändern, um ein maximales RPO zu erzwingen.

Wenn Sie diesen Parameter wie folgt festlegen, können Sie auch die von ihm generierten Metriken überwachen. Sie können dies tun, indem Sie `psql` oder ein anderes Tool verwenden, um den primären DB-Cluster der globalen Aurora-Datenbank abzufragen und detaillierte Informationen über den Betrieb Ihrer Aurora PostgreSQL–basierten globalen Datenbank zu erhalten. Um zu erfahren wie, siehe Überwachen von Aurora-PostgreSQL-basierten globalen Datenbanken.

Themen

- Festlegen des Recovery Point Objective
- Anzeigen des Recovery Point Objective (Zeitraums zwischen zwei Datensicherungen)
- Deaktivieren des Recovery Point Objective

Festlegen des Recovery Point Objective

Der Parameter `rds.global_db_rpo` steuert die RPO-Einstellung für eine PostgreSQL-Datenbank. Dieser Parameter wird von Aurora PostgreSQL unterstützt. Gültige Werte für `rds.global_db_rpo` liegen zwischen 20 und 2.147.483.647 Sekunden (68 Jahre). Wählen Sie einen realistischen Wert, der Ihren Geschäftsanforderungen und Ihrem Anwendungsfall entspricht. Wenn Sie beispielsweise bis zu 10 Minuten für Ihre RPO festlegen möchten, dann setzen Sie den Wert auf 600.

Sie können diesen Wert für Ihre Aurora-PostgreSQL-basierte globale Datenbank festlegen, indem Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API verwenden.

Konsole

So legen Sie den RPO fest:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie den primären Cluster Ihrer globalen Aurora-Datenbank und öffnen Sie die Registerkarte Konfiguration, um die DB-Cluster-Parametergruppe zu suchen. Die Standardparametergruppe für ein primäres DB-Cluster mit Aurora PostgreSQL 11.7 ist beispielsweise `default.aurora-postgresql11`.

   Parametergruppen können nicht direkt bearbeitet werden. Stattdessen führen Sie die folgenden Schritte aus:

   - Erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe, wobei Sie die entsprechende Standardparametergruppe als Ausgangspunkt verwenden. So erstellen Sie beispielsweise eine benutzerdefinierte DB-Cluster-Parametergruppe basierend auf `default.aurora-postgresql11`.
   - Legen Sie in Ihrer benutzerdefinierten DB-Parametergruppe den Wert des Parameters rds.global_db_rpo fest, der Ihrem Anwendungsfall entspricht. Gültige Werte liegen zwischen 20 Sekunden und dem maximalen ganzzahligen Wert von 2.147.483.647 (68 Jahre).
   - Wenden Sie die geänderte DB-Cluster-Parametergruppe auf Ihr Aurora-DB-Cluster an.

Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

AWS CLI

Zum Festlegen des Parameters `rds.global_db_rpo` verwenden Sie den CLI-Befehl modify-db-cluster-parameter-group. Geben Sie im Befehl den Namen der Parametergruppe Ihres primären Clusters und die Werte für den RPO-Parameter an.

Im folgenden Beispiel wird die RPO für die primäre DB-Cluster-Parametergruppe namens auf 600 Sekunden (10 Minuten) festgelegt `my_custom_global_parameter_group`.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Für Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

RDS-API

Um den Parameter `rds.global_db_rpo` zu ändern, verwenden Sie die Amazon-RDS-API-Operation ModifyDBClusterParameterGroup.

Anzeigen des Recovery Point Objective (Zeitraums zwischen zwei Datensicherungen)

Die Recovery Point Objective (RPO) einer globalen Datenbank wird im `rds.global_db_rpo`-Parameter für jeden DB-Cluster gespeichert. Sie können eine Verbindung zum Endpunkt für den sekundären Cluster herstellen, den Sie anzeigen möchten, und mit `psql` die Instance nach diesem Wert abfragen.

```
db-name=>show rds.global_db_rpo;
```

Wenn dieser Parameter nicht festgelegt ist, gibt die Abfrage Folgendes zurück:

```
rds.global_db_rpo
-------------------
```

```
 -1
(1 row)
```

Diese nächste Antwort stammt von einem sekundären DB-Cluster mit einer RPO-Einstellung von einer Minute.

```
rds.global_db_rpo
-------------------
 60
(1 row)
```

Sie können auch die CLI verwenden, um herauszufinden, ob in einem der Aurora-DB-Cluster `rds.global_db_rpo` aktiviert ist, indem Sie die CLI verwenden, um die Werte aller `user`-Parameter für den Cluster zu erhalten.

Für Linux, macOS oder Unix:

```
aws rds describe-db-cluster-parameters \
 --db-cluster-parameter-group-name lab-test-apg-global \
 --source user
```

Für Windows:

```
aws rds describe-db-cluster-parameters ^
 --db-cluster-parameter-group-name lab-test-apg-global *
 --source user
```

Der Befehl gibt für alle `user`-Parameter, die keine `default-engine`- oder `system`-DB-Cluster-Parameter sind, eine Ausgabe zurück, die der folgenden Ausgabe ähnelt.

```
{
    "Parameters": [
        {
            "ParameterName": "rds.global_db_rpo",
            "ParameterValue": "60",
            "Description": "(s) Recovery point objective threshold, in seconds, that
 blocks user commits when it is violated.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "20-2147483647",
```

```
            "IsModifiable": true,
            "ApplyMethod": "immediate",
            "SupportedEngineModes": [
                "provisioned"
            ]
        }
    ]
}
```

Weitere Informationen zum Anzeigen von Parametern der Clusterparametergruppe finden Sie unter Anzeigen von Parameterwerten für eine DB-Cluster Parametergruppe in Amazon Aurora.

Deaktivieren des Recovery Point Objective

Um den RPO zu deaktivieren, setzen Sie den `rds.global_db_rpo`-Parameter zurück. Sie können Parameter über die AWS-Managementkonsole, die AWS CLI oder die RDS-API zurücksetzen.

Konsole

So deaktivieren Sie den RPO:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Parameter groups (Parametergruppen) aus.
3. Wählen Sie in der Liste Ihre primäre DB-Cluster-Parametergruppe aus.
4. Wählen Sie Parameter bearbeiten aus.
5. Wählen Sie das Feld neben dem Parameter rds.global_db_rpo aus.
6. Klicken Sie auf Reset (Zurücksetzen).
7. Wenn auf dem Bildschirm Reset parameters in DB parameter group (Parameter in DB-Parametergruppe zurücksetzen) angezeigt wird, wählen Sie Reset parameters (Parameter zurücksetzen) aus.

Weitere Informationen zum Zurücksetzen eines Parameters mit der Konsole finden Sie unter Ändern von Parametern in einer DB-Cluster-Parametergruppe in Amazon Aurora.

AWS CLI

Um den Parameter `rds.global_db_rpo` zurückzusetzen, verwenden Sie den Befehl reset-db-cluster-parameter-group.

Für Linux, macOS oder Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name global_db_cluster_parameter_group \
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

Für Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name global_db_cluster_parameter_group ^
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

RDS-API

Um den Parameter `rds.global_db_rpo` zurückzusetzen, verwenden Sie die Amazon-RDS-API-Operation ResetDBClusterParameterGroup.

Regionsübergreifende Resilienz für sekundäre Cluster globaler Datenbanken

Die Aurora-PostgreSQL-Versionen 16.6, 15.10, 14.15, 13.18, 12.22 oder höher und die Aurora-MySQL-Versionen 3.09 oder höher enthalten Verbesserungen in Bezug auf die Verfügbarkeit, die es Lesereplikaten der sekundären Region ermöglichen, die Servicekontinuität bei ungeplanten Ereignissen wie Hardwarefehlern, Netzwerkunterbrechungen zwischen AWS-Regionen, großen Datenübertragungen zwischen den Clustern und anderen Ereignissen aufrechtzuerhalten.

Obwohl die Lesereplikate für Ihre Anwendungsanfragen verfügbar bleiben, kann sich die Verzögerung bei der Replikation weiter erhöhen, bis das Problem des ungeplanten Ereignisses behoben ist. Sie können die Verzögerung zwischen primären und sekundären Clustern mithilfe der `AuroraGlobalDBProgressLag`-CloudWatch-Metrik überwachen. Um die durchgehende Verzögerung zu messen, einschließlich aller Verzögerungen zwischen dem Cluster-Volume und den DB-Instances des sekundären Clusters, fügen Sie die Werte der CloudWatch-Metriken `AuroraGlobalDBProgressLag` und `AuroraReplicaLag` hinzu. Weitere Informationen zu Metriken finden Sie unter Amazon Aurora-Referenz für Metriken.

Die Leseverfügbarkeit der globalen Datenbank für Aurora MySQL und früheren Versionen von Aurora PostgreSQL kann während dieser ungeplanten Ereignisse beeinträchtigt sein.

Weitere Informationen zu neuen Funktionen in Aurora PostgreSQL 16.6, 15.10, 14.15, 13.18 und 12.22 finden Sie unter PostgreSQL 16.6 in den Aurora-PostgreSQL-Versionshinweisen

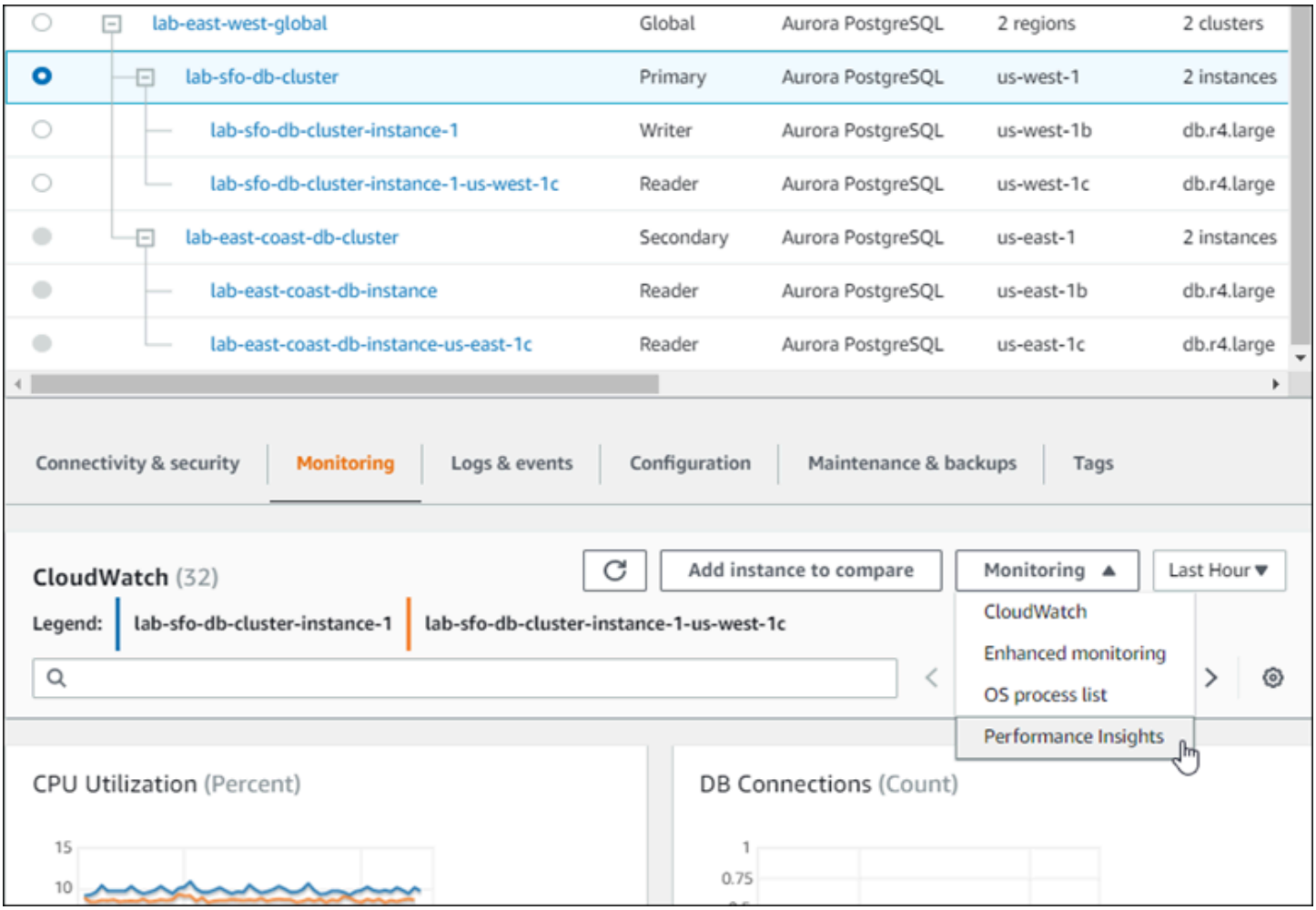
Weitere Informationen zu neuen Funktionen in den Aurora-MySQL-Versionen 3.09 und höher finden Sie unter Aktualisierungen der Datenbank-Engine für Version 3 von Amazon Aurora MySQL in den Aurora-MySQL-Versionshinweisen.

Überwachen einer globalen Amazon-Aurora-Datenbank

Wenn Sie die Aurora-DB-Cluster erstellen, aus denen Ihre Aurora globale Datenbank besteht, können Sie viele Optionen auswählen, mit denen Sie die Leistung Ihres DB-Clusters überwachen können. Diese Optionen umfassen Folgendes:

- Amazon RDS Performance Insights – Es ermöglicht Performance-Schema in der zugrunde liegenden Aurora-Datenbank-Engine. Weitere Informationen über Performance Insights und Aurora globale Datenbanken finden Sie unter Überwachung einer Amazon Aurora globalen Datenbank mit Amazon RDS Performance Insights.
- „Enhanced Monitoring“ (Erweiterte Überwachung) – Es generiert Metriken für die Prozess- oder Thread-Nutzung auf der CPU. Weitere Informationen zur erweiterten Überwachung finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).
- Amazon CloudWatch Logs – Es publiziert festgelegte Protokolltypen nach CloudWatch Logs. Fehlerprotokolle werden standardmäßig veröffentlicht. Sie können jedoch andere für Ihre Aurora-Datenbank-Engine spezifische Protokolle auswählen.
  - Für Aurora MySQL–basierte Aurora-DB-Cluster können Sie das Audit-Log, das allgemeine Protokoll und das langsame Abfrage-Log exportieren.
  - Für Aurora PostgreSQL–basierte Aurora-DB-Cluster können Sie das PostgreSQL-Protokoll exportieren.
- Für Aurora-MySQL-basierte globale Datenbanken können Sie bestimmte `information_schema`-Tabellen verwenden, um den Status Ihrer globalen Aurora-Datenbank und deren Instances zu überprüfen. Um zu erfahren wie dies geht, vgl. Überwachung der Aurora-MySQL-basierten globalen Datenbanken.
- Für Aurora-PostgreSQL-basierte globale Datenbanken können Sie bestimmte Funktionen verwenden, um den Status Ihrer Aurora globalen Datenbank und ihrer Instances zu überprüfen. Um zu erfahren wie dies geht, vgl. Überwachen von Aurora-PostgreSQL-basierten globalen Datenbanken.

Der folgende Screenshot zeigt einige der Optionen, die auf der Registerkarte Überwachung eines primären Aurora-DB-Clusters in einer Aurora globalen Datenbank verfügbar sind.



Weitere Informationen finden Sie unter Überwachung von Metriken in einem Amazon-Aurora-Cluster.

Überwachung einer Amazon Aurora globalen Datenbank mit Amazon RDS Performance Insights

Sie können Amazon RDS Performance Insights für Ihre globalen Aurora-Datenbanken verwenden. Sie aktivieren diese Funktion einzeln für jeden Aurora-DB-Cluster in Ihrer Aurora globalen Datenbank. Dazu wählen Sie Performance Insights aktivieren im Abschnitt Zusätzliche Konfiguration der Seite „Datenbank erstellen“. Oder Sie können Ihre Aurora-DB-Cluster ändern, um diese Funktion zu verwenden, nachdem sie in Betrieb sind. Sie können Performance Insights für jeden Cluster, der zur Aurora globalen Datenbank gehört, aktivieren bzw. deaktivieren.

Die von Performance Insights erstellten Berichte gelten für jeden Cluster in der globalen Datenbank. Wenn Sie einer globalen Aurora-Datenbank, die Performance Insights bereits verwendet, eine neue

sekundäre AWS-Region hinzufügen, müssen Sie Performance Insights im neu hinzugefügten Cluster aktivieren. Die Performance-Insights-Einstellung wird nicht von der vorhandenen globalen Datenbank übernommen.

Sie können beim Anzeigen der Performance-Insights-Seite für eine DB-Instance, die einer globalen Datenank angefügt ist, zwischen AWS-Regionen wechseln. Möglicherweise werden unmittelbar nach dem Wechsel der AWS-Regionen allerdings keine Leistungsdaten angezeigt. Auch wenn die DB-Instances in jeder AWS-Region identische Namen tragen, weicht die zugeordnete Performance-Insights-URL für jede DB-Instance ab. Wählen Sie nach dem Wechsel der AWS-Regionen den Namen der DB-Instance erneut im Performance-Insights-Navigationsbereich aus.

Für DB-Instances, die einer globalen Datenbank zugeordnet sind, können sich in jeder AWS-Region andere Faktoren auf die Leistung auswirken. Die DB-Instances können in jeder AWS-Region z. B. eine andere Kapazität aufweisen.

Weitere Informationen zum Verwenden von Performance Insights finden Sie unter Überwachung mit Performance Insights auf .

Überwachung von globalen Aurora-Datenbanken mithilfe von Datenbank-Aktivitätsstreams

Durch die Verwendung von Datenbank-Aktivitätsstreams können Sie Alarme für die Prüfung von Aktivitäten in den DB-Clustern Ihrer globalen Datenbank überwachen und einstellen. Sie starten einen Datenbank-Aktivitätsstream separat auf jedem DB-Cluster. Jeder Cluster liefert Auditdaten innerhalb seiner eigenen AWS-Region an seinen eigenen Kinesis-Stream. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.

Überwachung der Aurora-MySQL-basierten globalen Datenbanken

Um den Status einer auf Aurora MySQL basierenden globalen Datenbank abzurufen, fragen Sie die Tabellen information_schema.aurora_global_db_status und information_schema.aurora_global_db_instance_

> **Note**
>
> Die Tabellen `information_schema.aurora_global_db_status` und `information_schema.aurora_global_db_instance_status` sind nur mit globalen Aurora-MySQL-Datenbanken ab Version 3.04.0 verfügbar.

So überwachen Sie eine Aurora-MySQL-basierte globale Datenbank

1. Stellen Sie mithilfe eines MySQL-Clients eine Verbindung zum primären Cluster-Endpunkt der globalen Datenbank her. Weitere Informationen zum Herstellen einer Verbindung finden Sie unter Verbinden mit Amazon Aurora Global Database.
2. Fragen Sie die Tabelle `information_schema.aurora_global_db_status` in einem mysql-Befehl ab, um die primären und sekundären Volumes aufzulisten. Diese Abfrage gibt die Verzögerungszeiten der sekundären DB-Cluster der globalen Datenbank zurück, wie im folgenden Beispiel dargestellt.

```
mysql> select * from information_schema.aurora_global_db_status;
```

```
AWS_REGION | HIGHEST_LSN_WRITTEN | DURABILITY_LAG_IN_MILLISECONDS |
 RPO_LAG_IN_MILLISECONDS | LAST_LAG_CALCULATION_TIMESTAMP | OLDEST_READ_VIEW_TRX_ID
-----------+---------------------+--------------------------------
+------------------------+---------------------------------
+------------------------
us-east-1  |           183537946 |                              0   |
      0 |  1970-01-01 00:00:00.000000     |               0
us-west-2  |           183537944 |                              428 |
      0 |  2023-02-18 01:26:41.925000     |               20806982
(2 rows)
```

   Die Ausgabe enthält eine Zeile für jeden DB-Cluster der globalen Datenbank, die die folgenden Spalten enthält:

   - AWS_REGION – die AWS-Region, in der sich dieser DB-Cluster befindet. Tabellen, die AWS-Regionen nach Engine aufführen, finden Sie unter Verfügbarkeit in Regionen.
   - HIGHEST_LSN_WRITTEN – die höchste Log-Sequenznummer (LSN), die derzeit auf diesem DB-Cluster geschrieben wird.

     Eine Log-Sequenznummer (LSN) ist eine eindeutige fortlaufende Nummer, die einen Datensatz im Datenbank-Transaktionsprotokoll identifiziert. LSNs werden so angeordnet, dass eine größere LSN eine spätere Transaktion darstellt.
   - DURABILITY_LAG_IN_MILLISECONDS – die Differenz der Zeitstempelwerte zwischen `HIGHEST_LSN_WRITTEN` in einem sekundären DB-Cluster und `HIGHEST_LSN_WRITTEN` im primären DB-Cluster. Der Wert ist immer 0 im primären DB-Cluster der globalen Aurora-Datenbank.

- RPO_LAG_IN_MILLISECONDS – die Verzögerung des Zeitraums im Hinblick auf Recovery Point Objective (RPO). Die RPO-Verzögerung ist die Zeit, die benötigt wird, bis die letzte Benutzertransaktion COMMIT auf einem sekundären DB-Cluster gespeichert wird, nachdem sie auf dem primären DB-Cluster einer globalen Aurora-Datenbank abgelegt wurde. Der Wert ist immer 0 im primären DB-Cluster der globalen Aurora-Datenbank.

  Einfach ausgedrückt berechnet diese Metrik das Recovery Point Objective für jeden DB-Cluster voon Aurora MySQL in der globalen Aurora-Datenbank, d. h., wie viele Daten bei einem Ausfall verloren gehen könnten. Wie die Verzögerung wird auch RPO zeitlich gemessen.
- LAST_LAG_CALCULATION_TIMESTAMP – der Zeitstempel mit Angabe des Zeitpunkts, zu dem die Werte für `DURABILITY_LAG_IN_MILLISECONDS` und `RPO_LAG_IN_MILLISECONDS` zuletzt berechnet wurden. Ein Zeitwert wie `1970-01-01 00:00:00+00` bedeutet, dass dies der primäre DB-Cluster ist.
- OLDEST_READ_VIEW_TRX_ID – die ID der ältesten Transaktion, bis zu der die Writer-DB-Instance Daten löschen kann.

3. Fragen Sie die Tabelle `information_schema.aurora_global_db_instance_status` ab, um alle sekundären DB-Instances sowohl für den primären DB-Cluster als auch für sekundäre DB-Cluster aufzulisten.

```
mysql> select * from information_schema.aurora_global_db_instance_status;
```

```
SERVER_ID            |              SESSION_ID              | AWS_REGION
 | DURABLE_LSN | HIGHEST_LSN_RECEIVED | OLDEST_READ_VIEW_TRX_ID |
 OLDEST_READ_VIEW_LSN | VISIBILITY_LAG_IN_MSEC
---------------------+--------------------------------------+------------
+-------------+----------------------+------------------------
+----------------------+------------------------
ams-gdb-primary-i2   | MASTER_SESSION_ID                    | us-east-1  |
 183537698   |                    0 |                      0 |
 0 |                      0
ams-gdb-secondary-i1 | cc43165b-bdc6-4651-abbf-4f74f08bf931 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                      0
ams-gdb-secondary-i2 | 53303ff0-70b5-411f-bc86-28d7a53f8c19 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                    677
```

```
ams-gdb-primary-i1   | 5af1e20f-43db-421f-9f0d-2b92774c7d02 | us-east-1  |
 183537697   |            183537698 |                20806930 |
 183537691 |                    21
(4 rows)
```

Die Ausgabe enthält eine Zeile für jede DB-Instance der globalen Datenbank, die die folgenden Spalten enthält:

- SERVER_ID – die Server-ID für die DB-Instance.
- SESSION_ID – eine eindeutige ID für die aktuelle Sitzung. Der Wert `MASTER_SESSION_ID` bezeichnet die (primäre) Writer-DB-Instance.
- AWS_REGION – die AWS-Region, in der sich diese DB-Instance befindet. Tabellen, die AWS-Regionen nach Engine aufführen, finden Sie unter Verfügbarkeit in Regionen.
- DURABLE_LSN – die LSN, die im Speicher als dauerhaft definiert wurde.
- HIGHEST_LSN_RECEIVED – die höchste LSN, die die DB-Instance von der DB-Writer-Instance empfangen hat.
- OLDEST_READ_VIEW_TRX_ID – die ID der ältesten Transaktion, bis zu der die Writer-DB-Instance Daten löschen kann.
- OLDEST_READ_VIEW_LSN – die älteste LSN, die von der DB-Instance zum Lesen aus dem Speicher verwendet wird.
- VISIBILITY_LAG_IN_MSEC – für Reader im primären DB-Cluster, wie weit diese DB-Instance der Writer-DB-Instance in Millisekunden hinterherhinkt. Für Reader in einem sekundären DB-Cluster, wie weit diese DB-Instance dem sekundären Volume in Millisekunden hinterherhinkt.

Um zu sehen, wie sich diese Werte im Laufe der Zeit verändern, betrachten Sie den folgenden Transaktionsblock, in dem eine Tabelleneinfügung eine Stunde dauert.

```
mysql> BEGIN;
mysql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
mysql> COMMIT;
```

In einigen Fällen kann es nach der `BEGIN`-Anweisung zu einer Netzwerktrennung zwischen dem primären DB-Cluster und dem sekundären DB-Cluster kommen. In diesem Fall beginnt der Wert DURABILITY_LAG_IN_MILLISECONDS des sekundäre DB-Clusters zu steigen. Am Ende der `INSERT`-Anweisung beträgt der Wert DURABILITY_LAG_IN_MILLISECONDS 1 Stunde. Der Wert RPO_LAG_IN_MILLISECOND ist jedoch 0, da alle Benutzerdaten, die zwischen dem primären

DB-Cluster und dem sekundären DB-Cluster übertragen werden, immer noch dieselben sind. Sobald die `COMMIT`-Anweisung abgeschlossen ist, steigt der Wert RPO_LAG_IN_MILLISECONDSWert steigt.

Überwachen von Aurora-PostgreSQL-basierten globalen Datenbanken

Verwenden Sie die Funktionen `aurora_global_db_status` und `aurora_global_db_instance_status`, um den Status einer Aurora-PostgreSQL-basierten globalen Datenbank anzuzeigen.

> **Note**
>
> Die Funktionen `aurora_global_db_status` und `aurora_global_db_instance_status` werden nur von Aurora PostgreSQL unterstützt.

So überwachen Sie eine Aurora PostgreSQL-basierte globale Datenbank

1. Stellen Sie mithilfe eines PostgreSQL-Dienstprogramms wie psql eine Verbindung zum primären Cluster-Endpunkt der globalen Datenbank her. Weitere Informationen zum Herstellen einer Verbindung finden Sie unter Verbinden mit Amazon Aurora Global Database.
2. Sie verwenden die Funktion `aurora_global_db_status` in einem psql-Befehl, um die primären und sekundären Volumes aufzulisten. Dadurch werden die Verzögerungszeiten der sekundären DB-Cluster der globalen Datenbank angezeigt.

   ```
   postgres=> select * from aurora_global_db_status();
   ```

   ```
   aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec |
    last_lag_calculation_time  | feedback_epoch | feedback_xmin
   ------------+---------------------+------------------------+-----------------
   +----------------------------+----------------+--------------
   us-east-1  |         93763984222 |                     -1 |              -1 |
    1970-01-01 00:00:00+00     |              0 |              0
   us-west-2  |         93763984222 |                    900 |            1090 |
    2020-05-12 22:49:14.328+00 |              2 |    3315479243
   (2 rows)
   ```

   Die Ausgabe enthält eine Zeile für jeden DB-Cluster der globalen Datenbank, die die folgenden Spalten enthält:

- aws_region Die AWS-Region, in der sich dieser DB-Cluster befindet. Tabellen, die AWS-Regionen nach Engine aufführen, finden Sie unter Verfügbarkeit in Regionen.
- highest_lsn_written – Die höchste Log-Sequenznummer (LSN), die derzeit auf diesem DB-Cluster geschrieben wird.

  Eine Log-Sequenznummer (LSN) ist eine eindeutige fortlaufende Nummer, die einen Datensatz im Datenbank-Transaktionsprotokoll identifiziert. LSNs werden so angeordnet, dass eine größere LSN eine spätere Transaktion darstellt.
- durability_lag_in_msec – Die Zeitstempeldifferenz zwischen der höchsten Log-Sequenznummer, die auf einem sekundären DB-Cluster (`highest_lsn_written`) geschrieben wurde, und der `highest_lsn_written` auf dem primären DB-Cluster.
- rpo_lag_in_msec – Die Verzögerung des Zeitraums zwischen zwei Datensicherungen (RPO). Diese Verzögerung ist die Zeitdifferenz zwischen dem letzten Benutzertransaktions-Commit, der auf einem sekundären DB-Cluster gespeichert ist, und dem letzten Benutzertransaktions-Commit, der auf dem primären DB-Cluster gespeichert ist.
- last_lag_calculation_time – Der Zeitstempel mit Angabe des Zeitpunkts, zu dem die Werte für `durability_lag_in_msec` und `rpo_lag_in_msec` zuletzt berechnet wurden.
- feedback_epoch – Die Epoche, die ein sekundärer DB-Cluster beim Erzeugen von Hot-Standby-Informationen verwendet.

  Hot-Standby bedeutet, dass ein DB-Cluster eine Verbindung herstellen und Abfragen durchführen kann, während sich der Server im Wiederherstellungs- oder Standby-Modus befindet. Hot-Standby-Feedbacks sind Informationen über den DB-Cluster, wenn dieser sich im Hot-Standby-Modus befindet. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation zu Hot Standby.
- feedback_xmin – Die minimale (älteste) aktive Transaktions-ID, die von einem sekundären DB-Cluster verwendet wird.

3. Verwenden Sie die Funktion `aurora_global_db_instance_status`, um alle sekundären DB-Instances sowohl für den primären DB-Cluster als auch für sekundäre DB-Cluster aufzulisten.

```
postgres=> select * from aurora_global_db_instance_status();
```

```
server_id                                    |              session_id
  | aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
---------------------------------------------+-------------------------------------
+-----------+-------------+-----------------+---------------+--------------
+----------------------+-----------------------
apg-global-db-rpo-mammothrw-elephantro-1-n1 | MASTER_SESSION_ID
 | us-east-1  | 93763985102 |                 |               |              |
                    |
apg-global-db-rpo-mammothrw-elephantro-1-n2 | f38430cf-6576-479a-b296-dc06b1b1964a
 | us-east-1  | 93763985099 |      93763985102 |             2 |    3315479243 |
        93763985095 |                     10
apg-global-db-rpo-elephantro-mammothrw-n1   | 0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
 | us-west-2  | 93763985095 |      93763985099 |             2 |    3315479243 |
        93763985089 |                   1017
(3 rows)
```

Die Ausgabe enthält eine Zeile für jede DB-Instance der globalen Datenbank, die die folgenden Spalten enthält:

- server_id – Die Server-ID für die DB-Instance.
- session_id – Eine eindeutige ID für die aktuelle Sitzung.
- aws_region Die AWS-Region, in der sich diese DB-Instance befindet. Tabellen, die AWS-Regionen nach Engine aufführen, finden Sie unter Verfügbarkeit in Regionen.
- durable_lsn – Der LSN wurde im Speicher dauerhaft gemacht.
- highest_lsn_rcvd – Die höchste LSN, die die DB-Instance von der DB-Writer-Instance empfangen hat.
- feedback_epoch – Die Epoche, die die DB-Instance beim Erzeugen der Hot-Standby-Informationen verwendet.

  Hot standby bedeutet, dass eine DB-Instance eine Verbindung herstellen und Abfragen durchführen kann, während sich der Server im Wiederherstellungs- oder Standby-Modus befindet. Hot-Standby-Feedbacks sind Informationen über die DB-Instance, wenn diese sich im Hot-Standby-Modus befindet. Weitere Informationen finden Sie in der PostgreSQL-Dokumentation zu Hot Standby.
- feedback_xmin – Die minimale (älteste) aktive Transaktions-ID, die von der DB-Instance verwendet wird.

- oldest_read_view_lsn – Die älteste LSN, die von der DB-Instance zum Lesen aus dem Speicher verwendet wird.
- visibility_lag_in_msec – Wie stark diese DB-Instance gegenüber der DB-Schreiber-Instance zeitlich verzögert ist.

Um zu sehen, wie sich diese Werte im Laufe der Zeit verändern, betrachten Sie den folgenden Transaktionsblock, in dem eine Tabelleneinfügung eine Stunde dauert.

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
psql> COMMIT;
```

In einigen Fällen kann es nach der `BEGIN`-Anweisung zu einer Netzwerktrennung zwischen dem primären DB-Cluster und dem sekundären DB-Cluster kommen. Wenn dies der Fall ist, beginnt sich der `durability_lag_in_msec`-Wert des sekundären DB-Clusters zu erhöhen. Am Ende der `INSERT`-Anweisung beträgt der `durability_lag_in_msec`-Wert 1 Stunde. Der Wert `rpo_lag_in_msec` ist jedoch 0, da alle Benutzerdaten, die zwischen dem primären DB-Cluster und dem sekundären DB-Cluster übertragen werden, immer noch dieselben sind. Sobald die `COMMIT`-Anweisung abgeschlossen ist, erhöht sich der `rpo_lag_in_msec`-Wert.

Blue/Green Bereitstellungen für Amazon Aurora Global Database verwenden

Amazon Blue/Green RDS-Bereitstellungen bieten die Möglichkeit, Datenbankänderungen sicher zu testen. Für Ihre globale Datenbank ermöglichen blue/green Bereitstellungen die Durchführung von Upgrades und Wartungsarbeiten mit minimalen Ausfallzeiten. Sie können eine vollständig verwaltete Staging-Umgebung (grün) erstellen, die Ihre gesamte Produktionsumgebung (blau) widerspiegelt, einschließlich des primären Clusters und aller zugehörigen sekundären Regionen in mehreren AWS Regionen. Die Staging-Umgebung entspricht Ihrem Produktions-Setup, sodass Sie Änderungen zuverlässig testen können, bevor Sie zur neuen Umgebung wechseln. Während des gesamten Prozesses sorgen blue/green Bereitstellungen dafür, dass die Staging- und Produktionsumgebung synchron bleiben. Wenn Sie bereit sind, Ihre Staging-Umgebung zur neuen Produktionsumgebung zu machen, führen Sie einen Switchover durch. blue/green Bei diesem Vorgang werden Ihre primären und alle sekundären Regionen auf die grüne Umgebung umgestellt, wobei die Ausfallzeiten in der Regel unter einer Minute liegen. Der Service benennt Cluster, Instances und Endpoints automatisch um, sodass sie der ursprünglichen Produktionsumgebung entsprechen, sodass Ihre Anwendungen

ohne Konfigurationsänderungen auf die neue Produktionsumgebung zugreifen können und der Betriebsaufwand minimiert wird.

Themen

- Vorteile der Verwendung von Blue/Green Bereitstellungen mit Aurora Global Database
- So funktionieren Blue/Green Bereitstellungen mit Aurora Global Database

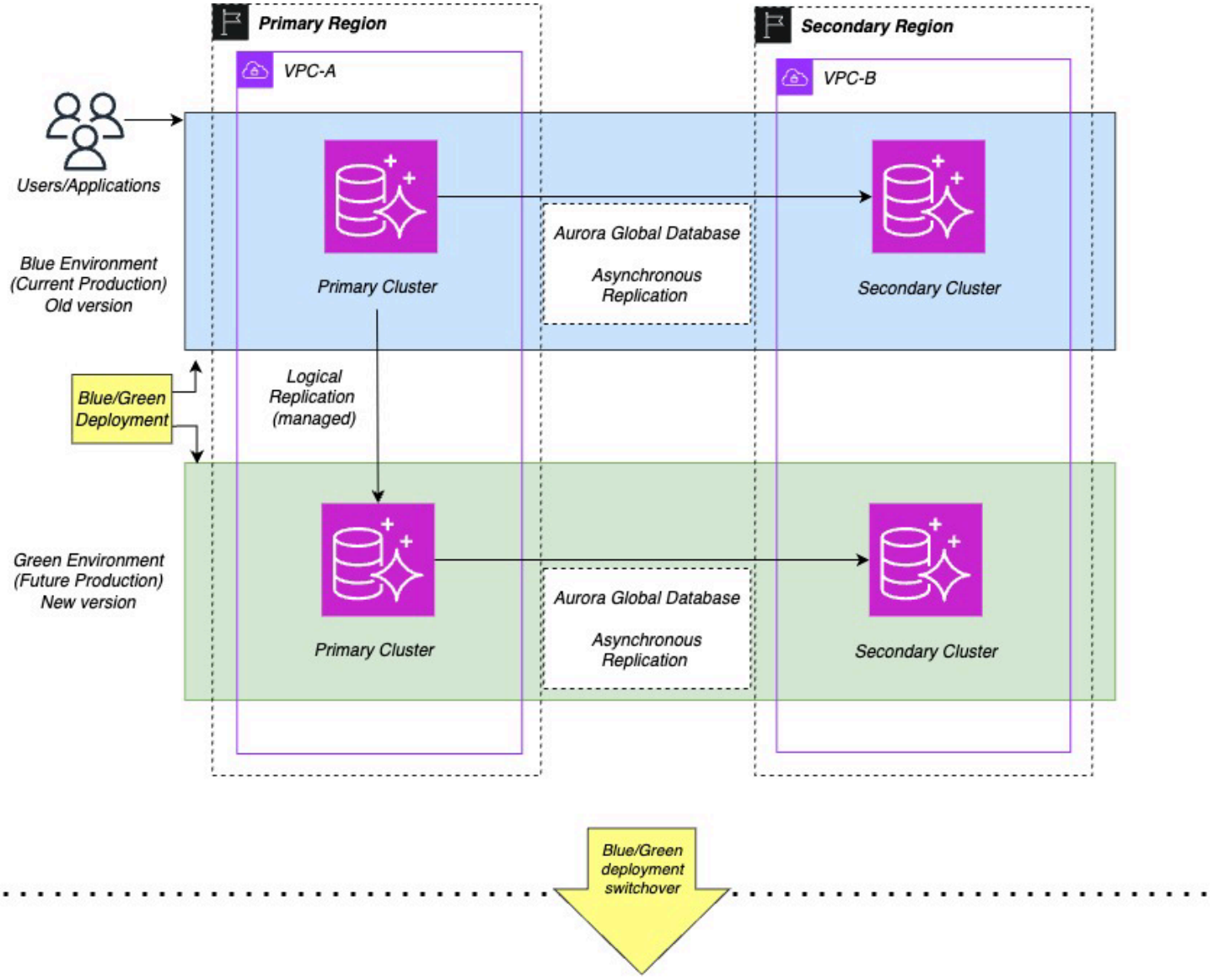
Vorteile der Verwendung von Blue/Green Bereitstellungen mit Aurora Global Database

- Führen Sie Hauptversionen, Nebenversionen und Systemupdates, einschließlich Datenbank-Patches und Betriebssystem-Upgrades, auf Aurora Global Databases durch und sorgen Sie dabei für minimale Ausfallzeiten.
- Erstellen Sie eine produktionsbereite (grüne) Staging-Umgebung sowohl in den primären als auch in den sekundären Regionen der globalen Datenbank, um neuere Datenbankfunktionen zu testen und zu implementieren.
- Sicheres Switchover mithilfe integrierter Switchover-Leitplanken mit Ausfallzeiten in der Regel unter einer Minute, je nach Arbeitslast.
- Behält die Disaster-Recovery-Funktionen während des Blue/Green Switchover-Prozesses bei und ermöglicht so ein globales Datenbank-Failover während des Switchovers. Blue/Green
- Ihr Datenverkehr wird an die neue Produktionsumgebung weitergeleitet, ohne dass Änderungen an der Anwendung erforderlich sind.

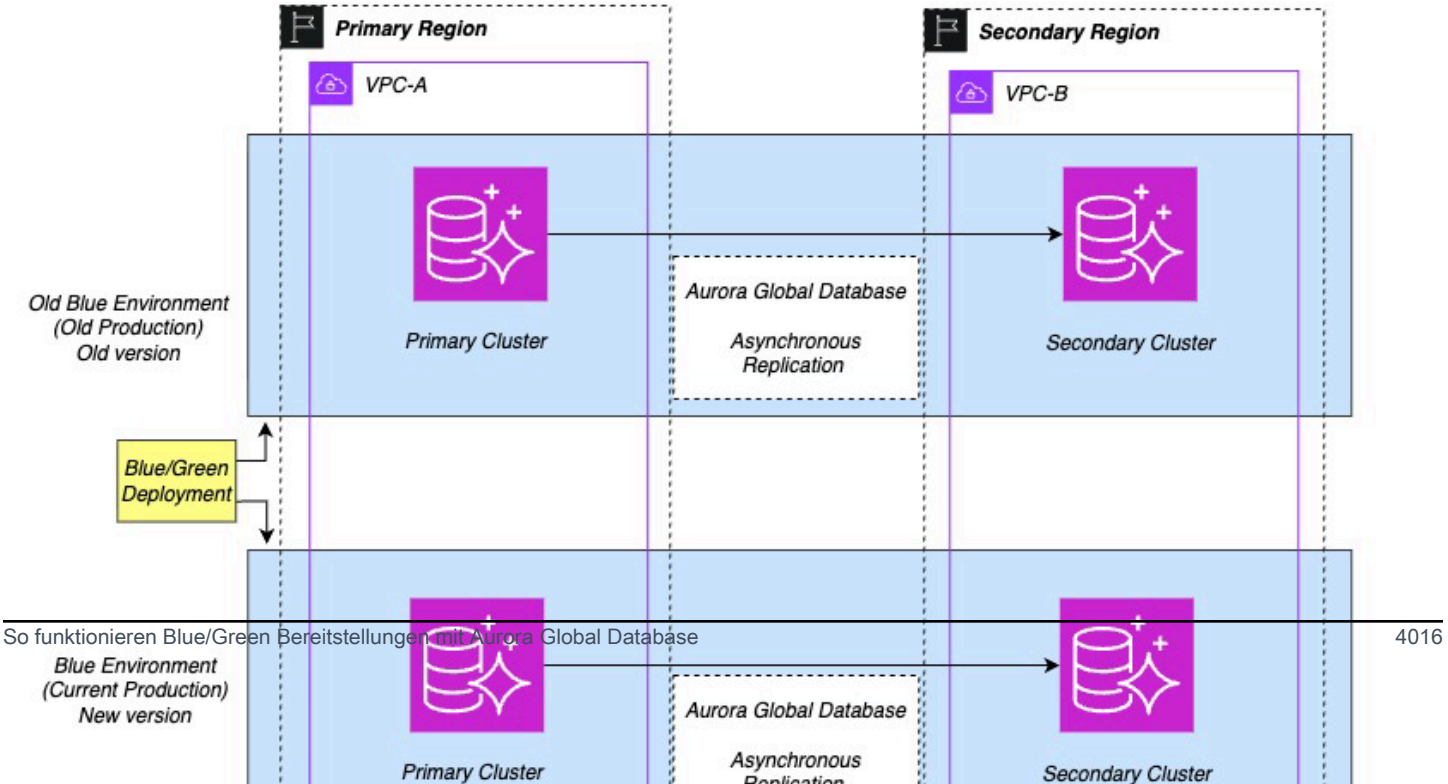
So funktionieren Blue/Green Bereitstellungen mit Aurora Global Database

Einzelheiten zum Erstellen, Anzeigen, Wechseln und Löschen eines Blue/Green Deployments finden Sie unterVerwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates. Sie können dies für Upgrades von Haupt- oder Nebenversionen, zur Verbesserung der Datenbankleistung durch Parameteraktualisierungen und zur Einführung neuer Datenbankfunktionen bei minimaler Ausfallzeit verwenden.

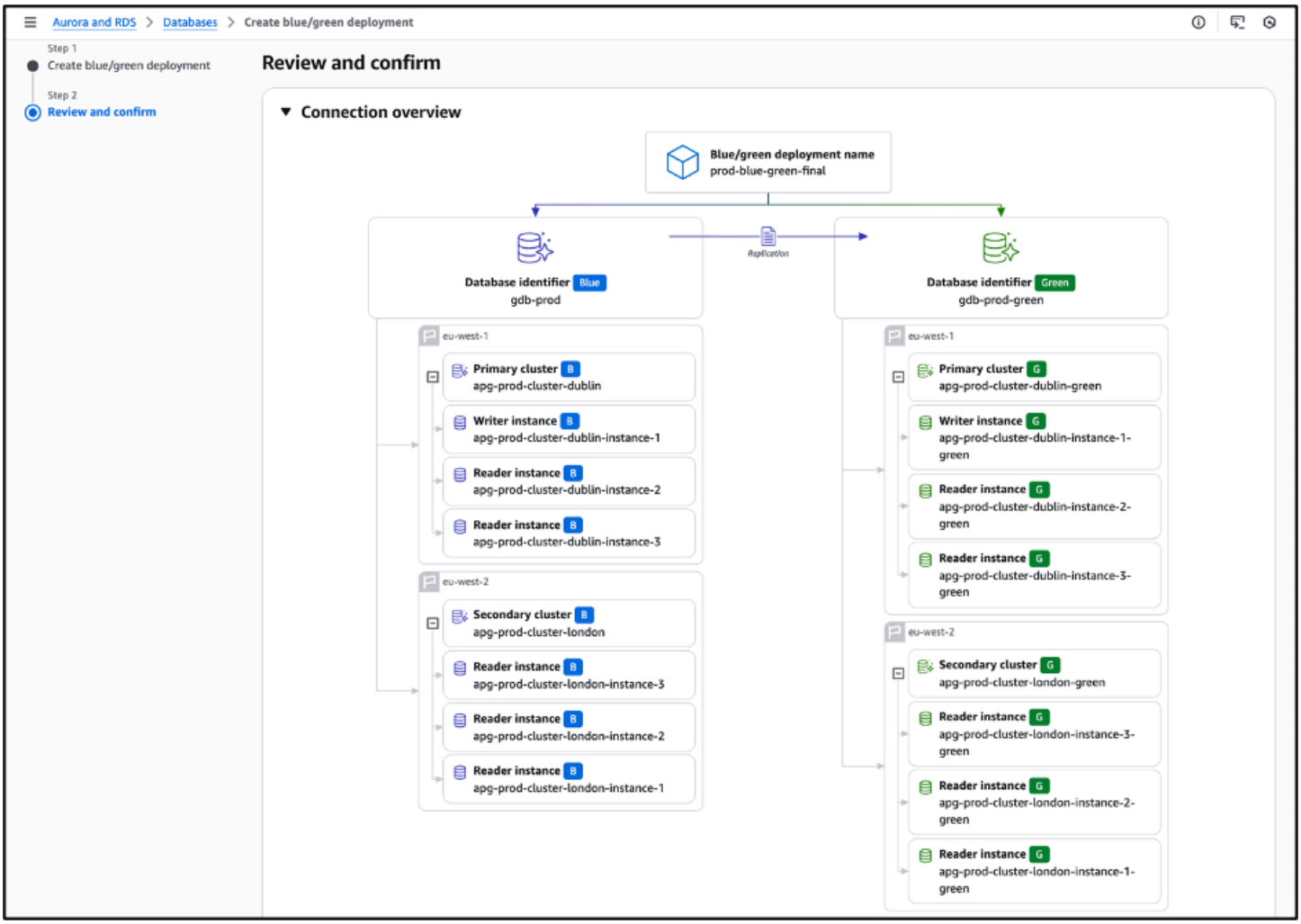
Im Folgenden wird dargestellt, wie ein blue/green Einsatz für Aurora Global Database mit einer sekundären Region vor und nach einem blue/green Switchover aussieht.


Primary Region
VPC-A
Secondary Region
VPC-B
Users/Applications
Blue Environment
(Current Production)
Old version
Primary Cluster
Aurora Global Database
Asynchronous
Replication
Secondary Cluster
Blue/Green
Deployment
Logical
Replication
(managed)
Green Environment
(Future Production)
New version
Primary Cluster
Aurora Global Database
Asynchronous
Replication
Secondary Cluster

Blue/Green
deployment
switchover


Primary Region
VPC-A
Secondary Region
VPC-B
Old Blue Environment
(Old Production)
Old version
Primary Cluster
Aurora Global Database
Asynchronous
Replication
Secondary Cluster
Blue/Green
Deployment
Blue Environment
(Current Production)
New version
Primary Cluster
Aurora Global Database
Asynchronous
Replication
Secondary Cluster

Sie können ein blue/green Deployment von der Hauptregion Ihrer globalen Datenbank aus erstellen. Wählen Sie die Engine-Konfigurationen wie Haupt- oder Nebenversion der Engine, DB-Parametergruppe und DB-Cluster-Parametergruppe für die grüne Umgebung aus. Amazon RDS kopiert die Topologie der blauen Umgebung für die grüne Umgebung. Eine visuelle Darstellung in AWS-Managementkonsole ist wie unten dargestellt.



> **Note**
>
> Globales Failover wird während eines blue/green Switchovers unterstützt, aber globales Switchover wird während eines Switchovers nicht unterstützt. blue/green

Wenn Sie während eines blue/green RDS-Switchovers einen globalen Failover einleiten, wird die Zielregion automatisch zur blauen Umgebung zurückgesetzt oder zur grünen Umgebung weitergeleitet, bevor der globale Failover stattfindet.

Informationen zum Erstellen, Anzeigen, Wechseln und Löschen von blue/green Bereitstellungen finden Sie unter. Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates Folgen Sie demselben Arbeitsablauf für globale Datenbanken, wobei die spezifischen Anweisungen in den entsprechenden Schritten aufgeführt sind.

Verwenden von globalen Amazon-Aurora-Datenbanken mit anderen AWS-Services

Sie können Ihre globalen Aurora-Datenbanken mit anderen AWS-Services wie Amazon S3 und AWS Lambda verwenden. Dies erfordert, dass alle Aurora-DB-Cluster in Ihrer globalen Datenbank die gleichen Privilegien, externen Funktionen usw. in den jeweiligen AWS-Regionen aufweisen. Da ein schreibgeschützter sekundärer Aurora-DB-Cluster in einer Aurora globalen Datenbank zur Rolle des primären Clusters hochgestuft werden kann, empfehlen wir, dass Sie im Voraus Schreibrechte für alle Aurora-DB-Cluster für alle Dienste einrichten, die Sie mit Ihrer Aurora globalen Datenbank verwenden möchten.

Die folgenden Verfahren fassen die für jeden AWS-Service zu ergreifenden Maßnahmen zusammen.

So rufen Sie AWS Lambda-Funktionen aus einer globalen Aurora-Datenbank auf

1. Führen Sie für alle Aurora-Cluster, aus denen die globale Aurora-Datenbank besteht, die unter beschriebenen Schritte au Aufrufen einer Lambda-Funktion aus einem Amazon Aurora MySQL-DB-Cluster.
2. Legen Sie für jeden Cluster in der Aurora globalen Datenbank den (ARN) der neuen IAM (IAM) -Rolle fest.
3. Verknüpfen Sie die in Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben erstellte Rolle mit allen Clustern in der globalen Aurora-Datenbank, um Datenbankbenutzern in einer globalen Aurora-Datenbank das Aufrufen von Lambda-Funktionen zu erlauben.
4. Konfigurieren Sie jeden Cluster in der globalen Aurora-Datenbank so, dass ausgehende Verbindungen zu Lambda zugelassen werden. Detaillierte Anweisungen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

So laden Sie Daten aus Amazon S3

1. Führen Sie für alle Aurora-Cluster, aus denen die globale Aurora-Datenbank besteht, die unter beschriebenen Schritte au Laden von Daten in einen Amazon Aurora MySQL-DB-Cluster aus Textdateien in einem Amazon S3-Bucket.
2. Legen Sie bei jedem Aurora-Cluster in der globalen Datenbank für den Amazon-Ressourcennamen (ARN) der neuen IAM-Rolle entweder den Cluster-Parameter `aurora_load_from_s3_role` oder `aws_default_s3_role` fest Wenn für `aurora_load_from_s3_role` keine IAM-Rolle angegeben wird, verwendet Aurora die unter `aws_default_s3_role` angegebene IAM-Rolle.
3. Um Datenbankbenutzern in einer globalen Aurora-Datenbank den Zugriff auf Amazon S3 zu gewähren, verknüpfen Sie die in Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben erstellte Rolle mit jedem Aurora-Cluster in der globalen Datenbank.
4. Konfigurieren Sie jeden Aurora-Cluster in der globalen Datenbank so, dass ausgehende Verbindungen zu S3 zugelassen werden. Detaillierte Anweisungen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

So speichern Sie abgefragte Daten auf Amazon S3

1. Führen Sie für alle Aurora-Cluster, aus denen die globale Aurora-Datenbank besteht, die unter Speichern von Daten aus einem Amazon Aurora MySQL-DB-Cluster in Textdateien in einem Amazon S3-Bucket oder Exportieren von Daten aus einem/einer Aurora PostgreSQL-DB-Cluster zu Amazon S3 beschriebenen Verfahren durch.
2. Legen Sie bei jedem Aurora-Cluster in der globalen Datenbank für den Amazon-Ressourcennamen (ARN) der neuen IAM-Rolle entweder den Cluster-Parameter `aurora_select_into_s3_role` oder `aws_default_s3_role` fest Wenn für `aurora_select_into_s3_role` keine IAM-Rolle angegeben wird, verwendet Aurora die unter `aws_default_s3_role` angegebene IAM-Rolle.
3. Um Datenbankbenutzern in einer globalen Aurora-Datenbank den Zugriff auf Amazon S3 zu gewähren, verknüpfen Sie die in Erstellen einer IAM-Rolle, um Amazon Aurora den Zugriff auf AWS-Services zu erlauben erstellte Rolle mit jedem Aurora-Cluster in der globalen Datenbank.
4. Konfigurieren Sie jeden Aurora-Cluster in der globalen Datenbank so, dass ausgehende Verbindungen zu S3 zugelassen werden. Detaillierte Anweisungen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.

Aktualisieren einer globalen Datenbank von Amazon Aurora

Das Upgrade einer globalen Aurora-Datenbank folgt den gleichen Verfahren wie das Upgrade von Aurora-DB-Clustern. Im Folgenden sind jedoch einige wichtige Unterschiede aufgeführt, die Sie beachten müssen, bevor Sie den Prozess starten.

Wir empfehlen, den primären und den sekundären DB-Cluster auf dieselbe Version zu aktualisieren. Sie können ein verwaltetes regionsübergreifendes Datenbank-Failover für eine globale Aurora-Datenbank nur durchführen, wenn die primären und sekundären DB-Cluster dieselben Engine-Haupt- und Nebenversionen sowie dasselbe Patch-Level haben. Die Patch-Level können je nach Nebenversion der Engine unterschiedlich sein. Weitere Informationen finden Sie unter Patch-Level-Kompatibilität für verwaltete regionsübergreifende Umstellungen und Failover.

Hauptversions-Upgrades

Wenn Sie ein Hauptversions-Upgrade einer Amazon Aurora Global Database durchführen, aktualisieren Sie den globalen Datenbank-Cluster statt der einzelnen darin enthaltenen Cluster.

Informationen zum Aktualisieren einer globalen Aurora-PostgreSQL-Datenbank auf eine höhere Hauptversion finden Sie unter Hauptversions-Upgrades für globale Datenbanken.

> **Note**
>
> Mit einer globalen Aurora-Datenbank, die auf Aurora PostgreSQL basiert, können Sie kein Hauptversions-Upgrade der Aurora-DB-Engine durchführen, wenn die Recovery Point Objective (RPO)-Funktion aktiviert ist. Weitere Informationen über RPO-Funktion finden Sie unter Verwalten von RPOs für Aurora PostgreSQL–basierte globale Datenbanken.

Informationen zum Aktualisieren einer globalen Aurora-MySQL-Datenbank auf eine höhere Hauptversion finden Sie unter In-Situ-Hauptversions-Upgrades für globale Datenbanken.

> **Note**
>
> Bei einer globalen Aurora-Datenbank, die auf Aurora MySQL basiert, können Sie ein direktes Upgrade von Aurora-MySQL-Version 2 auf Version 3 nur durchführen, wenn Sie den `lower_case_table_names`-Parameter auf den Standardwert setzen und Ihre globale Datenbank neu starten.

Um ein Hauptversions-Upgrade auf Aurora MySQL Version 3 bei Verwendung von `lower_case_table_names` durchzuführen, gehen Sie wie folgt vor:

1. Entfernen Sie alle sekundären Regionen aus dem globalen Cluster. Führen Sie die Schritte unter Entfernen eines Clusters aus einer Amazon Aurora Global Database aus.
2. Aktualisieren Sie die Engine-Version der primären Region zu Aurora-MySQL-Version 3. Führen Sie die Schritte unter Erläuterung der Durchführung eines direkten Upgrades aus.
3. Fügen Sie dem globalen Cluster sekundäre Regionen hinzu. Führen Sie die Schritte unter Hinzufügen einer AWS-Region zu einer globalen Amazon-Aurora-Datenbank aus.

Sie können stattdessen auch die Snapshot-Wiederherstellungsmethode verwenden. Weitere Informationen finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.

Unterversion-Upgrades

Bei einem Unterversion-Upgrade auf einer globalen Aurora-Datenbank aktualisieren Sie alle sekundären Cluster, bevor Sie den primären Cluster aktualisieren.

Informationen zum Aktualisieren einer globalen Aurora-PostgreSQL-Datenbank auf eine kleinere Hauptversion finden Sie unter So führen Sie Upgrades von Nebenversionen durch und wenden Patches an. Informationen zum Aktualisieren einer globalen Aurora-MySQL-Datenbank auf eine Unterversion finden Sie unter Upgrade von Aurora MySQL durch Ändern der Engine-Version.

Bevor Sie die Aktualisierung durchführen, lesen Sie die folgenden Hinweise:

- Eine Aktualisierung der Unterversion eines sekundären Clusters hat keinerlei Auswirkungen auf die Verfügbarkeit oder Nutzung des primären Clusters.
- Ein sekundärer Cluster muss über mindestens eine DB-Instance verfügen, um ein Unterversions-Upgrade durchzuführen.
- Wenn Sie eine globale Aurora-MySQL-Datenbank auf Version 2.11.* aktualisieren, müssen Sie Ihre primären und sekundären DB-Cluster auf exakt dieselbe Version aktualisieren, einschließlich des Patch-Levels.
- Wenn Sie eine globale Aurora-PostgreSQL-Datenbank aktualisieren, müssen Sie Ihre primären und sekundären DB-Cluster auf exakt dieselbe Version aktualisieren, einschließlich des Patch-Levels. Um das Patch-Level zu aktualisieren, wenden Sie alle ausstehenden Wartungsaktionen auf den sekundären Cluster an.

- Um verwaltete regionsübergreifende Umstellungen oder Failover zu unterstützen, müssen Sie Ihre primären und sekundären DB-Cluster möglicherweise auf genau dieselbe Version aktualisieren, einschließlich des Patch-Levels. Diese Anforderung gilt für Aurora MySQL und einige Aurora-PostgreSQL-Versionen. Eine Liste der Versionen, die Umstellungen und Failover zwischen Clustern mit unterschiedlichen Patch-Leveln ermöglichen, finden Sie unter Patch-Level-Kompatibilität für verwaltete regionsübergreifende Umstellungen und Failover.

Patch-Level-Kompatibilität für verwaltete regionsübergreifende Umstellungen und Failover

Wenn Ihre globale Aurora-Datenbank eine der folgenden Engine-Nebenversionen ausführt, können Sie verwaltete regionsübergreifende Umstellungen oder Failover durchführen, auch wenn die Patch-Level Ihrer primären und sekundären DB-Cluster nicht übereinstimmen. Bei Engine-Nebenversionen, die niedriger sind als die in dieser Liste aufgeführten, müssen Ihre primären und sekundären DB-Cluster dieselben Haupt- und Nebenversionen sowie Patch-Level ausführen, um verwaltete regionsübergreifende Umstellungen und Failover durchführen zu können. Sehen Sie sich unbedingt die Versionsinformationen und die Hinweise in der folgenden Tabelle an, wenn Sie Upgrades für Ihren primären Cluster und/oder Ihre sekundären Cluster planen.

> **Note**
>
> Bei manuellen regionsübergreifenden Failovern können Sie den Failover-Vorgang ausführen, solange auf dem sekundären Ziel-DB-Cluster dieselbe Engine-Haupt- und -Nebenversion wie auf dem primären DB-Cluster ausgeführt wird. In diesem Fall müssen die Patch-Level nicht übereinstimmen.
>
> Wenn Ihre Engine-Versionen identische Patch-Level erfordern, können Sie das Failover manuell durchführen, indem Sie die Schritte unter Ausführen von manuellen geplanten Failovers für globale Aurora-Datenbanken ausführen.

Datenbank-Engine	Engine-Nebenversionen	Hinweise
Aurora MySQL	Keine Nebenversionen	Keine der Nebenversionen von Aurora MySQL unterstützt verwaltete regionsübergreifende Umstellungen und Failover, wenn die Patch-Level

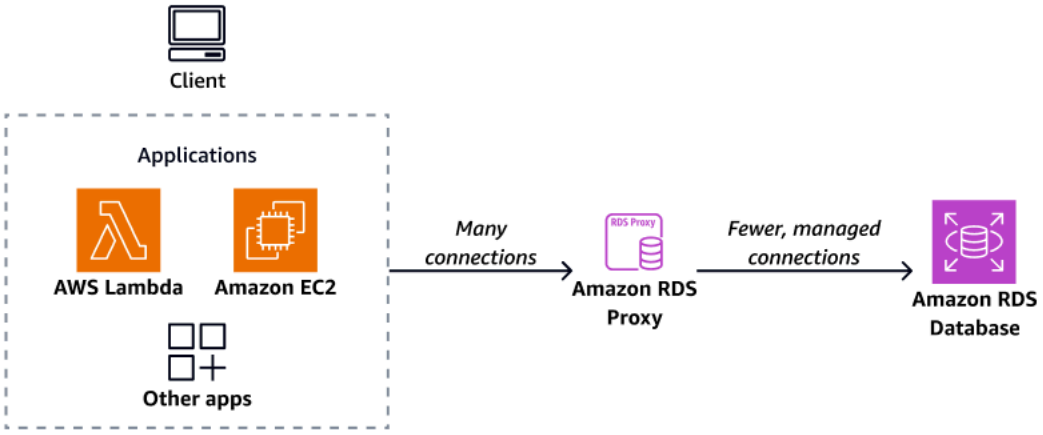
Datenbank-Engine	Engine-Nebenversionen	Hinweise
		der primären und sekundären DB-Cluster nicht übereinstimmen.

Datenbank-Engine	Engine-Nebenversionen	Hinweise
Aurora PostgreSQL	• Version 15 oder höhere Hauptversionen • Version 14.5 oder höhere Unterversionen • Version 13.8 oder höhere Unterversionen • Version 12.12 oder höhere Unterversionen • Version 11.17 oder höhere Unterversionen	Mit den in der vorherigen Spalte aufgeführten Engine-Versionen können Sie verwaltete regionsübergreifende Umstellungen oder Failover von einem primären DB-Cluster mit einem Patch-Level auf einen sekundären DB-Cluster mit einem anderen Patch-Level durchführen. Bei niedrigeren Nebenversionen können Sie verwaltete regionsübergreifende Umstellungen und Failover nur durchführen, wenn die Patch-Level der primären und sekundären DB-Cluster übereinstimmen. **Warning** Wenn Sie einen Cluster Ihrer globalen Datenbank auf eine der folgenden Patch-Versionen aktualisieren, können Sie regionsübergreifende Umstellungen oder Failover erst durchführen, wenn alle Cluster in Ihrer globalen Datenbank eine dieser Patch-Versionen oder eine neuere ausführen. • Patch-Versionen 16.1.6, 16.2.4, 16.3.2 und 16.4.2

Datenbank-Engine	Engine-Nebenversionen	Hinweise
		• Patch-Versionen 15.3.8, 15.4.9, 15.5.6, 15.6.4, 15.7.2 und 15.8.2 • Patch-Versionen 14.8.8, 14.9.9, 14.10.6, 14.11.4, 14.12.2 und 14.13.2

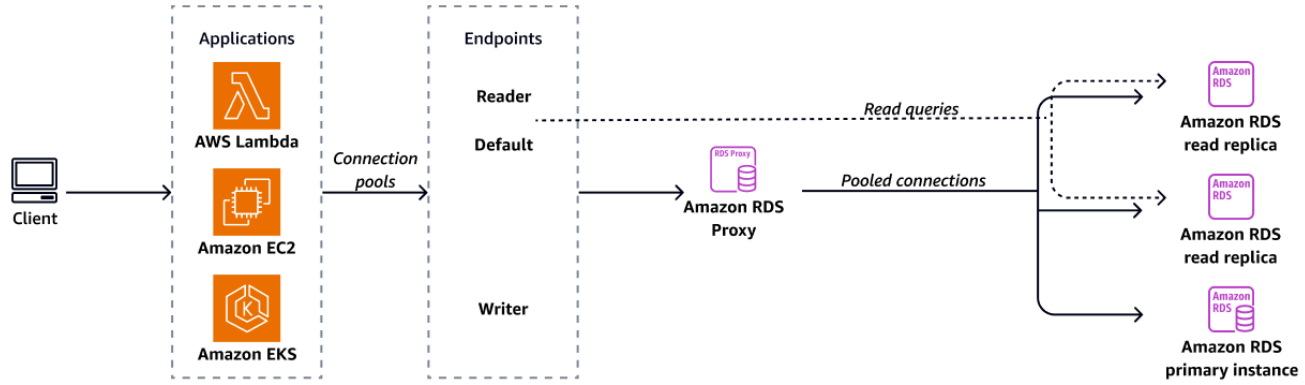
Amazon-RDS-Proxy für Aurora

Durch die Verwendung von Amazon-RDS-Proxy können Sie Ihren Anwendungen erlauben, Datenbankverbindungen zu bündeln und gemeinsam zu nutzen, um ihre Skalierbarkeit zu verbessern. RDS-Proxy macht Anwendungen widerstandsfähiger gegenüber Datenbankfehlern, indem er automatisch eine Verbindung zu einer Standby-DB-Instance herstellt, während Anwendungsverbindungen erhalten bleiben. Mithilfe von RDS Proxy können Sie die AWS Identity and Access Management (IAM-) Authentifizierung für Clients erzwingen, die eine Verbindung zum Proxy herstellen, und der Proxy kann mithilfe der IAM-Datenbankauthentifizierung oder der in gespeicherten Anmeldeinformationen eine Verbindung zu Datenbanken herstellen. AWS Secrets Manager



Mit RDS-Proxy können Sie unvorhersehbare Spitzen des Datenbankverkehrs bewältigen. Andernfalls könnten diese Spitzen Probleme verursachen, weil Verbindungen überbelegt sind oder schnell neue Verbindungen erstellt werden. RDS-Proxy richtet einen Datenbankverbindungspool ein und verwendet Verbindungen in diesem Pool wieder. Dieser Ansatz vermeidet den Speicher- und CPU-Overhead, der jedes Mal beim Öffnen einer neuen Datenbankverbindung erforderlich wäre. Um eine Datenbank vor Überbuchung zu schützen, können Sie die Anzahl der erstellten Datenbankverbindungen steuern.

RDS-Proxy stellt Anwendungsverbindungen in Warteschlangen oder drosselt sie, wenn sie nicht sofort vom Verbindungspool bedient werden können. Obwohl Latenzen zunehmen können, kann Ihre Anwendung weiter skalieren, ohne dass die Datenbank abrupt ausfällt oder überfordert wird. Wenn Verbindungsanforderungen die von Ihnen angegebenen Grenzwerte überschreiten, lehnt RDS-Proxy Anwendungsverbindungen ab (d. h. die Last wird abgeworfen). Gleichzeitig wird die vorhersehbare Leistung für die Last beibehalten, die mit verfügbarer Kapazität bedient werden kann.



Sie können den Overhead für die Verarbeitung von Anmeldeinformationen reduzieren und für jede neue Verbindung eine sichere Verbindung herstellen. Einige dieser Arbeiten kann RDS-Proxy im Auftrag der Datenbank verarbeiten.

RDS-Proxy ist mit den Engine-Versionen, die es unterstützt, vollständig kompatibel. Sie können RDS-Proxy für die meisten Anwendungen ohne Codeänderungen aktivieren. Eine Liste der unterstützten Engine-Versionen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Amazon-RDS-Proxy.

Themen

- Verfügbarkeit von Regionen und Versionen
- Kontingente und Einschränkungen für RDS-Proxy
- Planen des Verwendungsortes von RDS-Proxy
- Konzepte und Terminologie zu RDS-Proxy
- Erste Schritte mit RDS-Proxy
- Verwalten eines RDS-Proxy
- Arbeiten mit Amazon-RDS-Proxy-Endpunkten
- Überwachen von RDS-Proxy-Metriken mit Amazon CloudWatch
- Arbeiten mit RDS-Proxy-Ereignissen
- Befehlszeilenbeispiele für RDS Proxy
- Fehlersuche für RDS-Proxy
- Verwenden von RDS-Proxy mit AWS CloudFormation
- Verwenden von RDS Proxy mit globalen Aurora-Datenbanken

Verfügbarkeit von Regionen und Versionen

Hinweise zur Versionsunterstützung der Datenbank-Engine und zur Verfügbarkeit von RDS Proxy in einem bestimmten AWS-Region Fall finden Sie unterUnterstützte Regionen und Aurora-DB-Engines für Amazon-RDS-Proxy.

Kontingente und Einschränkungen für RDS-Proxy

Die folgenden Einschränkungen gelten für RDS-Proxy:

- Jede AWS-Konto ID ist auf 20 Proxys begrenzt. Wenn für Ihre Anwendung mehr Proxys erforderlich sind, fordern Sie eine Erhöhung über die Seite Service Quotas in der AWS-Managementkonsole an. Wählen Sie auf der Seite Service Quotas Amazon Relational Database Service (Amazon RDS) aus und suchen Sie nach Proxys, um eine Kontingenterhöhung anzufordern. AWS kann Ihr Kontingent oder die ausstehende Überprüfung Ihrer Anforderung automatisch um Support erhöhen.
- Jedem Proxy können bis zu 200 Secrets Manager Manager-Geheimnisse zugeordnet sein, wodurch Verbindungen bei der Verwendung von Geheimnissen auf bis zu 200 verschiedene Benutzerkonten beschränkt werden.
- Jeder Proxy hat einen Standardendpunkt. Sie können auch bis zu 20 Proxy-Endpunkte für jeden Proxy hinzufügen. Sie können diese Endpunkte erstellen, anzeigen, ändern und löschen.
- In einem Aurora-Cluster, werden alle Verbindungen, die den Standard-Proxy-Endpunkt verwenden, von der Aurora-Writer-Instance verarbeitet. Um einen Lastenausgleich für leseintensive Workloads durchzuführen, können Sie einen schreibgeschützten Endpunkt für einen Proxy erstellen. Dieser Endpunkt übergibt Verbindungen an den Reader-Endpunkt des Clusters. Auf diese Weise können Ihre Proxy-Verbindungen die Vorteile von Aurora-Leseskalierbarkeit nutzen. Weitere Informationen finden Sie unter Überblick über Proxy-Endpunkte.
- Sie können RDS-Proxy mit Clustern von Aurora Serverless v2 verwenden, jedoch nicht mit Clustern von Aurora Serverless v1.
- Ihr RDS-Proxy muss sich in derselben VPC wie die Datenbank befinden. Obwohl die Datenbank öffentlich zugänglich sein kann, kann der Proxy dies nicht sein. Wenn Sie beispielsweise Prototypen für Ihre Datenbank auf einem lokalen Host erstellen, können Sie keine Verbindung zu Ihrem Proxy herstellen, es sei denn, Sie richten die erforderlichen Netzwerkanforderungen ein, um eine Verbindung zum Proxy zuzulassen. Dies liegt daran, dass sich Ihr lokaler Host außerhalb der VPC des Proxys befindet.

> **Note**
>
> Für Aurora-DB-Cluster können Sie den VPC-übergreifenden Zugriff aktivieren. Erstellen Sie dazu einen zusätzlichen Endpunkt für einen Proxy und geben Sie eine andere VPC, Subnetze und Sicherheitsgruppen mit diesem Endpunkt an. Weitere Informationen finden Sie unter Zugriff auf Aurora über VPCs.

- Sie können RDS-Proxy nicht mit einer VPC verwenden, für deren Tenancy `dedicated` festgelegt wurde.
- Für IPv6 Endpunkt-Netzwerktypen konfigurieren Sie Ihre VPC und Subnetze so, dass sie nur unterstützt werden. IPv6 Konfigurieren Sie Ihre VPC IPv4 und Subnetze sowohl für Netzwerktypen als auch für IPv6 Zielverbindungen so, dass sie den Dual-Stack-Modus unterstützen.
- Wenn Sie den RDS-Proxy mit einem Aurora-DB-Cluster der verwenden, für den die IAM-Authentifizierung aktiviert ist, kann der Proxy entweder mithilfe der IAM-Authentifizierung oder der in Secrets Manager gespeicherten Anmeldeinformationen eine Verbindung zur Datenbank herstellen. Clients, die eine Verbindung zum Proxy herstellen, müssen sich mit IAM-Anmeldeinformationen authentifizieren. Detaillierte Konfigurationsanweisungen finden Sie unter und Datenbankanmeldedaten für RDS Proxy einrichten Konfigurieren der IAM-Authentifizierung für RDS-Proxy
- Sie können RDS-Proxy nicht mit benutzerdefiniertem DNS verwenden, wenn Sie die SSL-Hostnamenvalidierung nutzen.
- Jeder Proxy kann einem einzelnen Ziel-DB-Cluster zugeordnet werden. Sie können jedoch mehrere Proxys demselben DB-Cluster zuordnen.
- Jede Anweisung mit einer Textgröße über 16 KB bewirkt, dass der Proxy die Sitzung in der aktuellen Verbindung fixiert.
- In bestimmten Regionen gelten Einschränkungen für Availability-Zones (AZ), die Sie bei der Erstellung des Proxys berücksichtigen müssen. Die Region USA Ost (Nord-Virginia) unterstützt RDS-Proxy nicht in der Availability Zone `use1-az3`. Die Region USA West (Nordkalifornien) unterstützt RDS-Proxy nicht in der Availability Zone `usw1-az2`. Achten Sie beim Auswählen von Subnetzen während der Proxy-Erstellung darauf, dass Sie keine Subnetze in den oben genannten Availability Zones auswählen.
- Derzeit unterstützt RDS-Proxy keine globalen Bedingungskontextschlüssel.

  Weitere Informationen über globale Bedingungskontextschlüssel finden Sie unter Globale AWS - Bedingungskontextschlüssel im IAM-Benutzerhandbuch.

- Sie können RDS-Proxy nicht mit RDS Custom für SQL Server verwenden.
- Um Änderungen von Datenbankparametergruppen auf Ihrem Proxy widerzuspiegeln, ist ein Instance-Neustart erforderlich, auch wenn Sie sich dafür entschieden haben, Ihre Änderungen sofort zu übernehmen. Für Parameter auf Cluster-Ebene ist ein clusterweiter Neustart erforderlich.
- Ihr Proxy erstellt den DB-Benutzer `rdsproxyadmin` automatisch, wenn Sie ein Proxyziel registrieren. Dies ist ein geschützter Benutzer, der für die Proxyfunktionalität unerlässlich ist. Sie sollten jegliche Manipulation des Benutzers `rdsproxyadmin` vermeiden. Das Löschen oder Ändern des Benutzers `rdsproxyadmin` oder seiner Berechtigungen kann zu einer vollständigen Nichtverfügbarkeit des Proxys für Ihre Anwendung führen.

Weitere Informationen zu den Einschränkungen für jede DB-Engine finden Sie in den folgenden Abschnitten:

- Weitere Einschränkungen für Aurora MySQL
- Weitere Einschränkungen für Aurora PostgreSQL

Weitere Einschränkungen für Aurora MySQL

Die folgenden zusätzlichen Einschränkungen gelten für RDS-Proxy mit Datenbanken von RDS für MySQL:

- Die RDS-Proxyunterstützung für die Authentifizierung `caching_sha2_password` erfordert eine sichere Verbindung (TLS).
- Es ist bekannt, dass bei der RDS-Proxyunterstützung für `caching_sha2_password` Kompatibilitätsprobleme mit bestimmten Go-SQL-Treiberversionen vorhanden sind.
- Bei Verwendung des MySQL 8.4 C-Treibers kann die API `mysql_stmt_bind_named_param` falsch formatierte Pakete bilden, wenn die Anzahl der Parameter die Anzahl der Platzhalter in vorbereiteten Anweisungen übersteigt. Dies führt zu falschen Antworten. Weitere Informationen finden Sie unter MySQL bug report.
- Derzeit führen alle Proxys Listening auf Port 3306 für MySQL durch. Die Proxys stellen weiterhin eine Verbindung mit Ihrer Datenbank her, indem Sie den Port verwenden, den Sie in den Datenbankeinstellungen angegeben haben.
- Sie können RDS Proxy nicht mit selbstverwalteten MySQL-Datenbanken in EC2 Instanzen verwenden.

- Sie können RDS-Proxy nicht mit einer RDS-for-MySQL-DB-Instance verwenden, bei welcher der Parameter `read_only` in der DB-Parametergruppe auf 1 gesetzt wurde.
- RDS-Proxy unterstützt den komprimierten MySQL-Modus nicht. Es unterstützt z. B. nicht die Komprimierung, die von den Optionen `--compress` oder `-C` des `mysql`-Befehls verwendet wird.
- Datenbankverbindungen, die einen Befehl `GET DIAGNOSTIC` verarbeiten, geben möglicherweise ungenaue Informationen zurück, wenn der RDS-Proxy dieselbe Datenbankverbindung für eine weitere Abfrage wiederverwendet. Dies kann passieren, wenn der RDS-Proxy Datenbankverbindungen multiplexiert.
- Einige SQL-Anweisungen und -Funktionen wie `SET LOCAL` können den Verbindungsstatus ändern, ohne dass Pinning verursacht wird. Informationen zum aktuellen Fixierungsverhalten finden Sie unter Vermeiden von Pinning beim RDS-Proxy.
- Die Verwendung der Funktion `ROW_COUNT()` in einer Abfrage mit mehreren Anweisungen wird nicht unterstützt.
- RDS-Proxy unterstützt keine Client-Anwendungen, die nicht mehrere Antwortnachrichten in einem TLS-Eintrag verarbeiten können.
- RDS-Proxy unterstützt die doppelten MySQL-Passwörter nicht.
- RDS-Proxy funktioniert möglicherweise nicht wie erwartet, wenn Sie den Parameter `init_connect` in Ihrer RDS-DB-Parametergruppe so konfigurieren, dass Sitzungsstatusvariablen festgelegt werden. Legen Sie stattdessen die Initialisierungsabfrage für Ihren Proxy so fest, dass Sitzungsinitialisierungsanweisungen ausgeführt werden, wenn Sie den Proxy für die Verbindung mit Ihrer Datenbank verwenden.

> **Important**
>
> Setzen Sie in der Initialisierungsabfrage bei Proxys, die mit MySQL-Datenbanken verknüpft sind, den Konfigurationsparameter `sql_auto_is_null` nicht auf `true` oder einen Wert ungleich Null. Dies kann zu einem falschen Anwendungsverhalten führen.

Weitere Einschränkungen für Aurora PostgreSQL

Die folgenden zusätzlichen Einschränkungen gelten für RDS-Proxy mit Datenbanken von Aurora PostgreSQL:

- RDS-Proxy unterstützt keine Sitzungs-Pinning-Filter für PostgreSQL.

- Derzeit führen alle Proxys Listening auf Port 5432 für PostgreSQL durch.
- Für PostgreSQL unterstützt RDS-Proxy derzeit nicht das Abbrechen einer Abfrage von einem Client durch Ausgeben von `CancelRequest`. Dies ist beispielsweise der Fall, wenn Sie eine lange andauernde Abfrage in einer interaktiven psql-Sitzung mithilfe von Strg+C abbrechen.
- Die Ergebnisse der PostgreSQL-Funktion lastval sind nicht immer genau. Verwenden Sie zur Umgehung die INSERT-Anweisung mit der Klausel `RETURNING`.
- RDS-Proxy unterstützt derzeit den Streaming-Replikationsmodus nicht.
- Die `postgres` Standarddatenbank muss auf der RDS-Instanz für PostgreSQL vorhanden sein, damit der RDS-Proxy funktioniert. Löschen Sie diese Datenbank nicht, auch wenn Ihre Anwendung unterschiedliche Datenbanken verwendet.
- Wenn Sie `ALTER ROLE` verwenden, um die Benutzerrolle mit `SET ROLE` zu ändern, werden nachfolgende Verbindungen als dieser Benutzer zum Proxy diese Rolleneinstellung möglicherweise nicht, wenn bei diesen Verbindungen Pinning auftritt. Nutzen Sie bei der Verwendung eines Proxys `SET ROLE` in der Initialisierungsabfrage des Proxys, um dies zu vermeiden. Weitere Informationen finden Sie in der Initialisierungsabfrage unter Erstellen eines Proxys für Amazon Aurora.

> **Important**
>
> Wenn Sie bei vorhandenen Proxys mit PostgreSQL-Datenbanken die Datenbankauthentifizierung so ändern, dass nur `SCRAM` verwendet wird, ist der Proxy für bis zu 60 Sekunden nicht verfügbar. Um das Problem zu vermeiden, führen Sie einen der folgenden Schritte aus:
>
> - Stellen Sie sicher, dass die Datenbank sowohl die `SCRAM`- als auch die `MD5`-Authentifizierung zulässt.
> - Wenn Sie nur die `SCRAM`-Authentifizierung verwenden möchten, erstellen Sie einen neuen Proxy, migrieren Sie Ihren Anwendungsdatenverkehr auf den neuen Proxy und löschen Sie dann den zuvor mit der Datenbank verknüpften Proxy.

Planen des Verwendungsortes von RDS-Proxy

Sie können ermitteln, welche Ihrer DB-Instances, Cluster und Anwendungen möglicherweise am meisten von der Verwendung von RDS-Proxy profitieren. Berücksichtigen Sie dazu folgende Faktoren:

- Alle DB-Cluster, auf denen gelegentlich der Fehler „zu viele Verbindungen“ auftritt, sind gut geeignet für die Zuordnung zu einem Proxy. Dies ist häufig durch einen hohen Wert der `ConnectionAttempts` CloudWatch Metrik gekennzeichnet. Der Proxy ermöglicht es Anwendungen, viele Clientverbindungen zu öffnen, während der Proxy eine geringere Anzahl langlebiger Verbindungen mit dem DB-Cluster verwaltet.
- Bei , die kleinere AWS Instance-Klassen wie T2 oder T3 verwenden, kann die Verwendung eines Proxys helfen, Probleme zu vermeiden out-of-memory. Sie kann auch dazu beitragen, den CPU-Overhead für das Herstellen von Verbindungen zu reduzieren. Diese Bedingungen können auftreten, wenn es um eine große Anzahl von Verbindungen geht.
- Sie können bestimmte CloudWatch Amazon-Metriken überwachen, um festzustellen, ob sich ein bestimmten Grenzwerten nähert. Diese Grenzwerte gelten für die Anzahl der Verbindungen und den Arbeitsspeicher, der mit der Verbindungsverwaltung verbunden ist. Sie können auch bestimmte CloudWatch Metriken überwachen, um festzustellen, ob ein viele kurzlebige Verbindungen verarbeitet. Das Öffnen und Beenden solcher Verbindungen kann einen Leistungs-Overhead für Ihre Datenbank verursachen. Informationen zu den zu überwachenden Metriken finden Sie unter Überwachen von RDS-Proxy-Metriken mit Amazon CloudWatch.
- AWS Lambda Funktionen können auch gute Kandidaten für die Verwendung eines Proxys sein. Diese Funktionen stellen häufig kurze Datenbankverbindungen her, die von dem von RDS-Proxy angebotenen Verbindungspooling profitieren. Sie können alle IAM-Authentifizierungen nutzen, die Sie bereits für Lambda-Funktionen haben, anstatt Datenbankanmeldeinformationen im Lambda-Anwendungscode zu verwalten.
- Anwendungen, die in der Regel eine große Anzahl von Datenbankverbindungen öffnen und schließen und über keine integrierten Mechanismen für das Verbindungspooling verfügen, bieten sich für die Verwendung eines Proxys an.
- Anwendungen, die eine große Anzahl von Verbindungen über lange Zeiträume offen halten, sind in der Regel gute Kandidaten für die Verwendung eines Proxys. Anwendungen in Branchen wie Software as a Service (SaaS) oder E-Commerce minimieren häufig die Latenz für Datenbankanfragen, da sie Verbindungen offen lassen. Mit RDS-Proxy kann eine Anwendung mehr Verbindungen offen halten, als dies möglich ist, wenn eine direkte Verbindung mit dem DB--Cluster hergestellt wird.

- Möglicherweise haben Sie die IAM-Authentifizierung und den Secrets Manager nicht übernommen, da die Einrichtung einer solchen Authentifizierung für alle komplex ist. Der Proxy kann die Authentifizierungsrichtlinien für Clientverbindungen für bestimmte Anwendungen erzwingen. Sie können alle IAM-Authentifizierungen nutzen, die Sie bereits für Lambda-Funktionen haben, anstatt Datenbankanmeldeinformationen im Lambda-Anwendungscode zu verwalten.
- RDS-Proxy kann dazu beitragen, Anwendungen widerstandsfähiger und transparenter gegenüber Datenbankausfällen zu machen. RDS-Proxy umgeht Domain Name System (DNS)-Caches, um die Failover-Zeiten für Multi-AZ-Datenbanken von Aurora um bis zu 66 % zu reduzieren. RDS-Proxy leitet den Datenverkehr außerdem automatisch an eine neue Datenbank-Instance weiter, wobei Anwendungsverbindungen erhalten bleiben. Dadurch werden Failovers für Anwendungen transparenter.

Konzepte und Terminologie zu RDS-Proxy

Sie können die Verbindungsverwaltung für Ihre Amazon-Aurora-DB-Cluster vereinfachen, indem Sie RDS-Proxy verwenden.

RDS-Proxy verarbeitet den Netzwerkverkehr zwischen der Clientanwendung und der Datenbank. Es tut dies auf aktive Weise, indem es das Datenbankprotokoll erfasst Anschließend passt er sein Verhalten basierend auf den SQL-Operationen aus Ihrer Anwendung und den Ergebnismengen aus der Datenbank an.

RDS-Proxy reduziert den Arbeitsspeicher- und CPU-Overhead für die Verbindungsverwaltung in Ihrer Datenbank. Die Datenbank benötigt weniger Arbeitsspeicher und CPU-Ressourcen, wenn Anwendungen viele gleichzeitige Verbindungen öffnen. Es erfordert auch keine Logik in Ihren Anwendungen, um Verbindungen zu schließen und wieder zu öffnen, die für eine lange Zeit inaktiv bleiben. Ebenso erfordert es weniger Anwendungslogik, um Verbindungen im Falle eines Datenbankproblems wiederherzustellen.

Die Infrastruktur für RDS Proxy ist hochverfügbar und wird über mehrere Availability Zones (AZs) bereitgestellt. Berechnung, Arbeitsspeicher und Speicher für RDS-Proxy sind unabhängig von Ihrer Ihrem Aurora-DB-Cluster. Diese Trennung hilft, den Overhead auf Ihren Datenbankservern zu senken, sodass sie ihre Ressourcen für die Bereitstellung von Datenbank-Workloads einsetzen können. Die RDS-Proxy-Rechenressourcen sind serverless und werden automatisch basierend auf der Datenbank-Workload skaliert.

Themen

- Überblick über RDS-Proxy-Konzepte
- Verbindungspooling
- RDS-Proxy-Sicherheit
- Failover
- Transaktionen

Überblick über RDS-Proxy-Konzepte

RDS-Proxy verbreitet die Infrastruktur zum Ausführen des Verbindungspoolings und die anderen Funktionen, die in den folgenden Abschnitten beschrieben werden. Die in der RDS-Konsole dargestellten Proxys werden auf der Seite Proxys angezeigt.

Jeder Proxy verarbeitet Verbindungen zu einer einzelnen einem einzelnen Aurora-DB-Cluster. Bei von Aurora bereitgestellten Clustern bestimmt der Proxy die aktuelle Writer-Instance automatisch.

Die Verbindungen, die ein Proxy offen hält und für Ihre Datenbankanwendung verfügbar macht, bilden den Verbindungspool.

Standardmäßig kann RDS-Proxy eine Verbindung nach jeder Transaktion in Ihrer Sitzung wiederverwenden. Diese Wiederverwendung auf Transaktionsebene wird als Multiplexing bezeichnet. Wenn RDS-Proxy vorübergehend eine Verbindung aus dem Verbindungspool entfernt wird, um sie wiederzuverwenden, wird dieser Vorgang als Ausleihen der Verbindung bezeichnet. Wenn dies sicher ist, wird diese Verbindung an den Verbindungspool RDS-Proxy zurückgegeben.

In einigen Fällen kann RDS-Proxy nicht gewährleisten, dass eine Datenbankverbindung außerhalb der aktuellen Sitzung sicher wiederzuverwenden ist. In diesen Fällen bleibt die Sitzung auf derselben Verbindung erhalten, bis die Sitzung beendet ist. Dieses Fallback-Verhalten wird als Fixierung (Pinning) bezeichnet.

Ein Proxy hat einen Standard-Endpunkt. Sie stellen eine Verbindung mit diesem Endpunkt her, wenn Sie mit einem Amazon-Aurora-DB-Cluster arbeiten. Sie tun dies, anstatt eine Verbindung zu dem read/write Endpunkt herzustellen, der direkt mit dem verbunden ist. Die speziellen Endpunkte für einen Aurora-Cluster stehen Ihnen weiterhin zur Verfügung. Für Aurora-DB-Cluster können Sie auch zusätzliche read/write und schreibgeschützte Endpunkte erstellen. Weitere Informationen finden Sie unter Überblick über Proxy-Endpunkte.

Beispielsweise können Sie für read/write Verbindungen ohne Verbindungspooling weiterhin eine Verbindung zum Cluster-Endpunkt herstellen. Sie können weiterhin eine Verbindung zum Leser-

Endpunkt für schreibgeschützte Verbindungen mit Lastenausgleich herstellen. Sie können weiterhin eine Verbindung zu den Instance-Endpunkten für die Diagnose und Behebung von Fehlern in bestimmten DB-Instances innerhalb eines Clusters herstellen. Wenn Sie andere AWS Dienste verwenden, z. B. AWS Lambda um eine Verbindung zu RDS-Datenbanken herzustellen, ändern Sie deren Verbindungseinstellungen, sodass der Proxy-Endpunkt verwendet wird. Beispielsweise geben Sie den Proxy-Endpunkt an, damit Lambda-Funktionen auf Ihre Datenbank zugreifen und gleichzeitig die RDS-Proxy-Funktionalität nutzen können.

Jeder Proxy enthält eine Zielgruppe. Diese Zielgruppe verkörpert die den Aurora DB-Cluster, mit dem der Proxy eine Verbindung herstellen kann. Bei einem Aurora-Cluster ist die Zielgruppe standardmäßig allen DB-Instances in diesem Cluster zugeordnet. Auf diese Weise kann sich der Proxy mit jeder Aurora-DB-Instance verbinden, die zur Writer-Instance im Cluster hochgestuft wird. Die der Aurora-DB-Cluster, die/der mit einem Proxy verknüpft ist, wird als Ziel dieses Proxys bezeichnet. Wenn Sie einen Proxy über die Konsole erstellen, erstellt RDS-Proxy auch die entsprechende Zielgruppe und registriert die zugeordneten Ziele automatisch.

Eine Engine-Familie ist eine verwandte Gruppe von Datenbank-Engines, die dasselbe DB-Protokoll verwenden. Sie wählen die Engine-Familie für jeden Proxy, den Sie erstellen.

Verbindungspooling

Jeder Proxy führt ein Verbindungspooling für die Writer- und Reader-Instance der zugehörigen Aurora-DB aus. Das Verbindungspooling ist eine Optimierung, die den Overhead reduziert, der mit dem Öffnen und Beenden von Verbindungen und dem Vorhandensein vieler aktiver Verbindungen gleichzeitig verbunden ist. Dieser Overhead umfasst den Arbeitsspeicher, der für die Verarbeitung jeder neuen Verbindung erforderlich ist. Es ist außerdem CPU-Overhead erforderlich, um jede Verbindung zu schließen und eine neue zu öffnen. Beispiele hierfür sind Transport Layer Security/ Secure Sockets Layer (TLS/SSL), Handshaking, Authentifizierung, Verhandlungsmöglichkeiten usw. Das Verbindungspooling vereinfacht Ihre Anwendungslogik. Sie müssen keinen Anwendungscode schreiben, um die Anzahl gleichzeitig geöffneter Verbindungen zu minimieren.

Jeder Proxy führt darüber hinaus Multiplexing von Verbindungen aus, auch bekannt als Wiederverwendung von Verbindungen. Beim Multiplexing führt RDS-Proxy alle Operationen für eine Transaktion mit einer zugrunde liegenden Datenbankverbindung aus. RDS kann dann eine andere Verbindung für die nächste Transaktion verwenden. Sie können viele gleichzeitige Verbindungen zum Proxy öffnen und der Proxy hält eine geringere Anzahl von Verbindungen zur DB-Instance oder zum Cluster offen. Dadurch wird der Speicher-Overhead für Verbindungen auf dem

Datenbankserver weiter minimiert. Diese Technik reduziert auch die Wahrscheinlichkeit des Fehlers "zu viele Verbindungen".

RDS-Proxy-Sicherheit

RDS Proxy verwendet die vorhandenen RDS-Sicherheitsmechanismen wie TLS/SSL und AWS Identity and Access Management (IAM). Allgemeine Informationen zu diesen Sicherheitsfunktionen finden Sie unter Sicherheit in Amazon Aurora. Sie sollten sich außerdem unbedingt damit vertraut machen, wie Aurora mit Authentifizierung, Autorisierung und anderen Sicherheitsbereichen arbeitet.

RDS-Proxy kann als zusätzliche Sicherheitsebene zwischen Clientanwendungen und der zugrunde liegenden Datenbank fungieren. Sie können beispielsweise über TLS 1.3 eine Verbindung zum Proxy herstellen, selbst wenn die zugrunde liegende DB-Instance eine ältere Version von TLS unterstützt. Sie können mithilfe einer IAM-Rolle eine Verbindung zum Proxy herstellen, auch wenn der Proxy mithilfe der Authentifizierungsmethode Datenbankbenutzer und Kennwort eine Verbindung zur Datenbank herstellt. Mit dieser Technik können Sie hohe Authentifizierungsanforderungen für Datenbankanwendungen ohne einen kostspieligen Migrationsaufwand für die DB-Instances selbst erzwingen.

Mit RDS Proxy können Sie die folgenden Authentifizierungsmethoden verwenden:

- Anmeldeinformationen für die Datenbank
- Standard-IAM-Authentifizierung
- End-to-end IAM-Authentifizierung

Verwenden von IAM mit RDS-Proxy

RDS Proxy bietet zwei Methoden der IAM-Authentifizierung:

- Standard-IAM-Authentifizierung: Erzwingen Sie die IAM-Authentifizierung für Verbindungen zu Ihrem Proxy, während der Proxy mithilfe der in Secrets Manager gespeicherten Anmeldeinformationen eine Verbindung zur Datenbank herstellt. Dadurch wird die IAM-Authentifizierung für den Datenbankzugriff erzwungen, auch wenn die Datenbanken die native Passwortauthentifizierung verwenden. Der Proxy ruft die Datenbankanmeldedaten von Secrets Manager ab und wickelt die Authentifizierung gegenüber der Datenbank im Namen Ihrer Anwendung ab.
- End-to-end IAM-Authentifizierung: Erzwingt die IAM-Authentifizierung für Verbindungen direkt von Ihren Anwendungen zu Ihrer Datenbank über den Proxy. End-to-end Die IAM-

Authentifizierung vereinfacht Ihre Sicherheitskonfiguration und vermeidet die Verwaltung von Datenbankanmeldedaten in Secrets Manager. Diese zusätzliche Sicherheitsebene erzwingt eine IAM-basierte Zugriffskontrolle von der Client-Anwendung zur Datenbank.

Um die Standard-IAM-Authentifizierung zu verwenden, konfigurieren Sie Ihren Proxy so, dass Secrets Manager Manager-Geheimnisse für die Authentifizierung verwendet werden, und aktivieren Sie die IAM-Authentifizierung für Client-Verbindungen. Ihre Anwendungen authentifizieren sich beim Proxy mithilfe von IAM, während sich der Proxy mit den von Secrets Manager abgerufenen Anmeldeinformationen gegenüber der Datenbank authentifiziert.

Um die end-to-end IAM-Authentifizierung zu verwenden, konfigurieren Sie Ihren Proxy so, dass er die IAM-Authentifizierung verwendet, wenn Sie das Standardauthentifizierungsschema bei der Erstellung oder Änderung Ihres Proxys festlegen.

Für die end-to-end IAM-Authentifizierung müssen Sie die dem Proxy zugeordnete IAM-Rolle aktualisieren, um die Berechtigung zu erteilen. `rds-db:connect` Mit der end-to-end IAM-Authentifizierung entfällt die Notwendigkeit, einzelne Datenbankbenutzer über Secrets Manager Secrets beim Proxy zu registrieren.

Verwendung TLS/SSL mit RDS Proxy

Sie können mithilfe des TLS/SSL Protokolls eine Verbindung zum RDS-Proxy herstellen.

> **Note**
>
> RDS Proxy verwendet Zertifikate von AWS Certificate Manager (ACM). Wenn Sie RDS-Proxy verwenden, müssen Sie keine Amazon-RDS-Zertifikate herunterladen oder Anwendungen aktualisieren, die RDS-Proxy-Verbindungen verwenden.

Um TLS für alle Verbindungen zwischen dem Proxy und der Datenbank zu erzwingen, können Sie beim Erstellen eines Proxys die Einstellung Transport Layer Security erforderlich angeben, wenn Sie einen Proxy in der AWS-Managementkonsole erstellen oder ändern.

RDS Proxy kann auch sicherstellen, dass Ihre Sitzung TLS/SSL zwischen Ihrem Client und dem RDS-Proxy-Endpunkt verwendet wird. Damit RDS-Proxy so verfährt, müssen Sie die clientseitige Anforderung festlegen. Für SSL-Verbindungen mit einer Datenbank unter Verwendung von RDS-Proxy werden keine SSL-Sitzungsvariablen festgelegt.

- Geben Sie für Aurora MySQL die clientseitige Anforderung mit dem Parameter `--ssl-mode` an, wenn Sie den Befehl `mysql` ausführen.
- Geben Sie für Aurora PostgreSQL `sslmode=require` als Teil der `conninfo`-Zeichenfolge an, wenn Sie den Befehl `psql` ausführen.

RDS-Proxy unterstützt die TLS-Protokollversionen 1.0, 1.1 und 1.2. Sie können eine Verbindung mit dem Proxy herstellen, indem Sie eine höhere TLS-Version verwenden, als Sie in der zugrunde liegenden Datenbank verwenden.

Standardmäßig richten Client-Programme eine verschlüsselte Verbindung mit RDS-Proxy ein, wobei weitere Kontrolle über die Option `--ssl-mode` verfügbar ist. Clientseitig unterstützt RDS-Proxy alle SSL-Modi.

Für den Client sind die SSL-Modi die folgenden:

PREFERRED

SSL ist die erste Wahl, aber nicht erforderlich.

DISABLED

SSL ist nicht zulässig.

REQUIRED

SSL erzwingen.

VERIFY_CA

SSL erzwingen und die Zertifizierungsstelle (CA) überprüfen.

VERIFY_IDENTITY

SSL erzwingen und die CA sowie den CA-Hostname überprüfen.

Bei Verwendung eines Clients mit `--ssl-mode VERIFY_CA` oder `VERIFY_IDENTITY`, geben Sie die Option `--ssl-ca` an, die auf eine CA im `.pem`-Format verweist. Laden Sie für die zu verwendende `.pem` Datei alle Root-CA PEMs von Amazon Trust Services herunter und platzieren Sie sie in einer einzigen `.pem` Datei.

RDS-Proxy verwendet Platzhalterzertifikate, die sowohl für eine Domain als auch für deren Unterdomains gelten. Wenn Sie den `mysql`-Client für die Verbindung mit dem SSL-Modus

`VERIFY_IDENTITY` verwenden, müssen Sie derzeit den MySQL 8.0-kompatiblen Befehl `mysql` verwenden.

Failover

Failover ist eine Funktion mit hoher Verfügbarkeit, die eine Datenbank-Instance durch eine andere ersetzt, wenn die ursprüngliche Instance nicht verfügbar ist. Ein Failover kann aufgrund eines Problems mit einer Datenbank-Instance auftreten. Er kann aber auch Teil normaler Wartungsverfahren sein, z. B. während eines Datenbank-Upgrades. Failover gilt für Aurora-DB-Cluster mit einer oder mehreren Reader-Instances zusätzlich zur Writer-Instance.

Die Verbindung über einen Proxy macht Ihre Anwendungen widerstandsfähiger gegenüber einem Datenbank-Failover. Wenn die ursprüngliche DB-Instance nicht verfügbar ist, stellt RDS-Proxy eine Verbindung mit der Standby-Datenbank her, ohne dass die inaktiven Anwendungsverbindungen verworfen werden. Dadurch wird der Failover-Prozess beschleunigt und vereinfacht. Dies ist mit weniger Störungen für Ihre Anwendung verbunden als ein typisches Neustart- oder Datenbankproblem.

Ohne RDS-Proxy ist ein Failover mit einem kurzen Ausfall verbunden. Während des Ausfalls können Sie keine Schreibvorgänge für diese Datenbank ausführen. Alle vorhandenen Datenbankverbindungen werden unterbrochen und Ihre Anwendung muss sie erneut öffnen. Die Datenbank wird für neue Verbindungen und Schreiboperationen verfügbar, wenn eine schreibgeschützte DB-Instance anstelle einer nicht verfügbaren Instance hochgestuft wird.

Während DB-Failovers akzeptiert RDS-Proxy weiterhin Verbindungen mit derselben IP-Adresse und leitet Verbindungen automatisch an die neue primäre DB-Instance weiter. Clients, die eine Verbindung über RDS-Proxy herstellen, sind nicht anfällig für Folgendes:

- DNS (Domain Name System)-Ausbreitungsverzögerungen beim Failover.
- Lokale DNS-Zwischenspeicherung.
- Verbindungszeitüberschreitungen.
- Unsicherheit darüber, welche DB-Instance der aktuelle Writer ist.
- Warten auf eine Abfrageantwort eines früheren Writers, die nicht verfügbar wurde, ohne Verbindungen zu schließen.

Bei Anwendungen, die über eigene Verbindungspools verfügen, sorgt RDS-Proxy dafür, dass die meisten Verbindungen bei Failovers oder anderen Unterbrechungen aktiv bleiben. Nur Verbindungen, die sich mitten in einer Transaktion oder SQL-Anweisung befinden, werden unterbrochen. RDS-Proxy

akzeptiert sofort neue Verbindungen. Wenn der Datenbank-Schreiber nicht verfügbar ist, werden eingehende Anfragen in RDS-Proxy in die Warteschlange gesetzt.

Für Anwendungen, die nicht über eigene Verbindungspools verfügen, bietet RDS-Proxy schnellere Verbindungsraten und mehr offene Verbindungen. Es reduziert den teuren Overhead, der mit dem häufigen Herstellen neuer Verbindungen über die Datenbank verbunden ist. Hierfür werden Datenbankverbindungen wiederverwendet, die im RDS-Proxy-Verbindungspool verwaltet werden. Dieser Ansatz ist besonders wichtig für TLS-Verbindungen, die mit erheblichen Setupkosten verbunden sind.

Transaktionen

Alle Anweisungen innerhalb einer einzelnen Transaktion verwenden immer dieselbe zugrunde liegende Datenbankverbindung. Die Verbindung wird für eine andere Sitzung verfügbar, wenn die Transaktion beendet wird. Die Verwendung der Transaktion als Granularitätseinheit hat folgende Auswirkungen:

- Die Wiederverwendung von Verbindungen kann nach jeder einzelnen Anweisung erfolgen, wenn die Einstellung `autocommit` von Aurora MySQL aktiviert ist.
- Ist die Einstellung `autocommit` deaktiviert, beginnt im Gegensatz dazu die erste Anweisung, die Sie in einer Sitzung ausgeben, eine neue Transaktion. Angenommen, Sie geben die Sequenz `SELECT`, `INSERT`, `UPDATE` und andere Data Manipulation Language (DML)-Anweisungen ein. In diesem Fall wird die Wiederverwendung von Verbindungen erst vorgenommen, wenn Sie einen `COMMIT`, `ROLLBACK` ausgeben oder die Transaktion anderweitig beenden.
- Die Eingabe einer DDL-Anweisung (Data Definition Language) bewirkt, dass die Transaktion beendet wird, nachdem diese Anweisung abgeschlossen wurde.

RDS-Proxy erkennt über das Netzwerkprotokoll, das von der Datenbank-Clientanwendung verwendet wird, wenn eine Transaktion beendet wird. Die Transaktionserkennung beruht nicht auf Schlüsselwörtern wie beispielsweise `COMMIT` oder `ROLLBACK`, die im Text der SQL-Anweisung angezeigt werden.

In einigen Fällen kann RDS-Proxy eine Datenbankanforderung erkennen, die es unpraktisch macht, Ihre Sitzung auf eine andere Verbindung zu verschieben. In diesen Fällen wird das Multiplexing für diese Verbindung während der verbleibenden Sitzung deaktiviert. Die gleiche Regel gilt, wenn RDS-Proxy nicht sicher sein kann, dass das Multiplexing für die Sitzung praktikabel ist. Diese Operation wird als Fixierung (Pinning) bezeichnet. Hinweise zum Erkennen und Minimieren von Fixierungen finden Sie unter Vermeiden von Pinning beim RDS-Proxy.

Erste Schritte mit RDS-Proxy

Verwenden Sie die Informationen auf den folgenden Seiten, um zugehörige Sicherheitsoptionen einzurichten, zu verwalten Amazon-RDS-Proxy für Aurora und festzulegen. Diese Sicherheitsoptionen steuern, wer auf jeden Proxy zugreifen kann und wie jeder Proxy eine Verbindung mit DB-Instances herstellt.

Wenn Sie mit RDS-Proxy noch nicht vertraut sind, empfehlen wir, den Seiten in der Reihenfolge zu folgen, in der sie angezeigt werden.

Themen

- Einrichten der Netzwerkvoraussetzungen für RDS-Proxy
- Datenbankanmeldedaten für RDS Proxy einrichten
- Konfigurieren der IAM-Authentifizierung für RDS-Proxy
- Erstellen eines Proxys für Amazon Aurora
- Anzeigen eines Proxys
- Verbinden mit einer Datenbank über RDS-Proxy

Einrichten der Netzwerkvoraussetzungen für RDS-Proxy

Die Verwendung von RDS-Proxy erfordert eine gemeinsame Virtual Private Cloud (VPC) zwischen Ihrem Aurora-DB-Cluster und RDS-Proxy. Diese VPC sollte über mindestens zwei Subnetze verfügen, die sich in verschiedenen Availability Zones befinden. Ihr Konto kann entweder der Eigentümer dieser Subnetze sein oder sie mit anderen Konten teilen. Informationen zur VPC-Sharing finden Sie unter Arbeiten mit Shared VPCs.

Für den IPv6 Support ist eine zusätzliche Netzwerkkonfiguration erforderlich:

- IPv6 Endpunkt-Netzwerktypen — Ihre VPC und Subnetze müssen für die Unterstützung konfiguriert sein. IPv6 Dazu gehört die Zuweisung von IPv6 CIDR-Blöcken zu Ihrer VPC und Ihren Subnetzen.
- Dual-Stack-Endpunkt-Netzwerktypen — Ihre VPC und Subnetze müssen beide IPv4 und die Adressierung unterstützen. IPv6
- IPv6 Netzwerktypen für Zielverbindungen — Ihre Datenbank muss für den Dual-Stack-Modus konfiguriert sein, um IPv6 Verbindungen vom Proxy aus zu unterstützen.

Ihre Client-Anwendungsressourcen wie Amazon EC2, Lambda oder Amazon ECS können sich in derselben VPC wie der Proxy befinden. Sie können sich auch in einer vom Proxy getrennten VPC befinden. Wenn Sie eine Verbindung mit Aurora-DB-Clustern hergestellt haben, verfügen Sie bereits über die erforderlichen Netzwerkressourcen.

Themen

- Abrufen von Informationen zu Ihren Subnetzen
- Planen der Kapazität von IP-Adressen

Abrufen von Informationen zu Ihren Subnetzen

Wenn Sie Aurora zum ersten Mal verwenden, können Sie sich mit den Grundlagen der Herstellung einer Verbindung mit einer Datenbank vertraut machen, indem Sie die in Einrichten Ihrer Umgebung für Amazon Aurora beschriebenen Verfahren befolgen. Sie können auch dem Tutorial in Erste Schritte mit Amazon Aurora folgen.

Um einen Proxy zu erstellen, müssen Sie die Subnetze und die VPC angeben, in denen der Proxy arbeitet. Das folgende Linux-Beispiel zeigt AWS CLI Befehle, die die Subnetze VPCs und Subnetze untersuchen, die Ihnen gehören. AWS-Konto Insbesondere übergeben Sie das Subnetz IDs als Parameter, wenn Sie einen Proxy mit der CLI erstellen.

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

Das folgende Linux-Beispiel zeigt AWS CLI Befehle zur Bestimmung des Subnetzes, das einem bestimmten Aurora DB-Cluster einer bestimmten entspricht.

Bei einem Aurora-Cluster finden Sie zuerst die ID für eine der zugehörigen DB-Instances. Sie können das von dieser DB-Instance IDs verwendete Subnetz extrahieren. Untersuchen Sie dazu die verschachtelten Felder innerhalb der Attribute `DBSubnetGroup` und `Subnets` in der Describe-Ausgabe für die DB-Instance. Sie geben einige oder alle dieser Subnetze an, IDs wenn Sie einen Proxy für diesen Datenbankserver einrichten.

```
$ # Find the ID of any DB instance in the cluster.
$  aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].
[DBClusterMembers]|[0]|[0][*].DBInstanceIdentifier' --output text
```

```
my_instance_id
instance_id_2
instance_id_3
```

Nachdem Sie die ID der DB-Instance gefunden haben, untersuchen Sie die zugehörige VPC, um ihre Subnetze zu finden. Das folgende Linux-Beispiel zeigt wie es geht.

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0]|[Subnets]|[0]|[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

```
$ #From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

Planen der Kapazität von IP-Adressen

RDS Proxy passt seine Kapazität automatisch an die Konfiguration der bei ihm registrierten DB-Instances an. Bei bereitgestellten Instances wird dies durch die Instance-Größe und bei Aurora Serverless v2 Instances durch die maximale ACU-Kapazität bestimmt. Bestimmte Operationen

erfordern möglicherweise auch zusätzliche Kapazität, z. B. die Vergrößerung einer registrierten Datenbank oder interne Wartungsvorgänge für den RDS-Proxy. Bei diesen Vorgängen benötigt Ihr Proxy möglicherweise mehr IP-Adressen, um die zusätzliche Kapazität bereitzustellen. Mit diesen zusätzlichen Adressen kann Ihr Proxy skaliert werden, ohne Ihre Workload zu beeinträchtigen. Ein Mangel an freien IP-Adressen in Ihren Subnetzen verhindert, dass ein Proxy hochskaliert wird. Dies kann zu höheren Abfragelatenzen oder Verbindungsfehlern bei Clients führen. RDS benachrichtigt Sie durch das Ereignis `RDS-EVENT-0243`, wenn in Ihren Subnetzen nicht genügend freie IP-Adressen vorhanden sind. Weitere Informationen zu diesem Ereignis finden Sie unter Arbeiten mit RDS-Proxy-Ereignissen.

Reservieren Sie die empfohlene Mindestanzahl von IP-Adressen in Ihren Subnetzen für Ihren Proxy basierend auf den Klassengrößen der DB-Instance.

DB-Instance-Klasse	Mindestanzahl freier IP-Adressen
db.*.xlarge oder kleiner	10
db.*.2xlarge	15
db.*.4xlarge	25
db.*.8xlarge	45
qdb.*.12xlarge	60
db.*.16xlarge	75
db.*.24xlarge	110

Reservieren Sie Aurora Serverless v2 nämlich die folgende Mindestanzahl an freien IP-Adressen in Ihren Subnetzen für Ihren Proxy, basierend auf der maximalen ACU-Kapazität.

`Maximum ACU Capacity`	Mindestanzahl freier IP-Adressen
16 oder kleiner	10
32	15
64	25

Maximum ACU Capacity	Mindestanzahl freier IP-Adressen
96	30
128	40
160	50
192	55
224	65
256	75

> **Note**
>
> RDS Proxy verbraucht nicht mehr als 215 IP-Adressen für jeden Proxy in einer VPC.

RDS Proxy benötigt mindestens 10 IP-Adressen für Ihre Aurora-Datenbank. Bei der empfohlenen Anzahl von IP-Adressen handelt es sich um eine Schätzung für einen Proxy, der nur über einen Standardendpunkt verfügt. Für jeden weiteren benutzerdefinierten Endpunkt empfehlen wir, drei weitere IP-Adressen zu reservieren. Wir empfehlen, dass Sie für jede Aurora-Reader-Instance zusätzliche IP-Adressen reservieren, wie in der Tabelle angegeben, basierend auf der maximalen Größe dieses Readers ACUs für das Aurora Serverless v2 Ziel oder der DB-Instance-Größe für ein bereitgestelltes Ziel.

Um die erforderlichen IP-Adressen für einen Proxy zu schätzen, der mit einem Aurora-DB-Cluster verknüpft ist, mit:

- 1 bereitgestellte Writer-Instance in der Größe `db.r5.8xlarge` und 1 bereitgestellte Reader-Instance in der Größe. `db.r5.2xlarge`
- Der an diesen Cluster angehängte Proxy hat einen Standardendpunkt und einen benutzerdefinierten Endpunkt mit der Rolle „Nur Lesen“.

In diesem Fall benötigt der Proxy ungefähr 63 freie IP-Adressen (45 für die Writer-Instance, 15 für die Reader-Instance und 3 für den zusätzlichen benutzerdefinierten Endpunkt).

Um die erforderlichen IP-Adressen für einen Proxy zu schätzen, der einem Aurora-DB-Cluster zugeordnet ist, der über Folgendes verfügt:

- 1 Aurora Serverless v2 Writer-Instance mit einer maximalen Kapazität von 256 ACUs und 1 Serverless v2-Reader-Instance mit einer maximalen Kapazität von 192 ACUs.
- Der Proxy, der an diesen Cluster angehängt ist, hat den Standardendpunkt und einen benutzerdefinierten Endpunkt mit der Rolle „Nur Lesen“.

In diesem Fall benötigt der Proxy ungefähr 133 freie IP-Adressen (75 für die Writer-Instanz, 55 für die Reader-Instanz und 3 für den zusätzlichen benutzerdefinierten Endpunkt).

Um die erforderlichen IP-Adressen für einen Proxy zu schätzen, der einem Aurora-Cluster zugeordnet ist, der über Folgendes verfügt:

- 1 bereitgestellte Writer-Instance mit der DB-Instance-Größe db.r5.4xlarge und 1 Serverless v2-Reader-Instance mit einer maximalen Kapazität von 64. ACUs
- Der Proxy, der an diesen Cluster angehängt ist, hat den Standardendpunkt und einen benutzerdefinierten Endpunkt mit der Rolle „Nur Lesen“.

In diesem Fall benötigt der Proxy ungefähr 53 freie IP-Adressen (25 für die Writer-Instanz, 25 für die Reader-Instanz und 3 für den zusätzlichen benutzerdefinierten Endpunkt).

Um die erforderlichen IP-Adressen für einen Proxy zu schätzen, der einem Aurora-DB-Cluster zugeordnet ist, der über Folgendes verfügt:

- 1 bereitgestellte Writer-Instance der Größe db.r5.24xlarge und 3 bereitgestellte Reader-Instance der Größe db.r5.8xlarge.
- Der Proxy, der mit diesem DB-Cluster verbunden ist, verfügt über den Standardendpunkt und 1 benutzerdefinierten Endpunkt mit der schreibgeschützten Rolle.

In diesem Fall benötigt der Proxy 215 freie IP-Adressen. Berechnungen deuten zwar auf 248 IPs (110 + (3*45) + 3) hin, aber der RDS-Proxy verbraucht nicht mehr als 215 IP-Adressen für jeden Proxy in einer VPC.

Datenbankanmeldedaten für RDS Proxy einrichten

RDS Proxy in Amazon RDS verwendet AWS Secrets Manager , um Datenbankanmeldedaten sicher zu speichern und zu verwalten. Anstatt Anmeldeinformationen in Ihre Anwendung einzubetten, verknüpfen Sie einen Proxy mit einem Secrets-Manager-Secret, das die erforderlichen Authentifizierungsdetails enthält. Sie erstellen ein separates Secrets Manager-Secret für jedes Datenbankbenutzerkonto, mit dem der Proxy auf der dem Aurora-DB-Cluster eine Verbindung herstellt.

Alternativ können Sie den RDS-Proxy so konfigurieren, dass er die end-to-end IAM-Authentifizierung verwendet, sodass keine Datenbankanmeldedaten in Secrets Manager gespeichert werden müssen. RDS Proxy verwendet die IAM-Authentifizierung für beide Verbindungen client-to-proxy. proxy-to-database Dies bietet eine vollständig integrierte IAM-basierte Authentifizierungslösung, für die keine Verwaltung von Geheimnissen oder Passwörtern erforderlich ist. Informationen zum Hinzufügen eines neuen IAM-DB-Benutzers finden Sie unter. Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Themen

- Erstellen von Secrets zur Verwendung mit RDS-Proxy

Erstellen von Secrets zur Verwendung mit RDS-Proxy

Bevor Sie einen Proxy erstellen, müssen Sie zunächst mindestens ein Secret erstellen, in dem Ihre Datenbankanmeldeinformationen gespeichert werden.

Konsole

So erstellen Sie ein Secret

1. Öffnen Sie die Secrets Manager Manager-Konsole unter https://console.aws.amazon.com/secretsmanager/.
2. Wählen Sie Store a new secret (Ein neues Secret speichern).
3. Wählen Sie Anmeldeinformationen für Amazon-RDS-Datenbank aus.
4. Geben Sie einen Benutzernamen und ein Passwort ein. Die von Ihnen eingegebenen Anmeldeinformationen müssen mit den Anmeldeinformationen eines Datenbankbenutzers übereinstimmen, der in der zugehörigen RDS-Datenbank vorhanden ist. RDS-Proxy verwendet diese Anmeldeinformationen, um im Namen von Anwendungen Verbindungen zur Datenbank zu authentifizieren und herzustellen.

Wenn eine Unstimmigkeit vorliegt, können Sie das Secret so aktualisieren, dass es mit dem Datenbankkennwort übereinstimmt. Solange Sie das Secret nicht aktualisieren, schlagen Verbindungsversuche über den Proxy fehl, Verbindungen, die andere gültige geheime Schlüssel verwenden, funktionieren aber dennoch.

> **Note**
>
> Für RDS für SQL Server erfordert RDS-Proxy unabhängig von den Sortierungseinstellungen der DB-Instance ein Secret in Secrets Manager, bei dem Groß- und Kleinschreibung beachtet wird. Wenn Ihre Anwendung Benutzernamen mit unterschiedlicher Groß- und Kleinschreibung wie „Admin“ und „admin“ zulässt, müssen Sie für jeden separate Secrets erstellen. RDS-Proxy unterstützt keine Benutzernamenauthentifizierung ohne Berücksichtigung der Groß-/Kleinschreibung zwischen Client und Proxy.
> Weitere Informationen über die Sortierung in SQL Server finden Sie in der Dokumentation für Microsoft SQL Server.

5. Wählen Sie für Datenbank die Amazon-RDS-Datenbank aus, auf die das Secret zugreift.
6. Geben Sie weitere Einstellungen für das Secret ein und wählen Sie dann Speichern aus. Umfassende Anleitungen finden Sie unter Erstellen eines AWS Secrets Manager -Secrets im Benutzerhandbuch für AWS Secrets Manager .

AWS CLI

Wenn Sie über den einen Proxy erstellen AWS CLI, geben Sie die Amazon-Ressourcennamen (ARNs) der entsprechenden Geheimnisse an. Diesen Vorgang führen Sie für alle DB-Benutzerkonten aus, auf die der Proxy zugreifen kann. In der AWS-Managementkonsole wählen Sie die Geheimnisse anhand ihrer aussagekräftigen Namen aus.

- Verwenden Sie den Befehl create-secret, um ein Secrets Manager-Secret für die Verwendung mit RDS-Proxy zu erstellen:

```
aws secretsmanager create-secret \
  --name "secret_name" \
  --description "secret_description" \
  --region region_name \
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

- Sie können auch einen benutzerdefinierten Schlüssel erstellen, um Ihr Secrets Manager-Secret zu verschlüsseln. Im folgenden Beispiel wird ein Beispielschlüssel erstellt.

```
aws kms create-key --description "test-key" --policy '{
  "Id":"kms-policy",
  "Version": "2012-10-17",
  "Statement":
    [
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws:iam::account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
        "Principal":
          {
            "AWS":
              ["$USER_ARN","arn:aws:iam:account_id::role/Admin"]
          },
        "Action":
          [
            "kms:Create*",
            "kms:Describe*",
            "kms:Enable*",
            "kms:List*",
            "kms:Put*",
            "kms:Update*",
            "kms:Revoke*",
            "kms:Disable*",
            "kms:Get*",
            "kms:Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
          ],
        "Resource":"*"
      },
      {
        "Sid":"Allow use of the key",
```

```
        "Effect":"Allow",
        "Principal":{"AWS":"$ROLE_ARN"},
        "Action":["kms:Decrypt","kms:DescribeKey"],
        "Resource":"*"
      }
    ]
}'
```

Mit den folgenden Befehlen werden beispielsweise Secrets Manager-Secrets für zwei Datenbankbenutzer erstellt:

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

Verwenden Sie die folgenden Befehle, um diese mit Ihrem benutzerdefinierten AWS KMS Schlüssel verschlüsselten Geheimnisse zu erstellen:

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id
```

Verwenden Sie den Befehl list-secrets, um die Geheimnisse zu sehen, die Ihrem AWS Konto gehören:

```
aws secretsmanager list-secrets
```

Wenn Sie mit der CLI einen Proxy erstellen, übergeben Sie die Amazon-Ressourcennamen (ARNs) eines oder mehrerer Secrets an den `--auth` Parameter. Das folgende Beispiel zeigt, wie Sie einen Bericht nur mit dem Namen und dem ARN jedes Secrets Ihres AWS -Kontos erstellen. In diesem

Beispiel wird der `--output table` Parameter verwendet, der in AWS CLI Version 2 verfügbar ist. Wenn Sie AWS CLI Version 1 verwenden, verwenden Sie `--output text` stattdessen.

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

Verwenden Sie den get-secret-valueBefehl, um zu überprüfen, ob der geheime Schlüssel die richtigen Anmeldeinformationen im richtigen Format enthält. Ersetzen Sie *your_secret_name* durch den Kurznamen oder den ARN des Secrets.

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

Die Ausgabe enthält eine Zeile mit einem JSON-codierten Wert, der dem folgenden ähnelt:

```
...
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
...
```

Konfigurieren der IAM-Authentifizierung für RDS-Proxy

Um die AWS Identity and Access Management (IAM-) Authentifizierung für RDS Proxy in Amazon RDS einzurichten, erstellen und konfigurieren Sie eine IAM-Richtlinie, die die erforderlichen Berechtigungen gewährt.

In diesem Thema werden die Schritte zur Konfiguration der IAM-Authentifizierung für den RDS-Proxy beschrieben, einschließlich der Erstellung der erforderlichen IAM-Richtlinie und deren Anfügung an eine IAM-Rolle.

> **Tip**
>
> Dieses Verfahren ist nur erforderlich, wenn Sie Ihre eigene IAM-Rolle erstellen möchten. Andernfalls kann RDS die erforderliche Rolle automatisch erstellen, wenn Sie den Proxy einrichten, sodass Sie diese Schritte überspringen können.

Voraussetzungen

Stellen Sie vor der Einrichtung der IAM-Authentifizierung für RDS-Proxy sicher, dass Sie über Folgendes verfügen:

- AWS Secrets Manager – Mindestens ein gespeichertes Secret, das Datenbankanmeldedaten enthält. Anleitungen zum Erstellen von Secrets finden Sie unter the section called “Einrichten von Datenbank-Anmeldeinformationen”.

  Dies ist nicht erforderlich, wenn Sie die end-to-end IAM-Authentifizierung verwenden.
- IAM-Berechtigungen – Eine IAM-Rolle oder ein IAM-Benutzer mit Berechtigungen zum Erstellen und Verwalten von IAM-Richtlinien, -Rollen und -Secrets in AWS Secrets Manager.

Eine IAM-Richtlinie für end-to-end die IAM-Authentifizierung erstellen

Wenn Sie die end-to-end IAM-Authentifizierung verwenden, stellt RDS Proxy mithilfe der IAM-Authentifizierung eine Verbindung zu Ihrer Datenbank her, anstatt Anmeldeinformationen von Secrets Manager abzurufen. Dazu müssen Sie Ihre IAM-Rolle mit `rds-db:connect` Berechtigungen für die Datenbankkonten konfigurieren, die Sie mit dem Proxy verwenden möchten.

Um Ihren RDS-Proxy mithilfe von IAM gegenüber der Datenbank zu authentifizieren, erstellen Sie eine IAM-Rolle mit einer Richtlinie, die die erforderlichen Datenbankverbindungsberechtigungen gewährt.

Konsole

Um eine Rolle für die end-to-end IAM-Authentifizierung mit Ihrem Proxy zu erstellen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Erstellen Sie eine Berechtigungsrichtlinie für die Rolle. Allgemeine Schritte finden Sie unter IAM-Richtlinien erstellen (Konsole).

   Fügen Sie die folgende Richtlinie in den JSON-Editor ein und nehmen Sie die folgenden Änderungen vor:

   - Ersetzen Sie die Vorgabe durch Ihre eigene Konto-ID.
   - Ersetzen Sie es `us-east-2` durch den Ort, an dem sich der Proxy befinden muss.
   - Ersetzen Sie die Datenbankressource IDs und die Benutzernamen durch die Namen, die Sie verwenden möchten. Das Format der Ressourcen-ID unterscheidet sich zwischen RDS-Instances undAurora clusters.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds-db:connect",
            "Resource": [
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_1",
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_2"
            ]
        }
    ]
}
```

3. Erstellen Sie die Rolle fügen Sie ihr die Berechtigungsrichtlinie an. Allgemeine Schritte finden Sie unter Eine Rolle erstellen, um Berechtigungen an einen AWS Dienst zu delegieren.

   Wählen Sie für Vertrauenswürdiger Entitätstyp die Option AWS -Service aus. Wählen Sie unter Anwendungsfall die Option RDS und für den Anwendungsfall RDS — Rolle zur Datenbank hinzufügen aus.

4. Wählen Sie für Berechtigungsrichtlinien die Richtlinie aus, die Sie erstellt haben.

5. Geben Sie für Vertrauenswürdige Entitäten auswählen die folgende Vertrauensrichtlinie für die Rolle ein:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS CLI

Um die Rolle mithilfe von zu erstellen AWS CLI, senden Sie die folgende Anfrage:

```
aws iam create-role \
  --role-name my_e2e_iam_role_name \

  --assume-role-policy-document '{"Version":"2012-10-17",
     "Statement":[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'
```

Fügen Sie anschließend die Richtlinie an die Rolle an.

```
aws iam put-role-policy \
  --role-name my_e2e_iam_role_name \
  --policy-name e2e_iam_db_connect_policy \
  --policy-document '{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds-db:connect",
            "Resource": [
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_1",
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_2"
            ]
        }
    ]
}'
```

Wenn die IAM-Rolle und die Berechtigungen für die end-to-end IAM-Authentifizierung konfiguriert sind, können Sie jetzt einen Proxy mit der `DefaultAuthScheme` Einstellung auf erstellen. `IAM_AUTH` Dieser Proxy authentifiziert sich mithilfe von IAM direkt bei der Datenbank, ohne dass Secrets Manager Manager-Geheimnisse erforderlich sind. Detaillierte Anweisungen finden Sie unter the section called “Erstellen eines Proxys”.

Wenn Sie die end-to-end IAM-Authentifizierung verwenden, stellen Sie sicher, dass Ihre Datenbankbenutzer für die IAM-Authentifizierung konfiguriert sind, wie unter beschrieben. Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Erstellen einer IAM-Richtlinie für den Zugriff auf Secrets Manager

Damit RDS-Proxy Datenbankanmeldedaten von Secrets Manager abrufen kann, erstellen Sie eine IAM-Rolle mit einer Richtlinie, die die notwendigen Berechtigungen gewährt.

Konsole

So erstellen Sie eine Rolle für den Zugriff auf Ihre Secrets für die Verwendung mit Ihrem Proxy

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Erstellen Sie eine Berechtigungsrichtlinie für die Rolle. Allgemeine Schritte finden Sie unter IAM-Richtlinien erstellen (Konsole).

   Fügen Sie die folgende Richtlinie in den JSON-Editor ein und nehmen Sie die folgenden Änderungen vor:

   - Ersetzen Sie die Vorgabe durch Ihre eigene Konto-ID.
   - Ersetzen Sie `us-east-2` durch die Region, in der sich der Proxy befinden wird.
   - Ersetzen Sie die geheimen Namen durch die Namen, die Sie erstellt haben. Weitere Informationen finden Sie unter Angeben von KMS-Schlüsseln in IAM-Richtlinienanweisungen.
   - Ersetzen Sie die KMS-Schlüssel-ID durch die, die Sie zur Verschlüsselung der Secrets von Secrets Manager verwendet haben, entweder den Standardschlüssel oder Ihren eigenen Schlüssel.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-
east-2:111122223333:secret:secret_name_1",
                "arn:aws:secretsmanager:us-
east-2:111122223333:secret:secret_name_2"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:111122223333:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-
east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. Erstellen Sie die Rolle fügen Sie ihr die Berechtigungsrichtlinie an. Allgemeine Schritte finden Sie unter Eine Rolle erstellen, um Berechtigungen für einen AWS Dienst zu delegieren.

   Wählen Sie für Vertrauenswürdiger Entitätstyp die Option AWS -Service aus. Wählen Sie unter Anwendungsfall die Option RDS und für den Anwendungsfall RDS — Rolle zur Datenbank hinzufügen aus.

4. Wählen Sie für Berechtigungsrichtlinien die Richtlinie aus, die Sie erstellt haben.

5. Geben Sie für Vertrauenswürdige Entitäten auswählen die folgende Vertrauensrichtlinie für die Rolle ein:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS CLI

Um die Rolle mithilfe von zu erstellen AWS CLI, senden Sie die folgende Anfrage:

```
aws iam create-role \
  --role-name my_role_name \
  --assume-role-policy-document '{"Version": "2012-10-17",
      "Statement":[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'
```

Fügen Sie anschließend die Richtlinie an die Rolle an.

```
aws iam put-role-policy \
  --role-name my_role_name \
  --policy-name secret_reader_policy \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}'
```

Nachdem Sie die IAM-Rolle und die Berechtigungen konfiguriert haben, können Sie nun einen Proxy erstellen und ihn dieser Rolle zuordnen. Auf diese Weise kann der Proxy Datenbankanmeldedaten

sicher aus Ihren Anwendungen abrufen AWS Secrets Manager und die IAM-Authentifizierung für Ihre Anwendungen aktivieren. Detaillierte Anweisungen finden Sie unter the section called “Erstellen eines Proxys”.

Erstellen eines Proxys für Amazon Aurora

Sie können Amazon-RDS-Proxy verwenden, um die Skalierbarkeit, Verfügbarkeit und Sicherheit Ihrer Datenbankanwendungen zu verbessern, indem Sie Verbindungen bündeln und Datenbank-Failover effizienter verwalten. In diesem Thema wird das Erstellen eines Proxys erläutert. Stellen Sie vor Beginn sicher, dass Ihre Datenbank die erforderlichen Voraussetzungen erfüllt, einschließlich IAM-Berechtigungen und VPC-Konfiguration.

Sie können einen Proxy einem DB-Cluster von Aurora MySQL oder einem Aurora PostgreSQL zuordnen.

Konsole

So erstellen Sie einen Proxy

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Proxies (Proxys).
3. Wählen Sie Create proxy (Proxy erstellen).
4. Konfigurieren Sie die folgenden Einstellungen für Ihren Proxy.

Einstellung	Description
Engine-Familie	Das Datenbanknetzwerkprotokoll, das der Proxy erkennt, wenn er den Netzwerkverkehr zu und von der Datenbank interpretiert. **Note** Um zu verwenden, stellen Sie sicher, dass die `postgres`-Datenbank in Ihrer Instance beibehalten wird. Siehe Problembehandlung bei gelöschter `postgres`-Datenbank.

Einstellung	Description
Proxy-Kennung	Ein Name, der innerhalb Ihrer AWS Konto-ID und Ihrer aktuellen AWS Region eindeutig ist.
Timeout der inaktiven Clientverbindung	Der Proxy schließt eine Clientverbindung, wenn er für einen bestimmten Zeitraum inaktiv bleibt. Standardmäßig sind dies 1.800 Sekunden (30 Minuten). Eine Verbindung ist inaktiv, wenn die Anwendung innerhalb der angegebenen Zeit nach Abschluss der vorherigen Anforderung keine neue Anforderung sendet. Der Proxy hält die zugrunde liegende Datenbankverbindung geöffnet und gibt sie an den Verbindungspool zurück, sodass sie für neue Clientverbindungen verfügbar ist. Reduzieren Sie die Zeitüberschreitung für die inaktive Clientver bindung, um veraltete Verbindungen proaktiv zu entfernen. Um die Verbindungskosten bei Workload-Spitzen zu minimieren, erhöhen Sie den Timeout.
Datenbank	der Aurora-DB-Cluster zum Aufrufen über diesen Proxy. Die Liste enthält nur DB-Instances und Cluster mit kompatiblen Datenbank-Engines, Engine-Versionen und anderen Einstellu ngen. Wenn die Liste leer ist, erstellen Sie eine neue DB-Instan ce oder einen neuen Cluster, mit der/dem RDS-Proxy kompatibe l ist. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Erstellen eines Amazon Aurora-DB Clusters. Versuchen Sie dann erneut, den Proxy zu erstellen.
Maximale Verbindungen für Verbindungspool	Ein Wert zwischen 1 und 100, um den Prozentsatz des Limits `max_connections` zu definieren, den RDS-Proxy verwenden kann. Wenn Sie nur einen Proxy mit dieser DB-Instance oder diesem DB-Cluster verwenden möchten, legen Sie diesen Wert auf 100 fest. Weitere Informationen dazu, wie RDS-Proxy diese Einstellung verwendet, finden Sie unter the section called “MaxConnectionsPercent”.

Einstellung	Description
Sitzungs-Pinning-Filter	Hindert den RDS-Proxy daran, bestimmte erkannte Sitzungsstatus zu pinnen, wodurch die Standardsicherheitsmaßnahmen für Multiplexverbindungen umgangen werden. Derzeit unterstützt PostgreSQL diese Einstellung nicht, und die einzig verfügbare Option ist `EXCLUDE_VARIABLE_SETS`. Die Aktivierung dieser Option kann dazu führen, dass Sitzungsvariablen aus einer Verbindung andere Verbindungen beeinflussen, was zu Fehlern oder Korrektheitsproblemen führen kann, wenn Abfragen auf Sitzungsvariablen basieren, die außerhalb der aktuellen Transaktion festgelegt wurden. Verwenden Sie diese Option erst, nachdem Sie sich vergewissert haben, dass Ihre Anwendungen Datenbankverbindungen sicher gemeinsam nutzen können. Die folgenden Muster gelten als sicher: • `SET`-Anweisungen, bei denen der effektive Wert der Sitzungsvariablen nicht geändert wird. Mit anderen Worten, es gibt keine Änderung an der Sitzungsvariablen. • Sie ändern den Wert der Sitzungsvariablen und führen eine Anweisung in derselben Transaktion aus. Weitere Informationen finden Sie unter Vermeiden von Pinning beim RDS-Proxy.
Zeitüberschreitung für die Verbindung	Wenn Sie erwarten, dass der Proxy alle verfügbaren Datenbankverbindungen verwendet, legen Sie die Wartezeit fest, bis ein Timeout-Fehler zurückgegeben wird. Sie können bis zu fünf Minuten angeben. Diese Einstellung gilt nur, wenn der Proxy die maximale Anzahl von Verbindungen geöffnet hat und alle bereits verwendet werden.

Einstellung	Description
Initialisierungsabfrage	(Optional) Fügen Sie eine Initialisierungsabfrage hinzu oder ändern Sie die aktuelle. Sie können eine oder mehrere SQL-Anweisungen für die Ausführung durch den Proxy beim Öffnen jeder neuen Datenbankverbindung festlegen. Diese Einstellung wird normalerweise mit SET-Anweisungen verwendet, um sicherzustellen, dass jede Verbindung identische Einstellungen aufweist. Stellen Sie sicher, dass die von Ihnen hinzugefügte Abfrage gültig ist. Verwenden Sie Komma als Trennzeichen, um mehrere Variablen in eine einzelne SET-Anweisung einzuschließen. Zum Beispiel: `SET variable1 =value1, variable2 =value2` Verwenden Sie für mehrere Anweisungen Semikola als Trennzeichen. **Important** Da Sie im Rahmen der Zielgruppenkonfiguration auf die Initialisierungsabfrage zugreifen können, ist diese nicht durch Authentifizierungs- oder Verschlüsselungsmethoden geschützt. Jeder, der Zugriff auf Ihre Proxy-Zielgruppenkonfiguration hat, kann die Initialisierungsabfrage anzeigen. Sie sollten dieser Option keine sensiblen Daten wie Passwörter oder langlebige Verschlüsselungscodes hinzufügen.
AWS Identity and Access Management (IAM) -Rolle	Eine IAM-Rolle mit der Berechtigung zum Zugriff auf die Secrets von Secrets Manager, die die Anmeldeinformationen für Datenbankbenutzerkonten darstellen, die der Proxy verwenden kann. Alternativ können Sie eine neue IAM-Rolle aus der AWS-Managementkonsole verwenden.

Einstellung	Description
Secrets-Manager-Secrets	Erstellen oder wählen Sie Secrets Manager Manager-Geheimnisse, die die Anmeldeinformationen für Datenbankbenutzerkonten darstellen, die den Proxy verwenden können. Wenn das Standardauthentifizierungsschema auf „Kein“ gesetzt ist, ist dieses Feld erforderlich. Wenn das Standardauthentifizierungsschema auf IAM-Authentifizierung festgelegt ist, wird dieses Feld optional und in der Konsole entsprechend gekennzeichnet. Sie können ein oder mehrere Geheimnisse aus der Dropdownliste auswählen oder über den Link Neues Geheimnis erstellen ein neues Geheimnis erstellen.
Client-Authentifizierungstyp	Der Typ von Authentifizierung, den der Proxy für Verbindungen von Clients verwendet. Die ausgewählte Option gilt für alle Secrets-Manager-Secrets, die Sie diesem Proxy zuordnen. Wenn Sie für jedes Geheimnis einen anderen Client-Authentifizierungstyp angeben müssen, erstellen Sie Ihren Proxy stattdessen mithilfe der AWS CLI oder der API. Geben Sie diese Option nur an, wenn Ihre Client-Verbindung Datenbankanmeldeinformationen zur Authentifizierung verwendet.
IAM-Authentifizierung	Geben Sie für die IAM-Authentifizierung für Verbindungen zu Ihrem Proxy „Erforderlich“ „Zulässig“ oder „Nicht zulässig“ an. Die ausgewählte Option gilt für alle Secrets-Manager-Secrets, die Sie diesem Proxy zuordnen. Wenn Sie für jedes Geheimnis eine andere IAM-Authentifizierung angeben müssen, erstellen Sie Ihren Proxy stattdessen mithilfe der AWS CLI oder der API.

Einstellung	Description
Standardauthentifizierungsschema	Wählen Sie den Standardauthentifizierungstyp, den der Proxy für Client-Verbindungen zum Proxy und die Verbindungen vom Proxy zur zugrunde liegenden Datenbank verwendet. Ihnen stehen folgende Optionen zur Verfügung: • Keine (Standard) — Der Proxy ruft Datenbankanmeldeinformationen aus Secrets Manager Manager-Secrets ab. • IAM-Authentifizierung — Der Proxy verwendet die IAM-Authentifizierung, um eine Verbindung zur Datenbank herzustellen, wodurch die IAM-Authentifizierung aktiviert end-to-end wird. Wenn Sie die IAM-Authentifizierung auswählen, wird eine Informationswarnung angezeigt, die Sie daran erinnert, die IAM-Datenbankauthentifizierung für die Datenbanken in der Zielgruppenkonfiguration zu aktivieren. **Note** Diese Option wird nur für MySQL-, PostgreSQL- und MariaDB-Engine-Familien unterstützt.

Einstellung	Description
Datenbankkonten für die IAM-Authentifizierung	Dieses Feld wird nur angezeigt, wenn das Standardauthentifizierungsschema auf IAM-Authentifizierung und die Rolle Identity and Access Management (IAM) auf IAM-Rolle erstellen gesetzt ist. Benennen Sie die Datenbankbenutzerkonten für den Proxy, der mit der IAM-Authentifizierung verwendet werden soll. Dies ist ein Pflichtfeld. Geben Sie mehrere Konten an, indem Sie: • Geben Sie einen Datenbankbenutzernamen ein, um ihn als Tag hinzuzufügen • Verwenden bestimmter Datenbankbenutzernamen (z. B.`db_user`,`jane_doe`) • Verwendung von Platzhaltermustern für mehrere Benutzer (z. B.`db_test_*`) Jedes Konto wird als entfernbares Tag angezeigt, das Sie löschen können, indem Sie auf das X-Symbol klicken. Die Konsole verwendet diese Werte, um die entsprechenden `rds-db:connect` Berechtigungen in der IAM-Rollenrichtlinie zu erstellen.
Beantragen der Sicherheit der Transportebene	Wird TLS/SSL für alle Client-Verbindungen durchgesetzt. Der Proxy verwendet dieselben Verschlüsselungseinstellungen für seine Verbindung zur zugrunde liegenden Datenbank, unabhängig davon, ob die Clientverbindung verschlüsselt oder unverschlüsselt ist.

Einstellung	Description
Netzwerktyp der Zielverbindung	Die IP-Version, die der Proxy verwendet, um eine Verbindung zur Zieldatenbank herzustellen. Wählen Sie aus den folgenden Optionen aus: • IPv4— Der Proxy stellt über IPv4 Adressen eine Verbindung zur Datenbank her. • IPv6— Der Proxy stellt über IPv6 Adressen eine Verbindung zur Datenbank her. Der Standardwert ist IPv4. Zur Verwendung IPv6 muss Ihre Datenbank den Dual-Stack-Modus unterstützen. Der Dual-Stack-Modus ist für Zielverbindungen nicht verfügbar.
Netzwerktyp des Endpunkts	Die IP-Version für den Proxy-Endpunkt, über den Clients eine Verbindung zum Proxy herstellen. Wählen Sie aus den folgenden Optionen aus: • IPv4— Der Proxy-Endpunkt verwendet nur IPv4 Adressen. • IPv6— Der Proxy-Endpunkt verwendet nur IPv6 Adressen. • Dual-Stack — Der Proxy-Endpunkt unterstützt IPv4 sowohl IPv6 Adressen als auch. Der Standardwert ist IPv4. Um Dual-Stack verwenden zu können IPv6 , müssen Ihre VPC und Subnetze so konfiguriert sein, dass sie den ausgewählten Netzwerktyp unterstützen.
Subnets	Dieses Feld ist mit allen Subnetzen gefüllt, die mit Ihrer VPC verknüpft sind. Sie können alle Subnetze entfernen, die für den Proxy nicht benötigt werden, müssen jedoch mindestens zwei Subnetze übrig lassen. Stellen Sie bei Netzwerktypen IPv6 oder Dual-Stack-Endpunkten sicher, dass die ausgewählten Subnetze den ausgewählten Netzwerktyp unterstützen.

Einstellung	Description
VPC Security Group (VPC-Sicherheitsgruppe)	Wählen Sie eine vorhandene VPC-Sicherheitsgruppe aus oder erstellen Sie eine neue über die AWS-Managementkonsole. Konfigurieren Sie die Eingangsregeln so, dass Ihre Anwendungen auf den Proxy zugreifen können, und die Ausgangsregeln so, dass der Datenverkehr von Ihren Datenbankzielen zugelassen wird. **Note** Die Sicherheitsgruppe muss Verbindungen vom Proxy mit der Datenbank zulassen. Sie wird für eingehenden Datenverkehr von Ihren Anwendungen zum Proxy und für ausgehenden Datenverkehr vom Proxy zur Datenbank verwendet. Stellen Sie sicher, dass Ressourcen innerhalb dieser Sicherheitsgruppe miteinander kommunizieren können, wenn Sie für die Datenbank und den Proxy dieselbe Sicherheitsgruppe verwenden. Vermeiden Sie die Verwendung der Standardsicherheitsgruppe für die VPC oder einer mit einem anderen Konto verknüpften Sicherheitsgruppe, wenn Sie eine freigegebene VPC nutzen. Wählen Sie stattdessen eine Sicherheitsgruppe aus, die zu Ihrem Konto gehört. Wenn keine vorhanden ist, erstellen Sie eine. Weitere Informationen finden Sie unter Arbeiten mit geteilten VPCs. RDS stellt einen Proxy über mehrere Availability Zones hinweg bereit, um eine hohe Verfügbarkeit zu gewährleisten. Um die AZ-übergreifende Kommunikation zu ermöglichen, muss die Netzwerk-Zugriffssteuerungsliste (ACL) für Ihr Proxy-Subnetz ausgehenden Datenverkehr auf dem Engine-Port und eingehenden Datenverkehr auf allen Ports zulassen. Weitere Informationen zum Netzwerk ACLs finden Sie unter Steuern des

Einstellung	Description
	Datenverkehrs zu Subnetzen über das Netzwerk ACLs. Wenn die Netzwerk-ACL für Ihren Proxy und Ihr Ziel identisch sind, müssen Sie eine TCP-Protokoll-Ingress-Regel hinzufügen, bei der die Quelle auf die VPC-CIDR festgelegt ist. Sie müssen auch eine Engine-Port-spezifische TCP-Protokollregel für ausgehenden Datenverkehr hinzufügen, bei der die Quelle auf die VPC-CIDR festgelegt ist.
Erweiterte Protokollierung aktivieren	Aktivieren Sie diese Einstellung, um Proxy-Kompatibilitäts- oder Leistungsprobleme zu beheben. Wenn diese Option aktiviert ist, protokolliert RDS-Proxy detaillierte Leistungsinformationen, um Sie beim Debuggen des SQL-Verhaltens oder der Leistung und Skalierbarkeit von Proxyverbindungen zu unterstützen. Aktivieren Sie diese Einstellung nur zum Debuggen und stellen Sie sicher, dass angemessene Sicherheitsmaßnahmen zum Schutz vertraulicher Informationen in den Protokollen vorhanden sind. Um den Overhead zu minimieren, deaktiviert RDS-Proxy diese Einstellung 24 Stunden nach der Aktivierung automatisch. Verwenden Sie sie vorübergehend, um bestimmte Probleme zu beheben.

5. Wählen Sie Create proxy (Proxy erstellen).

AWS CLI

Um einen Proxy mit dem zu erstellen AWS CLI, rufen Sie den create-db-proxyBefehl mit den folgenden erforderlichen Parametern auf:

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--vpc-subnet-ids`

Bei `--engine-family`-Wert ist die Groß- und Kleinschreibung zu beachten.

Example

Für Linux, macOS oder Unix:

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
    [--default-auth-scheme { NONE | IAM_AUTH }] \
    [--auth ProxyAuthenticationConfig_JSON_string] \
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--endpoint-network-type { IPV4 | IPV6 | DUAL }] \
    [--target-connection-network-type { IPV4 | IPV6 }] \
    [--tags comma_separated_list]
```

Für Windows:

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--default-auth-scheme { NONE | IAM_AUTH }] ^
    [--auth ProxyAuthenticationConfig_JSON_string] ^
    [--vpc-security-group-ids space_separated_list] ^
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--endpoint-network-type { IPV4 | IPV6 | DUAL }] ^
    [--target-connection-network-type { IPV4 | IPV6 }] ^
    [--tags comma_separated_list]
```

Nachstehend finden Sie ein Beispiel für den JSON-Wert der `--auth`-Option. Dieses Beispiel wendet auf jedes Secret einen anderen Client-Authentifizierungstyp an.

```
[
  {
    "Description": "proxy description 1",
```

```
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret/1234abcd-12ab-34cd-56ef-1234567890cd",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"

  },

  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }

]
```

Der `--endpoint-network-type`-Parameter gibt die IP-Version für den Proxy-Endpunkt an, über den Clients eine Verbindung zum Proxy herstellen. Folgende sind gültige Werte:

- `IPV4`— Der Proxy-Endpunkt verwendet nur IPv4 Adressen (Standard).
- `IPV6`— Der Proxy-Endpunkt verwendet nur IPv6 Adressen.
- `DUAL`— Der Proxy-Endpunkt unterstützt IPv4 sowohl IPv6 Adressen als auch.

Der Parameter `--target-connection-network-type` gibt die IP-Version an, die der Proxy verwendet, um eine Verbindung zur Zieldatenbank herzustellen. Folgende sind gültige Werte:

- `IPV4`— Der Proxy stellt über IPv4 Adressen eine Verbindung zur Datenbank her (Standard).
- `IPV6`— Der Proxy stellt über IPv6 Adressen eine Verbindung zur Datenbank her.

Um Dual-Stack-Endpunkt-Netzwerktypen zu verwenden IPv6 , müssen Ihre VPC und Subnetze so konfiguriert sein, dass sie den ausgewählten Netzwerktyp unterstützen. Um den Netzwerktyp IPv6 Zielverbindung zu verwenden, muss Ihre Datenbank den Dual-Stack-Modus unterstützen.

> **Tip**
>
> Wenn Sie noch nicht wissen, welches Subnetz für den `--vpc-subnet-ids` Parameter verwendet werden IDs soll, finden Sie unter Einrichten der Netzwerkvoraussetzungen für RDS-Proxy Beispiele, wie Sie sie finden können.

> **Note**
>
> Diese Sicherheitsgruppe muss den Zugriff auf die Datenbank zulassen, mit welcher der Proxy eine Verbindung herstellt. Dieselbe Sicherheitsgruppe wird für eingehenden Datenverkehr von Ihren Anwendungen zum Proxy und für ausgehenden Datenverkehr vom Proxy zur Datenbank verwendet. Angenommen, Sie verwenden dieselbe Sicherheitsgruppe für Ihre Datenbank und Ihren Proxy. Stellen Sie in diesem Fall sicher, dass Sie angeben, dass Ressourcen in dieser Sicherheitsgruppe mit anderen Ressourcen in derselben Sicherheitsgruppe kommunizieren können.
> Wenn Sie eine freigegebene VPC verwenden, können Sie die Standardsicherheitsgruppe für die VPC oder eine zu einem anderen Konto gehörende Gruppe nicht verwenden. Wählen Sie eine Sicherheitsgruppe aus, die zu Ihrem Konto gehört. Wenn keine vorhanden ist, erstellen Sie eine. Weitere Informationen zu dieser Einschränkung finden Sie unter Arbeiten mit Shared VPCs.

Um die richtigen Verknüpfungen für den Proxy zu erstellen, verwenden Sie auch den register-db-proxy-targetsBefehl. Angabe des `default`-Zielgruppennamens. RDS-Proxy erstellt automatisch eine Zielgruppe mit diesem Namen, wenn Sie jeden Proxy erstellen.

```
aws rds register-db-proxy-targets
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

RDS-API

Um einen RDS-Proxy zu erstellen, rufen Sie den Amazon RDS-API-Vorgang Create aufDBProxy. Sie übergeben einen Parameter mit der AuthConfigDatenstruktur.

RDS-Proxy erstellt automatisch eine Zielgruppe mit dem Namen `default`, wenn Sie die einzelnen Proxys erstellen. Sie ordnen der Zielgruppe einen zu, indem Sie die Funktion DBProxyZiele registrieren aufrufen.

> **Important**
>
> Wenn Sie die IAM-Authentifizierung für das Standardauthentifizierungsschema auswählen:
>
> - Sie müssen die IAM-Datenbankauthentifizierung auf Ihren Zieldatenbank-Instances oder Clustern aktivieren, bevor der Proxy erfolgreich eine Verbindung herstellen kann.
> - Wenn Sie IAM-Rolle erstellen wählen, ist das Feld Datenbankkonten für die IAM-Authentifizierung erforderlich.
> - Wenn Sie eine vorhandene IAM-Rolle auswählen, aktualisiert die Konsole die Rolle nicht automatisch mit Datenbankverbindungsberechtigungen. Überprüfen Sie, ob die Rolle über die erforderlichen `rds-db:connect` Berechtigungen verfügt.

Anzeigen eines Proxys

Nachdem Sie einen oder mehrere RDS-Proxys erstellt haben, können Sie ihn bzw. sie in der AWS-Managementkonsole, der AWS CLI oder der RDS-API anzeigen und verwalten. Sie können ihre Konfigurationsdetails überprüfen, die Leistung überwachen und festlegen, welche Proxys nach Bedarf geändert oder gelöscht werden sollen.

Damit Datenbankanwendungen den Datenverkehr über einen Proxy weiterleiten können, müssen Sie den Proxy-Endpunkt in der Verbindungszeichenfolge angeben.

Konsole

So zeigen Sie einen Proxy in der Konsole an

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Proxies (Proxys).

3. Wählen Sie den Proxy-Namen aus, um seine Details anzuzeigen.
4. Auf der Detailseite zeigt der Abschnitt Zielgruppen an, wie der Proxy einer spezifischen einem spezifischen Aurora-DB-Cluster zugeordnet ist. Sie können zur Standardzielgruppenseite navigieren, um eine detailliertere Ansicht dieser Zuordnung zu erhalten, einschließlich der Konfigurationseinstellungen, die bei der Proxy-Erstellung definiert wurden. Zu diesen Einstellungen gehören maximaler Verbindungsprozentsatz, Zeitüberschreitung für die Verbindung, Engine-Familie und Sitzungs-Pinning-Filter.

CLI

Um Ihren Proxy mit der CLI anzuzeigen, verwenden Sie den Befehl describe-db-proxies. Standardmäßig werden alle Proxys im Besitz Ihres AWS-Kontos zurückgegeben. Um Details für einen einzelnen Proxy anzuzeigen, geben Sie seinen Namen mit dem Parameter `--db-proxy-name` an.

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

Verwenden Sie die folgenden Befehle, um die anderen Informationen anzuzeigen, die mit dem Proxy verknüpft sind.

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

Verwenden Sie die folgende Befehlsfolge, um weitere Details zu den Elementen anzuzeigen, die mit dem Proxy verknüpft sind:

1. Um eine Liste von Proxys anzufordern, führen Sie describe-db-proxies aus.
2. Führen Sie describe-db-proxy-target-groups `--db-proxy-name` aus, um Verbindungsparameter anzuzeigen, wie den maximalen Prozentsatz der Verbindungen, die der Proxy verwenden kann. Verwenden Sie den Namen des Proxys als Parameterwert.
3. Führen Sie zum Anzeigen der Details der des Aurora-DB-Clusters, die/der der zurückgegebenen Zielgruppe zugeordnet ist, describe-db-proxy-targets aus.

RDS-API

Verwenden Sie den Vorgang DescribeDBProxies, um Ihre Proxys mit der RDS-API anzuzeigen. Er gibt Werte des Datentyps DBProxy zurück.

Um Details zu den Verbindungseinstellungen für den Proxy anzuzeigen, verwenden Sie die Proxy-Bezeichner aus diesem Rückgabewert mit der Operation DescribeDBProxyTargetGroups. Er gibt Werte des Datentyps DBProxyTargetGroup zurück.

Sie können die dem Proxy zugeordnete RDS-Instance oder das Aurora-DB-Cluster mithilfe der Operation DescribeDBProxyTargets anzeigen. Er gibt Werte des Datentyps DBProxyTarget zurück.

Verbinden mit einer Datenbank über RDS-Proxy

Sie stellen eine Verbindung mit einem Aurora-DB-Cluster oder einem Cluster, der Aurora Serverless v2 verwendet, über einen Proxy in der Regel auf die gleiche Weise her, wie Sie eine direkte Verbindung mit der Datenbank herstellen. Der Hauptunterschied besteht darin, dass Sie anstelle des Cluster-Endpunkts den Proxy-Endpunkt angeben. Standardmäßig sind alle Proxyverbindungen read/write fähig und verwenden die Writer-Instanz. Wenn Sie den Reader-Endpunkt normalerweise für schreibgeschützte Verbindungen verwenden, können Sie einen zusätzlichen schreibgeschützten Endpunkt für den Proxy erstellen. Diesen Endpunkt können Sie auf die gleiche Weise verwenden. Weitere Informationen finden Sie unter Überblick über Proxy-Endpunkte.

Themen

- Mit Datenbankanmeldeinformationen eine Verbindung zu einer Datenbank herstellen
- Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen
- Überlegungen zum Herstellen einer Verbindung zu PostgreSQL

Mit Datenbankanmeldeinformationen eine Verbindung zu einer Datenbank herstellen

Gehen Sie wie folgt vor, um mithilfe von Datenbankanmeldedaten eine Verbindung zu einem Proxy herzustellen:

1. Suchen Sie den Proxy-Endpunkt. Im finden Sie den AWS-Managementkonsole Endpunkt auf der Detailseite für den entsprechenden Proxy. Mit dem AWS CLI können Sie den describe-db-proxiesBefehl verwenden. Im folgenden Beispiel wird gezeigt, wie dies geschieht.

```
# Add --output text to get output as a simple tab-separated list.
```

```
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. Geben Sie den Endpunkt als Hostparameter in der Verbindungszeichenfolge für Ihre Clientanwendung an. Geben Sie beispielsweise den Proxy-Endpunkt als Wert für die Option `mysql -h` oder `psql -h` an.
3. Geben Sie denselben Datenbankbenutzernamen und dasselbe Passwort an, die Sie normalerweise verwenden.

Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen

Wenn Sie die IAM-Authentifizierung mit RDS Proxy verwenden, haben Sie zwei Optionen für die Authentifizierung zwischen Ihrem Client und dem Proxy:

- Richten Sie Ihre Datenbankbenutzer so ein, dass sie sich mit normalen Benutzernamen und Passwörtern authentifizieren. RDS Proxy ruft den Benutzernamen und das Passwort von Secrets Manager ab. Die Verbindung von RDS-Proxy zur zugrunde liegenden Datenbank erfolgt nicht über IAM.

- Sie können auch die end-to-end IAM-Authentifizierung verwenden, bei der über den Proxy mithilfe von IAM eine Verbindung zu Ihrer Datenbank hergestellt wird, ohne dass Datenbankanmeldeinformationen erforderlich sind.

Um mittels IAM-Authentifizierung eine Verbindung mit dem RDS-Proxy herzustellen, befolgen Sie das gleiche allgemeine Verfahren wie für die IAM-Authentifizierung mit einem Aurora-DB-Cluster. Allgemeine Informationen zur Verwendung von IAM finden Sie unter Sicherheit in Amazon Aurora. Wenn Sie die end-to-end IAM-Authentifizierung verwenden, stellen Sie Ihrem DB-Benutzer das IAM-Authentifizierungs-Plugin zur Verfügung. Siehe Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung.

Zu den Hauptunterschieden bei der IAM-Nutzung für RDS-Proxy gehören die folgenden:

- Bei der Standard-IAM-Authentifizierung verfügen Datenbankbenutzer über reguläre Anmeldeinformationen in der Datenbank. Sie richten Secrets Manager-Geheimnisse ein, die diese Benutzernamen und Passwörter enthalten, und autorisieren RDS-Proxy, die Anmeldeinformationen von Secrets Manager abzurufen. Die IAM-Authentifizierung gilt für die Verbindung zwischen Ihrem Clientprogramm und dem Proxy. Der Proxy authentifiziert sich dann bei der Datenbank mit den Anmeldeinformationen aus Benutzername und Passwort, die von Secrets Manager abgerufen wurden.
- Mit der end-to-end IAM-Authentifizierung müssen Sie Secrets Manager nicht für Datenbankanmeldedaten konfigurieren. Die IAM-Authentifizierung gilt für die Verbindung zwischen dem Client zum Proxy und dem Proxy zur Datenbank.
- Anstelle des Instance-, Cluster- oder Leser-Endpunkts geben Sie den Proxy-Endpunkt an. Weitere Informationen zum Proxy-Endpunkt finden Sie unter Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung.
- Sie müssen Transport Layer Security (TLS)/Secure Sockets Layer (SSL) verwenden, wenn Sie mit IAM-Authentifizierung eine Verbindung zu einem Proxy herstellen.

Sie können einem bestimmten Benutzer Zugriff auf den Proxy gewähren, indem Sie die IAM-Richtlinie ändern. Ein Beispiel folgt.

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

> **Tip**
>
> Beachten Sie bei der Konfiguration der IAM-Authentifizierung für RDS-Proxyverbindungen die folgenden wichtigen Richtlinien, um Verbindungsprobleme zu vermeiden:
>
> - Gewähren Sie die `rds_iam` Rolle nicht und behalten Sie gleichzeitig die allgemeine Passwortauthentifizierung für denselben Datenbankbenutzer oder dieselbe Rolle bei.
> - Denken Sie daran, dass Clients zwar mithilfe der IAM-Authentifizierung eine Verbindung zum RDS-Proxy herstellen, RDS-Proxy jedoch immer mithilfe der Kennwortauthentifizierung über Secrets Manager eine Verbindung zur Datenbank herstellt.
> - Wenn Sie häufig Verbindungsabbrüche und Verbindungswiederverbindungen feststellen, entfernen Sie alle vorhandenen `rds_iam` Berechtigungen für den Benutzer oder die Rolle und verwenden Sie nur die Kennwortauthentifizierung.
> - Stellen Sie sicher, dass Ihre Passwortrichtlinie die SCRAM-SHA-256-Anforderungen für sichere Zeichen erfüllt.
>
> Die Kombination von IAM- und Kennwortauthentifizierungsmethoden für denselben Datenbankbenutzer kann zu Verbindungsinstabilität führen.

Überlegungen zum Herstellen einer Verbindung zu PostgreSQL

Stellen Sie beim Erstellen eines neuen PostgreSQL-Datenbankbenutzers für die Verbindung mit dem RDS-Proxy sicher, dass Sie die Benutzerberechtigung `CONNECT` für die Datenbank gewähren. Ohne diese kann der Benutzer keine Verbindung herstellen. Weitere Informationen finden Sie unter the section called “Hinzufügen eines neuen Datenbankbenutzers zu einer PostgreSQL-Datenbank bei Verwendung von RDS-Proxy”.

Wenn ein Client beginnt, eine Verbindung mit einer PostgreSQL-Datenbank herzustellen, sendet er eine Startmeldung. Diese Nachricht enthält Paare von Parameternamen und Wertzeichenfolgen. Weitere Informationen finden Sie unter `StartupMessage` in den PostgreSQL-Nachrichtenformaten in der PostgreSQL-Dokumentation.

Wenn Sie eine Verbindung über einen RDS-Proxy herstellen, kann die Startmeldung die folgenden aktuell erkannten Parameter enthalten:

- `user`

- `database`

Die Startmeldung kann auch die folgenden zusätzlichen Laufzeitparameter enthalten:

- `application_name`
- `client_encoding`
- `DateStyle`
- `TimeZone`
- `extra_float_digits`
- `search_path`

Weitere Informationen zum PostgreSQL-Messaging finden Sie im Frontend/Backend-Protokoll in der PostgreSQL-Dokumentation.

Für PostgreSQL empfehlen wir bei Verwendung von JDBC Folgendes, um Pinning zu vermeiden:

- Stellen Sie den JDBC-Verbindungsparameter `assumeMinServerVersion` mindestens auf `9.0` ein, um Pinning zu vermeiden. Dadurch wird verhindert, dass der JDBC-Treiber beim Starten der Verbindung einen zusätzlichen Roundtrip durchführt, wenn `SET extra_float_digits = 3` ausgeführt wird.
- Setzen Sie den JDBC-Verbindungsparameter `ApplicationName` auf *`any/your-application-name`*, um Pinning zu vermeiden. Dadurch wird verhindert, dass der JDBC-Treiber beim Starten der Verbindung einen zusätzlichen Roundtrip ausführt, wenn er ausführ `SET application_name = "PostgreSQL JDBC Driver"`. Beachten Sie, dass der JDBC-Parameter `ApplicationName`, der PostgreSQL-`StartupMessage`-Parameter aber `application_name` ist.

Weitere Informationen finden Sie unter Vermeiden von Pinning beim RDS-Proxy. Weitere Informationen zum Herstellen einer Verbindung mit JDBC finden Sie unter Verbinden mit der Datenbank in der PostgreSQL-Dokumentation.

Verwalten eines RDS-Proxy

Dieser Abschnitt enthält Informationen zum Verwalten des Betriebs und der Konfiguration von RDS-Proxys. Diese Verfahren helfen Ihrer Anwendung, Datenbankverbindungen möglichst effizient zu nutzen und eine maximale Wiederverwendung der Verbindung zu erzielen. Je mehr Sie die

Wiederverwendung der Verbindung nutzen können, desto mehr CPU- und Arbeitsspeicher-Overhead können Sie sparen. Dies reduziert wiederum die Latenz für Ihre Anwendung und ermöglicht es der Datenbank, mehr Ressourcen für die Verarbeitung von Anwendungsanforderungen zu verwenden.

Themen

- Ändern eines RDS-Proxy
- Hinzufügen eines neuen Datenbankbenutzers bei Verwendung von RDS-Proxy
- Umstellung von der Standard-IAM-Authentifizierung zur end-to-end IAM-Authentifizierung für RDS Proxy
- Überlegungen zu RDS-Proxy-Verbindungen
- Vermeiden von Pinning beim RDS-Proxy
- Löschen eines RDS-Proxy

Ändern eines RDS-Proxy

Sie können bestimmte Einstellungen ändern, die einem Proxy zugeordnet sind, nachdem Sie den Proxy erstellt haben. Dazu ändern Sie den Proxy selbst, die zugehörige Zielgruppe oder beides. Jedem Proxy ist eine Zielgruppe zugeordnet.

AWS-Managementkonsole

> **Important**
>
> Die Werte in den Feldern Client authentication type (Client-Authentifizierungstyp) und IAM authentication (IAM-Authentifizierung) gelten für alle Secrets-Manager-Secrets, die diesem Proxy zugeordnet sind. Um für jedes Geheimnis unterschiedliche Werte anzugeben, ändern Sie Ihren Proxy, indem Sie stattdessen die AWS CLI oder die API verwenden.

So ändern Sie die Einstellungen für einen Proxy:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Proxies (Proxys).
3. Wählen Sie in der Liste der Proxys den Proxy aus, dessen Einstellungen Sie ändern möchten, oder gehen Sie zur Detailseite.

4. Wählen Sie für Actions (Aktionen) die Option Modify (Ändern) aus.
5. Geben Sie die zu ändernden Eigenschaften ein, oder wählen Sie sie aus. Sie können folgende Formen angeben:

   - Proxy-Identifikator - Benennen Sie den Proxy um, indem Sie einen neuen Identifikator eingeben.
   - Zeitüberschreitung für Client-Verbindungsleerlauf— Geben Sie einen Zeitraum für das Timeout für die Clientleerlauf ein.
   - IAM-Rolle - Ändern Sie die IAM-Rolle, die zum Abrufen der Geheimnisse von Secrets Manager verwendet wird.

     > **Note**
     >
     > Sie können keine neue IAM-Rolle erstellen, wenn Sie das Standardauthentifizierungsschema auf IAM-Authentifizierung festlegen.

   - Secrets Manager-Secrets— Hinzufügen oder entfernen Sie Secrets Manager Geheimnisse. Diese Secrets entsprechen Datenbankbenutzernamen und Passwörtern.
   - Client-Authentifizierungstyp — Ändern Sie den Authentifizierungstyp für Client-Verbindungen zum Proxy.
   - IAM authentication (IAM-Authentifizierung) – Verlangen oder unterbinden Sie die IAM-Authentifizierung für Verbindungen mit dem Proxy.
   - Standardauthentifizierungsschema — Ändern Sie das Standardauthentifizierungsschema, das der Proxy für Client-Verbindungen zum Proxy und Verbindungen vom Proxy zur zugrunde liegenden Datenbank verwendet.
   - Require Transport Layer Security— Aktivieren oder deaktivieren Sie die Anforderung für Transport Layer Security (TLS).
   - VPC-Sicherheitsgruppe - Fügen Sie VPC-Sicherheitsgruppen hinzu oder entfernen Sie sie, damit der Proxy sie verwenden kann.
   - Aktivieren der erweiterten Protokollierung— Aktivieren oder deaktivieren Sie die erweiterte Protokollierung.
6. Wählen Sie Ändern aus.

Wenn Sie die Einstellungen, die Sie ändern möchten, in der Liste nicht gefunden haben, verwenden Sie das folgende Verfahren, um die Zielgruppe für den Proxy zu aktualisieren. Die Zielgruppe, die

einem Proxy zugeordnet ist, steuert die Einstellungen für die physischen Datenbankverbindungen. Jedem Proxy ist eine Zielgruppe mit dem Namen `default` zugeordnet, die automatisch zusammen mit dem Proxy erstellt wird. Sie können die Standardzielgruppe nicht umbenennen.

Sie können die Zielgruppe nur über die Proxy-Detailseite ändern, nicht über die Liste auf der Seite Proxies (Proxys).

So ändern Sie die Einstellungen für eine Proxy-Zielgruppe:

1. Gehen Sie auf der Seite Proxys zur Detailseite für einen Proxy.
2. Wählen Sie für Target groups (Zielgruppen) den `default`-Link aus. Derzeit haben alle Proxys eine einzelne Zielgruppe mit dem Namen `default`.
3. Wählen Sie auf der Detailseite für die Standard-Zielgruppe die Option Modify (Ändern).
4. Wählen Sie neue Einstellungen für die Eigenschaften aus, die Sie ändern können:

   - Datenbank – Wählen Sie einen anderen Aurora-Cluster aus.
   - Maximale Verbindungen des Verbindungspools - Legen Sie fest, wie viel Prozent der maximal verfügbaren Verbindungen der Proxy nutzen kann.
   - Vortrags-Anheftungsfilter – Wählen Sie optional einen Vortrags-Anheftungsfilter aus. Dadurch werden die standardmäßigen Sicherheitsmaßnahmen für das Multiplexing von Datenbankverbindungen über Client-Verbindungen hinweg umgangen. Derzeit wird die Einstellung für PostgreSQL nicht unterstützt. Die einzige Wahlmöglichkeit ist `EXCLUDE_VARIABLE_SETS`.

     Die Aktivierung dieser Einstellung kann dazu führen, dass sich Sitzungsvariablen einer Verbindung auf andere Verbindungen auswirken. Dies kann zu Fehlern oder Problemen mit der Korrektheit führen, wenn Ihre Abfragen von Sitzungsvariablenwerten abhängen, die außerhalb der aktuellen Transaktion festgelegt wurden. Erwägen Sie, diese Option zu verwenden, nachdem Sie sich vergewissert haben, dass Ihre Anwendungen Datenbankverbindungen über mehrere Client-Verbindungen gemeinsam nutzen können.

     Die folgenden Muster können als sicher angesehen werden:

     - `SET`-Anweisungen, bei denen der effektive Wert der Sitzungsvariablen nicht geändert wird, d. h. dass keine Änderung an der Sitzungsvariablen vorgenommen wird.
     - Sie ändern den Wert der Sitzungsvariablen und führen eine Anweisung in derselben Transaktion aus.

Weitere Informationen finden Sie unter Vermeiden von Pinning beim RDS-Proxy.

- Connection borrow timeout— Passen Sie das Timeout-Intervall der Verbindung an. Diese Einstellung gilt, wenn für den Proxy bereits die maximale Anzahl von Verbindungen verwendet wird. Diese Einstellung legt fest, wie lange der Proxy wartet, bis eine Verbindung verfügbar ist, bevor ein Timeout-Fehler zurückgegeben wird.
- Initialisierungsabfrage. (Optional) Fügen Sie eine Initialisierungsabfrage hinzu oder ändern Sie die aktuelle. Sie können eine oder mehrere SQL-Anweisungen für die Ausführung durch den Proxy beim Öffnen jeder neuen Datenbankverbindung festlegen. Diese Einstellung wird normalerweise mit SET-Anweisungen verwendet, um sicherzustellen, dass jede Verbindung identische Einstellungen aufweist. Stellen Sie sicher, dass die von Ihnen hinzugefügte Abfrage gültig ist. Verwenden Sie Komma als Trennzeichen, um mehrere Variablen in eine einzelne SET-Anweisung einzuschließen. Zum Beispiel:

  ```
  SET variable1=value1, variable2=value2
  ```

  Verwenden Sie für mehrere Anweisungen Semikola als Trennzeichen.

Sie können bestimmte Eigenschaften, z. B. die Zielgruppenkennung und die Datenbank-Engine, nicht ändern.

5. Wählen Sie Modify target group (Zielgruppe ändern).

AWS CLI

Um einen Proxy mit dem zu ändern AWS CLI, verwenden Sie die Befehle modify-db-proxymodify-db-proxy-target-group deregister-db-proxy-targets, und register-db-proxy-targets.

Mit dem Befehl `modify-db-proxy` können Sie Eigenschaften wie die folgenden ändern:

- Der Satz von Secrets Manager-Secrets, die vom Proxy verwendet werden.
- Ob TLS erforderlich ist.
- Das Timeout für Client-Leerlauf.
- Ob zusätzliche Informationen aus SQL-Anweisungen zum Debuggen protokolliert werden sollen.
- Die IAM-Rolle, die zum Abrufen von Secrets Manager-Secrets verwendet wird.
- Die vom Proxy verwendeten Sicherheitsgruppen.
- Das dem Proxy zugeordnete Standardauthentifizierungsschema.

Das folgende Beispiel zeigt, wie ein vorhandener Proxy umbenannt wird.

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

Um verbindungsbezogene Einstellungen zu ändern oder die Zielgruppe umzubenennen, verwenden Sie den Befehl `modify-db-proxy-target-group`. Derzeit haben alle Proxys eine einzelne Zielgruppe mit dem Namen `default`. Wenn Sie mit dieser Zielgruppe arbeiten, geben Sie den Namen des Proxys und `default` als Name der Zielgruppe an. Sie können die Standardzielgruppe nicht umbenennen.

Das folgende Beispiel zeigt, wie Sie zuerst die `MaxIdleConnectionsPercent`-Einstellung für einen Proxy überprüfen und dann mithilfe der Zielgruppe ändern.

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'

{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
```

```
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
}
```

Mit den Befehlen `deregister-db-proxy-targets` und `register-db-proxy-targets` ändern Sie, welchen Aurora-DB-Clustern der Proxy über die Zielgruppe zugeordnet ist. Derzeit kann jeder Proxy eine Verbindung zu einer einem Aurora-DB-Cluster herstellen. Die Zielgruppe verfolgt die Verbindungsdetails für alle alle DB-Instances in einem Aurora-Cluster.

Das folgende Beispiel beginnt mit einem Proxy, der einem Aurora MySQL-Cluster mit dem Namen zugeordnet is `cluster-56-2020-02-25-1399`. Das Beispiel zeigt, wie der Proxy so geändert wird, dass er eine Verbindung zu einem anderen Cluster namens herstellen kan `provisioned-cluster`.

Wenn Sie mit einem Aurora-DB-Cluster arbeiten, geben Sie die Option `--db-cluster-identifier` an.

Im folgenden Beispiel wird ein Aurora MySQL-Proxy geändert. Ein Aurora PostgreSQL-Proxy hat Port 5432.

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
```

```
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}

aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
```

```
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

RDS-API

Um einen Proxy mithilfe der RDS-API zu ändern, verwenden Sie die Operationen Modify DBProxy, Modify DBProxy TargetGroup, Deregister DBProxy Targets und Register DBProxy Targets.

Mit `ModifyDBProxy` können Sie Eigenschaften wie die folgenden ändern:

- Der Satz von Secrets Manager-Secrets, die vom Proxy verwendet werden.
- Ob TLS erforderlich ist.
- Das Timeout für Client-Leerlauf.
- Ob zusätzliche Informationen aus SQL-Anweisungen zum Debuggen protokolliert werden sollen.
- Die IAM-Rolle, die zum Abrufen von Secrets Manager-Secrets verwendet wird.
- Die vom Proxy verwendeten Sicherheitsgruppen.

Mit `ModifyDBProxyTargetGroup` können Sie verbindungsbezogene Einstellungen ändern. Derzeit haben alle Proxys eine einzelne Zielgruppe mit dem Namen `default`. Wenn Sie mit dieser Zielgruppe arbeiten, geben Sie den Namen des Proxys und `default` als Name der Zielgruppe an. Sie können die Standardzielgruppe nicht umbenennen.

Mit `DeregisterDBProxyTargets` und `RegisterDBProxyTargets` ändern Sie, welcher welchem Aurora-Cluster der Proxy über die Zielgruppe zugeordnet ist. Derzeit kann jeder Proxy eine Verbindung mit einem Aurora-DB-Cluster herstellen. Die Zielgruppe verfolgt die Verbindungsdetails für die DB-Instances in einem Aurora-Cluster.

Hinzufügen eines neuen Datenbankbenutzers bei Verwendung von RDS-Proxy

In einigen Fällen können Sie einen neuen Datenbankbenutzer zu einem Aurora-Cluster mit Proxy-Zuordnung hinzufügen. Fahren Sie fort, je nachdem, ob Sie die Standardauthentifizierung mit Secrets Manager Manager-Geheimnissen oder die end-to-end IAM-Authentifizierung verwenden.

Wenn Sie die Standard-IAM-Authentifizierung verwenden, folgen Sie diesen Anweisungen:

1. Erstellen Sie ein neues Secrets-Manager-Secret, indem Sie das unter beschriebene Verfahren verwende Datenbankanmeldedaten für RDS Proxy einrichten.
2. Aktualisieren Sie die IAM-Rolle, um RDS-Proxy Zugriff auf das neue Secrets Manager-Geheimnis zu gewähren. Aktualisieren Sie dazu den Ressourcenabschnitt der IAM-Rollenrichtlinie.
3. Ändern Sie den RDS-Proxy, um das neue Secret von Secrets Manager unter Secrets Manager Secrets hinzuzufügen.
4. Wenn der neue Benutzer an die Stelle eines vorhandenen Benutzers tritt, aktualisieren Sie die Anmeldeinformationen, die im Secrets Manager-Secret des Proxys für den vorhandenen Benutzer gespeichert sind.

Wenn Sie die end-to-end IAM-Authentifizierung verwenden, müssen Sie den Datenbankbenutzer erstellen und IAM-Berechtigungen konfigurieren. Führen Sie dazu die folgenden Schritte aus:

1. Erstellen Sie einen neuen Datenbankbenutzer in Ihrer Datenbank, der dem IAM-Benutzer- oder Rollennamen entspricht, den Sie für die Authentifizierung verwenden möchten.
2. Stellen Sie sicher, dass der Datenbankbenutzer mit dem IAM-Authentifizierungs-Plugin in der Datenbank konfiguriert ist. Siehe Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung.
3. Aktualisieren Sie die IAM-Richtlinie, um dem IAM-Benutzer oder der IAM-Rolle die `rds-db:connect` Berechtigung zu erteilen, wie unter beschrieben. Eine IAM-Richtlinie für end-to-end die IAM-Authentifizierung erstellen
4. Stellen Sie sicher, dass Ihr Proxy so konfiguriert ist, dass er die IAM-Authentifizierung als Standardauthentifizierungsschema verwendet.

Mit der end-to-end IAM-Authentifizierung müssen Sie Datenbankanmeldedaten nicht in Secrets Manager Manager-Geheimnissen verwalten, da IAM-Anmeldeinformationen für die Authentifizierung vom Client zum Proxy und vom Proxy zur Datenbank verwendet werden.

Hinzufügen eines neuen Datenbankbenutzers zu einer PostgreSQL-Datenbank bei Verwendung von RDS-Proxy

Gehen Sie beim Hinzufügen eines neuen Benutzers zu Ihrer PostgreSQL-Datenbank wie folgt vor, wenn Sie den folgenden Befehl ausgeführt haben:

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

Gewähren Sie dem Benutzer `rdsproxyadmin` die `CONNECT`-Berechtigung, damit der Benutzer Verbindungen in der Zieldatenbank überwachen kann.

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

Sie können auch anderen Zieldatenbankbenutzern die Durchführung von Zustandsprüfungen ermöglichen, indem Sie im obigen Befehl `rdsproxyadmin` auf den Datenbankbenutzer ändern.

Ändern des Passworts für einen Datenbankbenutzer bei Verwendung von RDS-Proxy

In einigen Fällen können Sie das Passwort für einen Datenbankbenutzer in einem Aurora-Cluster mit Proxy-Zuordnung ändern. Aktualisieren Sie dazu das entsprechende Secrets Manager-Secret mit dem neuen Passwort.

Wenn Sie die end-to-end IAM-Authentifizierung verwenden, müssen Sie keine Passwörter in Secrets Manager Secrets aktualisieren.

Umstellung von der Standard-IAM-Authentifizierung zur end-to-end IAM-Authentifizierung für RDS Proxy

Wenn Sie derzeit die Standard-IAM-Authentifizierung für den RDS-Proxy verwenden, bei der sich Clients mit IAM am Proxy authentifizieren, der Proxy jedoch mithilfe von Geheimnissen eine Verbindung zur Datenbank herstellt, können Sie zur IAM-Authentifizierung migrieren, bei der client-to-proxy sowohl proxy-to-database Verbindungen als auch Verbindungen die end-to-end IAM-Authentifizierung verwenden.

Um zur IAM-Authentifizierung zu wechseln end-to-end

1. Aktualisieren Sie die IAM-Rollenberechtigungen für RDS Proxy

   Erstellen Sie eine aktualisierte Proxy-Berechtigungsrichtlinie, die sowohl Secrets Manager als auch `rds:db-connect` Berechtigungen umfasst:

```
# Create updated proxy permission policy
cat > updated-proxy-policy.json # EOF
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetSecretsValue",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:secretName-1234f"
      ]
    },
    {
      "Sid": "RdsDBConnect",
      "Action": [
        "rds-db:connect"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:rds-db:us-east-1:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/
jane_doe"
      ]
    }
  ]
}
```

Aktualisieren Sie den Proxy Ihrer Rollenrichtlinie:

```
aws iam put-role-policy \
            --role-name RDSProxyRole \
            --policy-name UpdatedProxyPermissions \
            --policy-document file://updated-proxy-policy.json
```

2. Ändern Sie Ihren RDS-Proxy, um die end-to-end IAM-Authentifizierung zu aktivieren

```
aws rds modify-db-proxy \
  --db-proxy-name my-database-proxy \
```

```
  --default-auth-scheme IAM_AUTH \
  --region us-east-1
```

Stellen Sie sicher, dass der RDS-Proxystatus „Verfügbar“ `DefaultAuthScheme` lautet, `IAM_AUTH` bevor Sie fortfahren, um sicherzustellen, dass während der Migration keine Ausfallzeiten auftreten.

```
aws rds describe-db-proxies --db-proxy-name my-database-proxy --region us-east-1
```

Erwartete Ausgabe:

```
{
  "DBProxies": [
    {
      "DBProxyName": "my-database-proxy",
      "DBProxyArn": "arn:aws:rds:us-east-1:123456789012:db-
proxy:prx-0123456789abcdef",
      "Status": "available",
      ...
      "DefaultAuthScheme": "IAM_AUTH"
    }
  ]
}
```

3. Aktivieren Sie die IAM-Authentifizierung in der Datenbank

```
aws rds modify-db-cluster \
  --db-cluster-identifier my-database-cluster \
  --enable-iam-database-authentication \
  --region us-east-1
```

4. Konfigurieren Sie den Datenbankbenutzer für die IAM-Authentifizierung

   Für Aurora PostgreSQL:

```
GRANT rds_iam TO jane_doe;
```

   Für Aurora MySQL:

```
ALTER USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

5. Der Code Ihrer Client-Anwendung muss sich nicht ändern. Der Verbindungsvorgang bleibt derselbe:

   Für Aurora PostgreSQL:

```
# Generate authentication token
export PGPASSWORD=$(aws rds generate-db-auth-token \
  --hostname my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 \
  --port 5432 \
  --username jane_doe \
  --region us-east-1)

# Connect to database through proxy
psql "host=my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 port=5432 user=jane_doe dbname=postgres password=$PGPASSWORD sslmode=require
 sslrootcert=us-east-1-bundle.pem"
```

   Für Aurora MySQL:

```
# Generate authentication token
export MYSQL_PWD=$(aws rds generate-db-auth-token \
  --hostname my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 \
  --port 3306 \
  --username jane_doe \
  --region us-east-1)

# Connect to database through proxy
mysql -h my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com \
  -P 3306 \
  -u jane_doe \
  --ssl-ca=us-east-1-bundle.pem \
  --enable-cleartext-plugin
```

Überlegungen zu RDS-Proxy-Verbindungen

Konfigurieren der Verbindungseinstellungen

Um das Verbindungspooling von RDS-Proxy anzupassen, können Sie die folgenden Einstellungen ändern:

- IdleClientTimeout
- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

IdleClientTimeout

Sie können angeben, wie lange eine Client-Verbindung inaktiv sein kann, bevor der Proxy sie schließt. Der Standardwert ist 1.800 Sekunden (30 Minuten).

Eine Client-Verbindung gilt als inaktiv, wenn die Anwendung innerhalb der angegebenen Zeit nach Abschluss der vorherigen Anforderung keine neue Anforderung absendet. Die zugrunde liegende Datenbankverbindung bleibt offen und wird an den Verbindungspool zurückgegeben. Somit ist sie für neue Clientverbindungen verfügbar. Verringern Sie das Timeout für inaktive Client-Verbindungen, wenn der Proxy veraltete Verbindungen proaktiv entfernen soll. Wenn Ihr Workload häufige Verbindungen mit dem Proxy herstellt, erhöhen Sie das Timeout für inaktive Client-Verbindungen, um sich die Kosten für das Herstellen von Verbindungen zu sparen.

Diese Einstellung wird durch das Feld Idle client connection timeout (Zeitüberschreitung bei Client-Verbindungsinaktivität) in der RDS-Konsole und die Einstellung `IdleClientTimeout` in der AWS CLI und der API dargestellt. Informationen dazu, wie Sie den Wert des Felds Idle client connection timeout (Zeitüberschreitung bei Client-Verbindungsinaktivität) in der RDS-Konsole ändern, finden Sie unter AWS-Managementkonsole. Informationen dazu, wie Sie den Wert der Einstellung `IdleClientTimeout` ändern, finden Sie unter dem CLI-Befehl modify-db-proxy oder der API-Operation ModifyDBProxy.

MaxConnectionsPercent

Sie können die Anzahl der Verbindungen beschränken, die ein RDS-Proxy mit der Zieldatenbank herstellen kann. Sie geben das Limit als Prozentsatz der maximal für Ihre Datenbank

verfügbaren Verbindungen an. Diese Einstellung wird durch das Feld Connection pool maximum connections (Max. Verbindungen Verbindungspool) in der RDS-Konsole und die Einstellung `MaxConnectionsPercent` in der AWS CLI und der API dargestellt.

Der Wert `MaxConnectionsPercent` wird als Prozentsatz der Einstellung `max_connections` für den Aurora-DB-Cluster ausgedrückt, die von der Zielgruppe verwendet wird. Der Proxy erstellt nicht alle diese Verbindungen im Voraus. Bei dieser Einstellung kann sich der Proxy das Recht vorbehalten, diese Verbindungen dann herzustellen, wenn der Workload sie benötigt.

Beispielsweise legt RDS-Proxy für ein registriertes Datenbankziel, für das `max_connections` auf 1000 und `MaxConnectionsPercent` auf 95 festgelegt ist, 950 Verbindungen als Obergrenze für gleichzeitige Verbindungen mit diesem Datenbankziel fest.

Ein häufiger Nebeneffekt, der auftritt, wenn Ihre Workload die maximale Anzahl zulässiger Datenbankverbindungen erreicht, ist ein Anstieg der allgemeinen Abfragelatenz in Verbindung mit einer Erhöhung der Metrik `DatabaseConnectionsBorrowLatency`. Sie können die aktuell verwendeten Datenbankverbindungen und die Gesamtzahl der zulässigen Datenbankverbindungen überwachen, indem Sie die Metriken `DatabaseConnections` und `MaxDatabaseConnectionsAllowed` vergleichen.

Beachten Sie für das Festlegen dieses Parameters die folgenden bewährten Methoden:

- Sorgen Sie für ausreichend Verbindungsspielraum für Änderungen des Workload-Musters. Es wird empfohlen, den Parameter mindestens 30 % über Ihrer zuletzt überwachten maximalen Nutzung einzustellen. Da RDS-Proxy die Kontingente für Datenbankverbindungen auf mehrere Knoten neu verteilt, können interne Kapazitätsänderungen mindestens 30 % Spielraum für zusätzliche Verbindungen erfordern, um erhöhte Ausleihlatenzen zu vermeiden.
- RDS-Proxy reserviert eine bestimmte Anzahl von Verbindungen für die aktive Überwachung, um schnelles Failover, Weiterleitung von Datenverkehr und interne Operationen zu unterstützen. Die Metrik `MaxDatabaseConnectionsAllowed` umfasst diese reservierten Verbindungen nicht. Sie stellt die Anzahl der Verbindungen dar, die für den Workload verfügbar sind, und kann niedriger sein als der aus der Einstellung `MaxConnectionsPercent` abgeleitete Wert.

  Die folgenden Mindestwerte werden für `MaxConnectionsPercent` empfohlen:

  - db.t3.small: 100
  - db.t3.medium: 55
  - db.t3.large: 35
  - db.r3.large oder höher: 20

Wenn mehrere Ziel-Instances bei RDS-Proxy registriert sind, z. B. ein Aurora-Cluster mit Leserknoten, legen Sie den Mindestwert auf der Grundlage der kleinsten registrierten Instance fest.

Informationen dazu, wie Sie den Wert des Felds Connection pool maximum connections (Max. Verbindungen Verbindungspool) in der RDS-Konsole ändern, finden Sie unter AWS-Managementkonsole. Informationen dazu, wie Sie den Wert der Einstellung `MaxConnectionsPercent` ändern, finden Sie unter dem CLI-Befehl modify-db-proxy-target-group oder der API-Operation ModifyDBProxyTargetGroup.

> **Important**
>
> Wenn der DB-Cluster Teil einer globalen Datenbank mit aktivierter Schreibweiterleitung ist, reduzieren Sie den `MaxConnectionsPercent`-Wert Ihres Proxys um das Kontingent, das der Schreibweiterleitung zugewiesen ist. Das Kontingent für die Schreibweiterleitung ist im DB-Cluster-Parameter `aurora_fwd_writer_max_connections_pct` festgelegt. Informationen zur Schreibweiterleitung finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

Informationen zu Datenbankverbindungslimits finden Sie unter Maximale Verbindungen zu einer Aurora-MySQL-DB-Instance und Maximale Verbindungen zu einer Aurora-PostgreSQL-DB-Instance.

MaxIdleConnectionsPercent

Sie können die Anzahl inaktiver Datenbankverbindungen festlegen, die RDS-Proxy im Verbindungspool behalten kann. RDS-Proxy betrachtet eine Datenbankverbindung im Pool standardmäßig als inaktiv, wenn fünf Minuten lang keine Aktivität für die Verbindung aufgetreten ist.

Der Wert `MaxIdleConnectionsPercent` wird als Prozentsatz der Einstellung `max_connections` für die Zielgruppe der RDS-DB-Instance verwendet. Der Standardwert ist 50 Prozent von `MaxConnectionsPercent` und die Obergrenze ist der Wert `MaxConnectionsPercent`. Wenn `MaxConnectionsPercent` beispielsweise 80 ist, ist der Standardwert von `MaxIdleConnectionsPercent` 40.

Bei einem hohen Wert lässt der Proxy einen hohen Prozentsatz an ungenutzten Datenbankverbindungen offen. Bei einem niedrigen Wert schließt der Proxy einen hohen Prozentsatz von inaktiven Datenbankverbindungen. Wenn Ihre Workloads unvorhersehbar sind, sollten Sie

erwägen, einen hohen Wert für `MaxIdleConnectionsPercent` festzulegen. Dies bedeutet, dass RDS-Proxy Aktivitätsspitzen verarbeiten kann, ohne viele neue Datenbankverbindungen zu öffnen.

Diese Einstellung wird durch die Einstellung `MaxIdleConnectionsPercent` von `DBProxyTargetGroup` in der AWS CLI und der API dargestellt. Informationen dazu, wie Sie den Wert der Einstellung `MaxIdleConnectionsPercent` ändern, finden Sie unter dem CLI-Befehl modify-db-proxy-target-group oder der API-Operation ModifyDBProxyTargetGroup.

Informationen zu Datenbankverbindungslimits finden Sie unter Maximale Verbindungen zu einer Aurora-MySQL-DB-Instance und Maximale Verbindungen zu einer Aurora-PostgreSQL-DB-Instance.

ConnectionBorrowTimeout

Sie können angeben, wie lange RDS-Proxy warten soll, bis eine Datenbankverbindung im Verbindungspool verfügbar ist, bevor ein Timeout-Fehler zurückgegeben wird. Standardmäßig sind 120 Sekunden festgelegt. Diese Einstellung greift, wenn die maximale Anzahl von Verbindungen erreicht ist und daher keine Verbindungen im Verbindungspool verfügbar sind. Dies gilt auch, wenn keine geeignete Datenbank-Instance zur Bearbeitung der Anfrage verfügbar ist, beispielsweise wenn gerade ein Failover-Vorgang durchgeführt wird. Mit dieser Einstellung können Sie die beste Wartezeit für Ihre Anwendung festlegen, ohne das Abfrage-Timeout im Anwendungscode zu ändern.

Diese Einstellung wird durch das Feld Connection borrow timeout (Zeitüberschreitung für die Verbindung) in der RDS-Konsole oder die Einstellung `ConnectionBorrowTimeout` von `DBProxyTargetGroup` in der AWS CLI oder API dargestellt. Informationen dazu, wie Sie den Wert des Felds Connection borrow timeout (Zeitüberschreitung für die Verbindung) in der RDS-Konsole ändern, finden Sie unter AWS-Managementkonsole. Informationen dazu, wie Sie den Wert der Einstellung `ConnectionBorrowTimeout` ändern, finden Sie unter dem CLI-Befehl modify-db-proxy-target-group oder der API-Operation ModifyDBProxyTargetGroup.

Client- und Datenbankverbindungen

Verbindungen von Ihrer Anwendung zum RDS-Proxy werden als Client-Verbindungen bezeichnet. Verbindungen von einem Proxy zur Datenbank sind Datenbankverbindungen. Bei Verwendung von RDS-Proxy werden Client-Verbindungen am Proxy beendet, während Datenbankverbindungen innerhalb des RDS-Proxys verwaltet werden.

Anwendungsseitiges Verbindungspooling kann den Vorteil bieten, dass der wiederkehrende Verbindungsaufbau zwischen Ihrer Anwendung und dem RDS-Proxy reduziert wird.

Berücksichtigen Sie die folgenden Konfigurationsaspekte, bevor Sie einen anwendungsseitigen Verbindungspool implementieren:

- Maximale Client-Verbindungsdauer: RDS-Proxy erzwingt eine maximale Client-Verbindungsdauer von 24 Stunden. Dieser Wert kann nicht konfiguriert werden. Konfigurieren Sie Ihren Pool mit einer maximalen Verbindungsdauer von weniger als 24 Stunden, um unerwartete Verbindungsabbrüche beim Client zu vermeiden.
- Timeout bei Inaktivität der Client-Verbindung: RDS-Proxy erzwingt eine maximale Leerlaufzeit für Client-Verbindungen. Konfigurieren Sie Ihren Pool mit einem Timeout für inaktive Verbindungen, der niedriger ist als die Einstellung für das Leerlaufzeitlimit für Client-Verbindungen für den RDS-Proxy, um unerwartete Verbindungsabbrüche zu vermeiden.

Die maximale Anzahl von Client-Verbindungen, die in Ihrem anwendungsseitigen Verbindungs-Pool konfiguriert ist, muss nicht auf die Einstellung max_connections für RDS-Proxy beschränkt sein.

Das Pooling von Client-Verbindungen führt zu einer längeren Client-Verbindungsdauer. Wenn es bei Ihren Verbindungen zum Pinning kommt, kann das Pooling von Client-Verbindungen die Effizienz des Multiplexings verringern. Client-Verbindungen, die im anwendungsseitigen Verbindungs-Pool gepinnt, aber inaktiv sind, behalten weiterhin eine Datenbankverbindung und verhindern, dass die Datenbankverbindung von anderen Client-Verbindungen wiederverwendet wird. Überprüfen Sie Ihre Proxy-Protokolle, um zu ermitteln, ob bei Ihren Verbindungen ein Pinning auftritt.

> **Note**
>
> RDS-Proxy schließt Datenbankverbindungen nach 24 Stunden, wenn sie nicht mehr verwendet werden. Der Proxy führt diese Aktion unabhängig vom Wert der Einstellung für maximale Leerlaufverbindungen aus.

Vermeiden von Pinning beim RDS-Proxy

Multiplexing ist effizienter, wenn Datenbankanforderungen nicht auf Statusinformationen aus früheren Anforderungen angewiesen sind. In diesem Fall kann RDS-Proxy eine Verbindung zum Abschluss jeder Transaktion wiederverwenden. Beispiele für solche Zustandsinformationen sind die meisten Variablen und Konfigurationsparameter, die Sie durch `SET`-oder `SELECT`-Anweisungen ändern können. SQL-Transaktionen auf einer Clientverbindung können standardmäßig zwischen zugrunde liegenden Datenbankverbindungen Multiplexing durchführen.

Ihre Verbindungen zum Proxy können einen Status eingeben, der als Pinning (Fixieren) bezeichnet wird. Wenn eine Verbindung angeheftet wird, verwendet jede spätere Transaktion dieselbe zugrunde liegende Datenbankverbindung, bis die Sitzung beendet ist. Andere Clientverbindungen können diese Datenbankverbindung auch erst dann wieder verwenden, wenn die Sitzung beendet ist. Die Sitzung wird beendet, wenn die Clientverbindung unterbrochen wird.

RDS-Proxy heftet automatisch eine Clientverbindung an eine bestimmte DB-Verbindung an, wenn eine Sitzungsstatusänderung erkannt wird, die für andere Sitzungen nicht geeignet ist. Das Fixieren verringert die Effektivität der Wiederverwendung der Verbindung. Wenn alle oder fast alle Verbindungen fixiert sind, können Sie Ihren Anwendungscode oder Ihre Workload ändern, um dafür zu sorgen, dass Fixierungen weniger erforderlich sind.

Ihre Anwendung ändert beispielsweise eine Sitzungsvariable oder einen Konfigurationsparameter. In diesem Fall können sich spätere Anweisungen darauf verlassen, dass die neue Variable oder der neue Parameter wirksam ist. Wenn also RDS-Proxy Anforderungen verarbeitet, um Sitzungsvariablen oder Konfigurationseinstellungen zu ändern, wird diese Sitzung an die DB-Verbindung fixiert. Auf diese Weise bleibt der Sitzungsstatus für alle späteren Transaktionen in derselben Sitzung gültig.

Bei Datenbank-Engines gilt diese Regel nicht für alle Parameter, die Sie festlegen können. RDS-Proxy verfolgt bestimmte Anweisungen und Variablen. RDS-Proxy nimmt somit kein Pinning einer Sitzung vor, wenn Sie diese ändern. In diesem Fall verwendet RDS-Proxy nur die Verbindung für andere Sitzungen erneut, die dieselben Werte für diese Einstellungen haben. Die Listen der verfolgten Anweisungen und Variablen für Aurora MySQL finden Sie unter Welche Daten RDS-Proxy für Aurora-MySQL-Datenbanken verfolgt.

Welche Daten RDS-Proxy für Aurora-MySQL-Datenbanken verfolgt

RDS-Proxy verfolgt die folgenden MySQL-Anweisungen:

- DROP DATABASE
- DROP SCHEMA
- USE

RDS-Proxy verfolgt die folgenden MySQL-Variablen:

- `AUTOCOMMIT`
- `AUTO_INCREMENT_INCREMENT`
- `CHARACTER SET (or CHAR SET)`

- CHARACTER_SET_CLIENT
- CHARACTER_SET_DATABASE
- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE
- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

> **Note**
>
> RDS-Proxy verfolgt Änderungen an den Variablen TRANSACTION_ISOLATION und TRANSACTION_READ_ONLY, wenn Sie sie im Sitzungsbereich festlegen. Wenn Sie sie jedoch für den nächsten Transaktionsbereich festlegen, heftet der RDS-Proxy Verbindungen an. Dieses Verhalten gilt unabhängig davon, ob Sie eine SET- oder eine SET TRANSACTION-Anweisung verwenden, um diese Werte zu konfigurieren.

Minimieren des Fixierens

Die Leistungsoptimierung für RDS-Proxy beinhaltet den Versuch, die Wiederverwendung von Verbindungen auf Transaktionsebene (Multiplexing) zu maximieren, indem das Fixieren minimiert wird.

Sie können das Fixieren wie folgt minimieren:

- Vermeiden Sie unnötige Datenbankanforderungen, die Anheften (Pinning) verursachen könnten.
- Legen Sie Variablen und Konfigurationseinstellungen konsistent über alle Verbindungen hinweg fest. Auf diese Weise verwenden spätere Sitzungen häufiger Verbindungen, die über diese speziellen Einstellungen verfügen.

  Wenn für PostgreSQL jedoch eine Variable festgelegt wird, wird die Sitzung durch Pinning fixiert.
- Wenden Sie bei einer MySQL-Engine-Familiendatenbank einen Sitzungs-Pinning-Filter auf den Proxy an. Sie können bestimmte Arten von Operationen vom Fixieren der Sitzung ausnehmen, wenn Sie wissen, dass dies den korrekten Betrieb Ihrer Anwendung nicht beeinträchtigt.
- Stellen Sie fest, wie häufig das Fixieren auftritt, indem Sie die Amazon-CloudWatch-Metrik `DatabaseConnectionsCurrentlySessionPinned` überwachen. Hinweise zu diesen und anderen CloudWatch-Metriken finden Sie unter Überwachen von RDS-Proxy-Metriken mit Amazon CloudWatch.
- Wenn Sie `SET`-Anweisungen verwenden, um eine identische Initialisierung für jede Clientverbindung durchzuführen, können Sie dies tun, während Sie das Multiplexing auf Transaktionsebene beibehalten. In diesem Fall verschieben Sie die Anweisungen, die den ursprünglichen Sitzungsstatus einrichten, in die Initialisierungsabfrage, die von einem Proxy verwendet wird. Diese Eigenschaft ist eine Zeichenfolge, die eine oder mehrere SQL-Anweisungen enthält, die durch Semikola getrennt sind.

  Beispielsweise können Sie eine Initialisierungsabfrage für einen Proxy definieren, der bestimmte Konfigurationsparameter festlegt. RDS-Proxy wendet dann diese Einstellungen an, wenn eine neue Verbindung für diesen Proxy eingerichtet wird. Sie können die entsprechenden `SET`-Anweisungen aus Ihrem Anwendungscode entfernen, damit sie das Multiplexing auf Transaktionsebene nicht beeinträchtigen.

  Metriken zur Häufigkeit des Pinnings für einen Proxy finden Sie unter Überwachen von RDS-Proxy-Metriken mit Amazon CloudWatch.

Bedingungen, die für alle Engine-Familien zum Pinning führen

Der Proxy fixiert die Sitzung an der aktuellen Verbindung in den folgenden Situationen an, in denen Multiplexing unerwartetes Verhalten verursachen kann:

- Jede Anweisung mit einer Textgröße über 16 KB bewirkt, dass der Proxy die Sitzung fixiert.

Bedingungen, die das Fixieren für Aurora MySQL verursachen

Bei MySQL verursachen die folgenden Interaktionen ein Pinning:

- Die expliziten MySQL-Anweisungen `LOCK TABLE`, `LOCK TABLES` oder `FLUSH TABLES WITH READ LOCK` bewirken, dass der Proxy ein Pinning der Sitzung vornimmt.
- Durch Erstellen benannter Sperren mit `GET_LOCK` wird bewirkt, dass der Proxy ein Pinning der Sitzung vornimmt.
- Wenn Sie eine Benutzervariable oder eine Systemvariable festlegen (mit einigen Ausnahmen), wird die Sitzung an den Proxy angeheftet. Wenn dadurch die Wiederverwendung von Verbindungen erheblich eingeschränkt wird, können Sie `SET`-Vorgänge so konfigurieren, dass das Pinning vermieden wird. Passen Sie dazu die Eigenschaft für Sitzungs-Pinning-Filter an. Weitere Informationen erhalten Sie unter Erstellen eines Proxys für Amazon Aurora und Ändern eines RDS-Proxy.
- Beim Erstellen einer temporären Tabelle fixiert der Proxy die Sitzung. Auf diese Weise wird der Inhalt der temporären Tabelle während der gesamten Sitzung beibehalten, unabhängig von den Transaktionsgrenzen.
- Der Aufruf der Funktionen `ROW_COUNT` und `FOUND_ROWS` verursacht manchmal Pinning.

  Die genauen Umstände, unter denen diese Funktionen Pinning verursachen, können sich zwischen Aurora-MySQL-Versionen unterscheiden, die mit MySQL 5.7 kompatibel sind.
- Vorbereitete Anweisungen bewirken, dass der Proxy die Sitzung fixiert. Diese Regel bestimmt, ob die vorbereitete Anweisung SQL-Text oder das Binärprotokoll verwendet.
- RDS-Proxy pingt keine Verbindungen an, wenn Sie SET LOCAL verwenden.
- Das Aufrufen von gespeicherten Prozeduren und gespeicherten Funktionen verursacht kein Pinning. RDS-Proxy erkennt keine Änderungen des Sitzungsstatus, die aus solchen Aufrufen resultieren. Stellen Sie sicher, dass Ihre Anwendung den Sitzungsstatus in gespeicherten Routinen nicht ändert, wenn Sie darauf angewiesen sind, dass dieser Sitzungsstatus transaktionsübergreifend beibehalten wird. Beispielsweise ist RDS-Proxy derzeit nicht

mit einer gespeicherten Prozedur kompatibel, die eine temporäre Tabelle erstellt, die transaktionsübergreifend bestehen bleiben soll.

Wenn Sie über eingehende Kenntnisse über das Verhalten Ihrer Anwendung verfügen, können Sie das Pinning-Verhalten für bestimmte Anwendungsanweisungen überspringen. Dazu wählen Sie beim Erstellen des Proxys die Option Sitzungs-Pinning-Filter. Derzeit können Sie das Sitzungs-Pinning für das Festlegen von Sitzungsvariablen und Konfigurationseinstellungen deaktivieren.

Bedingungen, die das Fixieren für Aurora PostgreSQL verursachen

Für PostgreSQL verursachen die folgenden Interaktionen eine Fixierung:

- Verwenden von `SET`-Befehlen
- Verwenden der Befehle `PREPARE`, `DISCARD`, `DEALLOCATE` oder `EXECUTE` zur Verwaltung von vorbereiteten Anweisungen
- Erstellen von temporären Sequenzen, Tabellen oder Ansichten
- Deklarieren von Cursors
- Verwerfen des Sitzungsstatus
- Listening für einen Benachrichtigungskanal
- Laden eines Bibliotheksmoduls wie `auto_explain`
- Manipulieren von Sequenzen mit Funktionen wie `nextval` und `setval`
- Interagieren mit Sperren mit Funktionen wie `pg_advisory_lock` und `pg_try_advisory_lock`

> **Note**
>
> RDS-Proxy führt kein Pinning für Advisor-Sperren auf Transaktionsebene durch. Dies gilt insbesondere für `pg_advisory_xact_lock`, `pg_advisory_xact_lock_shared`, `pg_try_advisory_xact_lock` und `pg_try_advisory_xact_lock_shared`.

- Festlegen eines Parameters oder Zurücksetzen eines Parameters auf den Standardwert Insbesondere die Verwendung der Befehle `SET` und `set_config` zum Zuweisen von Standardwerten zu Sitzungsvariablen.
- Das Aufrufen von gespeicherten Prozeduren und gespeicherten Funktionen verursacht kein Pinning. RDS-Proxy erkennt keine Änderungen des Sitzungsstatus, die aus solchen Aufrufen resultieren. Stellen Sie sicher, dass Ihre Anwendung den Sitzungsstatus in gespeicherten Routinen nicht ändert, wenn Sie darauf angewiesen sind, dass dieser Sitzungsstatus

transaktionsübergreifend beibehalten wird. Beispielsweise ist RDS-Proxy derzeit nicht mit einer gespeicherten Prozedur kompatibel, die eine temporäre Tabelle erstellt, die transaktionsübergreifend bestehen bleiben soll.

- Verwerfen des Sitzungsstatus Wenn Sie Verbindungspooling-Bibliotheken verwenden, deren `DISCARD ALL`-Abfrage als Reset-Abfrage konfiguriert ist, fixiert RDS-Proxy Ihre Client-Verbindung bei der Freigabe. Dies verringert die Effizienz des Multiplexings des Proxys und kann zu unerwarteten Ergebnissen führen, da der Befehl `DISCARD ALL` die Sitzungsstatusverwaltung beeinträchtigen kann.

Löschen eines RDS-Proxy

Sie können einen Proxy löschen, wenn Sie ihn nicht mehr benötigen. Oder Sie löschen einen Proxy, wenn Sie die/den mit ihm verknüpfte DB-Instance oder -Cluster außer Betrieb nehmen.

AWS-Managementkonsole

So löschen Sie einen Proxy:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Proxies (Proxys).
3. Wählen Sie den zu löschenden Proxy aus der Liste aus.
4. Wählen Sie Delete Proxy (Proxy löschen).

AWS CLI

Um einen DB-Proxy zu löschen, verwenden Sie den AWS CLI-Befehl delete-db-proxy. Um zugehörige Zuordnungen zu entfernen, verwenden Sie auch den Befehl deregister-db-proxy-targets .

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

RDS-API

Um einen DB-Proxy zu löschen, rufen Sie die Amazon-RDS-API-Funktion DeleteDBProxy auf. Um verwandte Elemente und Zuordnungen zu löschen, rufen Sie auch die Funktionen DeleteDBProxyTargetGroup und DeregisterDBProxyTargets auf.

Arbeiten mit Amazon-RDS-Proxy-Endpunkten

RDS-Proxy-Endpunkte bieten flexible und effiziente Möglichkeiten zur Verwaltung von Datenbankverbindungen, wodurch Skalierbarkeit, Verfügbarkeit und Sicherheit verbessert werden. Mit Proxy-Endpunkten können Sie:

- Überwachung und Fehlerbehebung vereinfachen – Verwenden Sie mehrere Endpunkte, um Verbindungen von verschiedenen Anwendungen unabhängig voneinander nachzuverfolgen und zu verwalten.
- Leseskalierbarkeit verbessern – Nutzen Sie Leser-Endpunkte mit Aurora-DB-Clustern, um den Leseverkehr effizient zu verteilen und so die Leistung für abfrageintensive Workloads zu optimieren.
- VPC-übergreifenden Zugriff ermöglichen — Connect Datenbanken VPCs mithilfe von VPC-übergreifenden Endpunkten miteinander, sodass Ressourcen wie EC2 Amazon-Instances in einer VPC auf Datenbanken in einer anderen zugreifen können.

Themen

- Überblick über Proxy-Endpunkte
- Limits für Proxy-Endpunkte
- Verwenden von Reader-Endpunkten mit Aurora-Clustern
- Zugriff auf Aurora über VPCs
- Erstellen eines Proxy-Endpunktes
- Anzeigen von Proxy-Endpunkten
- Ändern eines Proxy-Endpunkts
- Löschen eines Proxy-Endpunkts

Überblick über Proxy-Endpunkte

Arbeiten mit RDS-Proxy-Endpunkten umfassen die gleichen Arten von Verfahren wie bei Aurora-DB-Cluster- und Reader-Endpunkten. Wenn Sie nicht mit Aurora-Endpunkten vertraut sind finden Sie weitere Informationen unter Amazon-Aurora-Endpunktverbindungen.

Wenn Sie RDS-Proxy mit einem Aurora-Cluster verwenden, unterstützt der Standardendpunkt sowohl Lese- als auch Schreibvorgänge. Das bedeutet, dass er alle Anforderungen an die Writer-Instance des Clusters weiterleitet und so zu deren `max_connections`-Limit beiträgt. Um den Datenverkehr besser zu verteilen, können Sie zusätzliche read/write oder schreibgeschützte Endpunkte für Ihren Proxy einrichten, was ein effizienteres Workload-Management und eine verbesserte Skalierbarkeit ermöglicht.

Verwenden Sie zum Verarbeiten von Leseabfragen einen schreibgeschützten Endpunkt mit Ihrem Proxy, genauso wie Sie bei einem Reader-Endpunkt für einen von Aurora bereitgestellten Cluster vorgehen würden. Dieser Ansatz maximiert die Leseskalierbarkeit von Aurora, indem Abfragen auf eine oder mehrere Reader-DB-Instances verteilt werden. Indem Sie einen schreibgeschützten Endpunkt verwenden und nach Bedarf mehr Reader-Instances hinzufügen, können Sie die Anzahl gleichzeitiger Abfragen und Verbindungen erhöhen, die Ihr Cluster verarbeiten kann.

> **Tip**
>
> Wenn Sie mit dem einen Proxy für einen Aurora-Cluster erstellen, können Sie festlegen AWS-Managementkonsole, dass RDS Proxy automatisch einen Reader-Endpunkt erstellt. Informationen zu den Vorteilen eines Reader-Endpunkts finden Sie unter Verwenden von Reader-Endpunkten mit Aurora-Clustern.

Wenn Sie einen Proxy-Endpunkt erstellen, den Sie ihn einer anderen Virtual Private Cloud (VPC) als der des Proxys zuordnen. Auf diese Weise können Sie über eine andere VPC eine Verbindung zum Proxy herstellen, beispielsweise über eine VPC, die von einer anderen Anwendung innerhalb Ihres Unternehmens verwendet wird.

Informationen zu Limits im Zusammenhang mit Proxy-Endpunkten finden Sie unter Limits für Proxy-Endpunkte.

In den RDS-Proxy-Protokollen wird jedem Eintrag der Name des zugehörigen Proxy-Endpunkts vorangestellt. Dies kann entweder der Name sein, den Sie für einen benutzerdefinierten Endpunkt

angegeben haben, oder der spezielle Name `default` für den read/write Standardendpunkt des Proxys.

Jeder Proxy-Endpunkt hat seinen eigenen Satz von CloudWatch Metriken. Überwachen Sie Metriken für alle Proxy-Endpunkte, einen bestimmten Endpunkt oder alle read/write oder nur lesbare Endpunkte eines Proxys. Weitere Informationen finden Sie unter Überwachen von RDS-Proxy-Metriken mit Amazon CloudWatch.

Ein Proxy-Endpunkt verwendet denselben Authentifizierungsmechanismus wie der zugehörige Proxy. Für RDS-Proxy richtet automatisch Berechtigungen und Autorisierungen für den benutzerdefinierten Endpunkt ein, die mit den Eigenschaften des zugehörigen Proxys übereinstimmen.

Informationen zur Funktionsweise von Proxy-Endpunkten für DB-Cluster in einer globalen Aurora-Datenbank finden Sie unter So funktionieren RDS-Proxy-Endpunkte mit globalen Datenbanken.

Limits für Proxy-Endpunkte

Für RDS-Proxy-Endpunkte gelten die folgenden Beschränkungen:

- Der RDS-Proxy-Standardendpunkt kann nicht geändert werden.
- Die maximale Anzahl von benutzerdefinierten Endpunkten für einen Proxy beträgt 20. Daher kann ein Proxy bis zu 21 Endpunkte haben: den Standard-Endpunkt plus 20, die Sie erstellen.
- Wenn Sie zusätzliche Endpunkte mit einem Proxy verknüpfen, bestimmt RDS-Proxy automatisch, welche DB-Instances in Ihrem Cluster für jeden Endpunkt verwendet werden. Sie können bestimmte Instances nicht so auswählen, wie Sie es mit benutzerdefinierten Aurora-Endpunkten können.
- Für IPv6 oder Dual-Stack-Endpunkt-Netzwerktypen müssen Ihre VPC und Subnetze so konfiguriert sein, dass sie den ausgewählten Netzwerktyp unterstützen.

Wenn Sie einen Proxy erstellen, erstellt RDS automatisch einen VPC-Endpunkt für die sichere Kommunikation zwischen Anwendungen und der Datenbank. Der VPC-Endpunkt ist sichtbar und kann über die Amazon-VPC-Konsole aufgerufen werden.

Durch das Hinzufügen eines neuen Proxy-Endpunkts wird ein AWS PrivateLink Schnittstellenendpunkt bereitgestellt. Wenn Sie Ihrem Proxy einen oder mehrere Endpunkte hinzufügen, fallen zusätzliche Gebühren an. Weitere Informationen finden Sie unter Preise von RDS-Proxy.

Verwenden von Reader-Endpunkten mit Aurora-Clustern

Sie können schreibgeschützte Endpunkte namens Reader-Endpunkte erstellen und eine Verbindung herstellen, wenn Sie RDS-Proxy mit Aurora-Clustern verwenden. Diese Reader-Endpunkte tragen dazu bei, die Leseskalierbarkeit Ihrer abfrageintensiven Anwendungen zu verbessern. Reader-Endpunkte helfen auch, die Verfügbarkeit Ihrer Verbindungen zu verbessern, wenn eine Reader-DB-Instance in Ihrem Cluster nicht verfügbar ist.

> **Note**
>
> Wenn Sie angeben, dass ein neuer Endpunkt schreibgeschützt ist, verlangt RDS-Proxy, dass der Aurora-Cluster eine oder mehrere Reader-DB-Instances hat. In einigen Fällen ändern Sie ggf. das Ziel des Proxys in einen Aurora-Cluster, der nur einen Writer enthält. In diesem Fall schlagen alle Anfragen an den Reader-Endpunkt mit einem Fehler fehl. Anfragen schlagen auch fehl, wenn das Ziel des Proxys eine RDS-Instance anstelle eines Aurora-Clusters ist. Wenn ein Aurora-Cluster Reader-Instances hat, aber diese Instances nicht verfügbar sind, wartet RDS-Proxy darauf, die Anfrage zu senden, anstatt sofort einen Fehler zu zurückzugeben. Wenn innerhalb des Zeitraums der Zeitüberschreitung für die Verbindung keine Reader-Instance verfügbar wird, schlägt die Anfrage mit einem Fehler fehl.

Wie Reader-Endpunkte die Verfügbarkeit von Anwendungen unterstützen

In einigen Fällen sind möglicherweise eine oder mehrere Reader-Instances in Ihrem Cluster nicht verfügbar. In diesem Fall können Verbindungen, die einen Reader-Endpunkt eines DB-Proxys verwenden, schneller wiederhergestellt werden als solche, die den Aurora-Reader-Endpunkt verwenden. RDS-Proxy leitet Verbindungen nur an die verfügbaren Reader-Instances im Cluster weiter. Es gibt keine Verzögerung aufgrund von DNS-Caching, wenn eine Instance nicht verfügbar ist.

Wenn die Verbindung Multiplexing durchführt, leitet RDS-Proxy nachfolgende Abfragen ohne Unterbrechung Ihrer Anwendung an eine andere Reader-DB-Instance weiter. Bei der automatischen Umstellung auf eine neue Reader-Instance prüft RDS-Proxy die Replikationsverzögerung der alten und neuen Reader-Instance. RDS-Proxy stellt sicher, dass die neue Reader-Instance mit den gleichen Änderungen wie die vorherige Reader-Instance auf dem neuesten Stand ist. Auf diese Weise hat Ihre Anwendung niemals veraltete Daten, wenn RDS-Proxy von einer Reader-DB-Instance zu einer anderen wechselt.

Wenn die Verbindung fixiert ist, gibt die nächste Abfrage der Verbindung einen Fehler zurück. Ihre Anwendung kann sich jedoch sofort wieder mit demselben Endpunkt verbinden. RDS-Proxy leitet die Verbindung zu einer anderen Reader-DB-Instance weiter, die sich im Status `available` befindet. Wenn Sie die Verbindung manuell wiederherstellen, überprüft RDS-Proxy nicht die Replikationsverzögerung zwischen den alten und neuen Reader-Instances.

Wenn Ihr Aurora-Cluster keine Reader-Instances verfügbar hat, prüft RDS-Proxy, ob diese Bedingung vorübergehend oder dauerhaft ist. Das Verhalten ist in jedem Fall wie folgt:

- Angenommen, Ihr Cluster hat eine oder mehrere Reader-DB-Instances, aber keine von ihnen befindet sich im Status `Available`. Zum Beispiel könnten alle Reader-Instances neu gestartet werden oder Probleme aufweisen. In diesem Fall warten Versuche, eine Verbindung zu einem Reader-Endpunkt herzustellen, darauf, dass eine Reader-Instance verfügbar wird. Wenn innerhalb des Zeitraums der Zeitüberschreitung für die Verbindung keine Reader-Instance verfügbar wird, schlägt der Verbindungsversuch fehl. Wenn eine Reader-Instance verfügbar wird, ist der Verbindungsversuch erfolgreich.
- Angenommen, Ihr Cluster hat keine Reader-DB-Instances. In diesem Fall gibt RDS-Proxy sofort einen Fehler zurück, wenn Sie versuchen, eine Verbindung zu einem Reader-Endpunkt herzustellen. Um dieses Problem zu beheben, fügen Sie Ihrem Cluster eine oder mehrere Reader-Instances hinzu, bevor Sie eine Verbindung zum Reader-Endpunkt herstellen.

Wie Reader-Endpunkte bei der Skalierbarkeit von Abfragen unterstützen

Reader-Endpunkte für einen Proxy unterstützen mit einer Aurora-Abfrage die Skalierbarkeit auf folgende Weise:

- Wenn Sie Reader-Instances zu Ihrem Aurora-Cluster hinzufügen, kann RDS-Proxy neue Verbindungen zu beliebigen Reader-Endpunkten an die verschiedenen Reader-Instances weiterleiten. Auf diese Weise verlangsamen Abfragen, die mit einer Reader-Endpunktverbindung durchgeführt werden, keine Abfragen, die über eine andere Reader-Endpunktverbindung durchgeführt werden. Die Abfragen laufen auf separaten DB-Instances. Jede DB-Instance verfügt über eigene Rechenressourcen, Puffer-Cache usw.
- Wo praktisch, verwendet RDS-Proxy dieselbe Reader-DB-Instance für alle Abfragen unter Verwendung einer bestimmten Reader-Endpunktverbindung. Auf diese Weise kann eine Reihe von verwandten Abfragen in denselben Tabellen das Caching, die Planoptimierung usw. für eine bestimmte DB-Instance nutzen.

- Wenn eine Reader-DB-Instance nicht verfügbar ist, hängt die Auswirkung auf Ihre Anwendung davon ab, ob die Sitzung Multiplexing durchführt oder fixiert ist. Wenn die Sitzung Multiplexing durchführt, leitet RDS-Proxy leitet alle nachfolgenden Abfragen an eine andere Reader-DB-Instance weiter, ohne dass Sie etwas unternehmen müssen. Wenn die Sitzung fixiert ist, bekommt Ihre Anwendung einen Fehler und muss sich erneut verbinden. Sie können sich sofort wieder mit dem Reader-Endpunkt verbinden und RDS-Proxy leitet die Verbindung zu einer verfügbaren Reader-DB-Instance. Weitere Informationen zum Multiplexing und Pinning für Proxy-Sitzungen finden Sie unter Überblick über RDS-Proxy-Konzepte.
- Je mehr Reader-DB-Instances Sie in dem Cluster haben, desto mehr gleichzeitige Verbindungen können Sie mithilfe von Reader-Endpunkten herstellen. Angenommen, Ihr Cluster verfügt über vier Reader-DB-Instances, die jeweils so konfiguriert sind, dass sie 200 gleichzeitige Verbindungen unterstützen. Nehmen wir außerdem an, Ihr Proxy ist so konfiguriert, dass er 50 % der maximalen Verbindungen verwendet. Hier beträgt die maximale Anzahl von Verbindungen, die Sie über die Reader-Endpunkte im Proxy herstellen können, 100 (50% von 200) für Reader 1. Es sind auch 100 für Reader 2 usw. bei insgesamt 400. Wenn Sie die Anzahl der Reader-DB-Instances im Cluster auf acht verdoppeln, verdoppelt sich die maximale Anzahl von Verbindungen über die Reader-Endpunkte ebenfalls, und zwar auf 800.

Beispiele für die Verwendung von Reader-Endpunkten

Das folgende Linux-Beispiel zeigt, wie Sie bestätigen können, dass Sie mit einem Aurora MySQL-Cluster durch einen Reader-Endpunkt verbunden sind. Die `innodb_read_only`-Konfigurationseinstellung ist aktiviert. Versuche Schreibvorgänge wie `CREATE DATABASE`-Anweisungen durchzuführen, schlagen mit einem Fehler fehl. Und Sie können bestätigen, dass Sie mit einer Reader-DB-Instance verbunden sind, indem Sie den Namen der DB-Instance mit der `aurora_server_id`-Variable prüfen.

> **Tip**
>
> Verlassen Sie sich nicht nur darauf, den DB-Instance-Namen zu überprüfen, um festzustellen, ob die Verbindung schreibgeschützt oder schreibgeschützt ist. read/write Denken Sie daran, dass DB-Instances in Aurora-Clustern die Rollen zwischen Writer und Reader ändern können, wenn Failovers auftreten.

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u
 admin -p
```

```
...
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it
 cannot execute this statement

mysql> select @@aurora_server_id;
+--------------------------------------+
| @@aurora_server_id                   |
+--------------------------------------+
| proxy-reader-endpoint-demo-instance-3 |
+--------------------------------------+
```

Das folgende Beispiel zeigt, wie Ihre Verbindung zu einem Proxy-Reader-Endpunkt auch dann weiterfunktioniert, wenn die Reader-DB-Instance gelöscht wird. In diesem Beispiel hat der Aurora-Cluster zwei Reader-Instances, `instance-5507` und `instance-7448`. Die Verbindung zum Reader-Endpunkt beginnt mit der Verwendung einer der Reader-Instances. Im Beispiel wird diese Reader-Instance durch einen Befehl `delete-db-instance` gelöscht. Für nachfolgende Abfragen wechselt RDS-Proxy zu einer anderen Reader-Instance.

```
$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
  -u my_user -p
...
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-5507      |
+--------------------+

mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

```
mysql> select count(*) from information_schema.tables;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

Während die `mysql`-Sitzung noch ausgeführt wird, löscht der folgende Befehl die Reader-Instance, mit der der Reader-Endpunkt verbunden ist.

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

Abfragen in der `mysql`-Sitzung funktionieren weiterhin, ohne dass eine erneute Verbindung erforderlich ist. RDS-Proxy wechselt automatisch zu einer anderen Reader-DB-Instance.

```
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-7448      |
+--------------------+

mysql> select count(*) from information_schema.TABLES;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

Zugriff auf Aurora über VPCs

Standardmäßig befinden sich die Komponenten Ihres Aurora--Technologie-Stacks alle in derselben Amazon-VPC. Nehmen wir zum Beispiel an, dass eine Anwendung, die auf einer EC2 Amazon-Instance ausgeführt wird, eine Verbindung zu einer im Aurora-DB-Cluster herstellt. In diesem Fall müssen sich der Anwendungsserver und die Datenbank beide innerhalb derselben VPC befinden.

Mit RDS Proxy können Sie den Zugriff auf eine eines Aurora-DB-Clusters in einer VPC von Ressourcen in einer anderen VPC aus einrichten, z. B. EC2 Instances. Zum Beispiel könnte Ihre Organisation mehrere Anwendungen haben, die auf dieselben Datenbankressourcen zugreifen. Jede Anwendung könnte sich in einer eigenen VPC befinden.

Um den VPC-übergreifenden Zugriff zu aktivieren, erstellen Sie einen neuen Endpunkt für den Proxy. Der Proxy selbst befindet sich in derselben VPC wie der Aurora-DB-Cluster. Der VPC-übergreifende Endpunkt befindet sich jedoch zusammen mit den anderen Ressourcen wie den Instances in der anderen VPC. EC2 Der VPC-übergreifende Endpunkt ist Subnetzen und Sicherheitsgruppen aus derselben VPC wie die EC2 und anderen Ressourcen zugeordnet. Mit diesen Zuordnungen können Sie von den Anwendungen aus eine Verbindung zum Endpunkt herstellen, die aufgrund der VPC-Einschränkungen sonst nicht auf die Datenbank zugreifen können.

In den folgenden Schritten wird erläutert, wie Sie einen VPC-übergreifenden Endpunkt erstellen und darauf über RDS-Proxy zugreifen:

1. Erstellen Sie zwei VPCs oder wählen Sie zwei aus VPCs , die Sie bereits für die Arbeit mit Aurora verwenden. Jede VPC sollte über eigene zugehörige Netzwerkressourcen wie ein Internet-Gateway, Routing-Tabellen, Subnetze und Sicherheitsgruppen verfügen. Wenn Sie nur eine VPC haben, können Sie Erste Schritte mit Amazon Aurora für die Schritte zum Einrichten einer anderen VPC zur erfolgreichen Verwendung von Aurora konsultieren. . Sie können auch Ihre bestehende VPC in der EC2 Amazon-Konsole überprüfen, um zu sehen, welche Arten von Ressourcen miteinander verbunden werden sollen.
2. Erstellen Sie einen DB-Proxy, der mit dem Aurora-DB-Cluster verknüpft ist, mit dem/der Sie eine Verbindung herstellen möchten. Folgen Sie dem Verfahren unter Erstellen eines Proxys für Amazon Aurora.
3. Auf der Seite Einzelheiten für Ihren Proxy in der RDS-Konsole unter dem Abschnitt Proxy-Endpunkte wählen Sie Endpunkt erstellen. Folgen Sie dem Verfahren unter Erstellen eines Proxy-Endpunktes.
4. Wählen Sie aus, ob der VPC-übergreifende Endpunkt read/write oder schreibgeschützt werden soll.
5. Anstatt den Standard der gleichen VPC wie für den Aurora-DB-Cluster zu akzeptieren, wählen Sie eine andere VPC. Diese VPC muss sich in derselben AWS Region befinden wie die VPC, in der sich der Proxy befindet.
6. Anstatt nun die Standardeinstellungen für Subnetze und Sicherheitsgruppen von derselben VPC wie für den Aurora-DB-Cluster zu akzeptieren, treffen Sie eine neue Auswahl. Wählen Sie diese basierend auf den Subnetzen und Sicherheitsgruppen aus der von Ihnen ausgewählten VPC.
7. Sie müssen keine der Einstellungen für die Secrets Manager-Secrets ändern. Dieselben Anmeldeinformationen funktionieren für alle Endpunkte für Ihren Proxy, unabhängig davon, in welcher VPC sich jeder Endpunkt befindet. In ähnlicher Weise funktionieren Ihre IAM-Konfiguration und Ihre Berechtigungen bei Verwendung der IAM-Authentifizierung auf allen Proxy-Endpunkten

einheitlich, auch wenn sich die Endpunkte an unterschiedlichen Standorten befinden. VPCs Pro Endpunkt ist keine zusätzliche IAM-Konfiguration erforderlich.

8. Warten Sie, bis der neue Endpunkt den Status verfügbar erreicht.
9. Notieren Sie sich den vollständigen Endpunktnamen. Dies ist der Wert, der auf *Region_name*`.rds.amazonaws.com` endet, den Sie als Teil der Verbindungszeichenfolge für Ihre Datenbankanwendung angeben.
10. Greifen Sie auf den neuen Endpunkt von einer Ressource in derselben VPC wie der Endpunkt zu. Eine einfache Möglichkeit, diesen Prozess zu testen, besteht darin, eine neue EC2 Instanz in dieser VPC zu erstellen. Melden Sie sich dann bei der EC2 Instanz an und führen Sie die `psql` Befehle `mysql` oder aus, um mithilfe des Endpunktwerts in Ihrer Verbindungszeichenfolge eine Verbindung herzustellen.

Erstellen eines Proxy-Endpunktes

Befolgen Sie diese Anleitungen zum Erstellen eines Proxy-Endpunkts:

Konsole

So erstellen Sie einen Proxy-Endpunkt

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Proxies (Proxys).
3. Klicken Sie auf den Namen des Proxys, für den Sie einen neuen Endpunkt erstellen möchten.

   Die Detailseite für diesen Proxy wird angezeigt.
4. Im Abschnitt Proxy-Endpunkte wählen Sie Proxy-Endpunkt erstellen.

   Das Fenster Proxy-Endpunkt erstellen wird angezeigt.
5. Für Name des Proxy-Endpunkts geben Sie einen beschreibenden Namen Ihrer Wahl ein.
6. Wählen Sie für die Target-Rolle aus, ob der Endpunkt schreibgeschützt read/write oder schreibgeschützt werden soll.

   Verbindungen, die read/write Endpunkte verwenden, können alle Arten von Vorgängen ausführen, z. B. DDL-Anweisungen (Data Definition Language), DML-Anweisungen (Data Manipulation Language) und Abfragen. Diese Endpunkte stellen immer eine Verbindung mit der primären Instance des Aurora-Clusters her. Sie können read/write Endpunkte für

allgemeine Datenbankoperationen verwenden, wenn Sie in Ihrer Anwendung nur einen einzigen Endpunkt verwenden. Sie können read/write Endpunkte auch für Verwaltungsvorgänge, OLTP-Anwendungen (Online Transaction Processing) und extract-transform-load (ETL) -Jobs verwenden.

Verbindungen, die einen schreibgeschützten Endpunkt verwenden, können nur Abfragen durchführen. Wenn es mehrere Reader-Instances im Aurora-Cluster gibt, kann RDS-Proxy für jede Verbindung mit dem Endpunkt eine andere Reader-Instance verwenden. Auf diese Weise kann eine abfrageintensive Anwendung die Clustering-Fähigkeit von Aurora nutzen. Sie können dem Cluster mehr Abfragekapazität hinzufügen, indem Sie weitere Reader-DB-Instances hinzufügen. Diese schreibgeschützten Verbindungen verursachen keinen Overhead für die primäre Instance des Clusters. Auf diese Weise verlangsamen Ihre Berichts- und Analyseabfragen die Schreibvorgänge Ihrer OLTP-Anwendungen nicht.

7. Wählen Sie für Virtual Private Cloud (VPC) die Standardeinstellung für den Zugriff auf den Endpunkt von denselben EC2 Instanzen oder anderen Ressourcen aus, die normalerweise für den Zugriff auf den Proxy oder die zugehörige Datenbank verwendet werden. Wenn Sie den VPC-übergreifenden Zugriff für diesen Proxy einrichten möchten, wählen Sie eine andere VPC als die Standard-VPC aus. Weitere Informationen zum VPC-übergreifenden Zugriff finden Sie unter Zugriff auf Aurora über VPCs.
8. Wählen Sie für den Netzwerktyp des Endpunkts die IP-Version für den Proxy-Endpunkt aus. Die folgenden Optionen sind verfügbar:

   - IPv4— Der Proxy-Endpunkt verwendet nur IPv4 Adressen (Standard).
   - IPv6— Der Proxy-Endpunkt verwendet nur IPv6 Adressen.
   - Dual-Stack — Der Proxy-Endpunkt unterstützt IPv4 sowohl IPv6 Adressen als auch.

   Um Dual-Stack verwenden zu können IPv6 , müssen Ihre VPC und Subnetze so konfiguriert sein, dass sie den ausgewählten Netzwerktyp unterstützen.
9. Für Subnetze füllt RDS-Proxy standardmäßig dieselben Subnetze wie der zugehörige Proxy aus. Wenn Sie den Zugriff auf den Endpunkt einschränken möchten, damit nur ein Teil des Adressbereichs der VPC eine Verbindung herstellen kann, entfernen Sie ein oder mehrere Subnetze.
10. Für die VPC-Sicherheitsgruppe können Sie eine vorhandene Sicherheitsgruppe auswählen oder eine neue erstellen. Standardmäßig füllt RDS-Proxy dieselbe Sicherheitsgruppe oder Gruppen wie der zugehörige Proxy aus. Wenn die eingehenden und ausgehenden Regeln für den Proxy für diesen Endpunkt geeignet sind, behalten Sie die Standardauswahl bei.

Wenn Sie eine neue Sicherheitsgruppe erstellen möchten, geben Sie auf dieser Seite einen Namen für die Sicherheitsgruppe an. Bearbeiten Sie dann später die Sicherheitsgruppeneinstellungen von der EC2 Konsole aus.

11. Wählen Sie Proxy-Endpunkt erstellen.

AWS CLI

Verwenden Sie den AWS CLI create-db-proxy-endpointBefehl, um einen Proxy-Endpunkt zu erstellen.

Nutzen Sie die folgenden erforderlichen Parameter:

- `--db-proxy-name` *`value`*
- `--db-proxy-endpoint-name` *`value`*
- `--vpc-subnet-ids` *`list_of_ids`*. Trennen Sie das Subnetz IDs durch Leerzeichen. Sie geben die ID der VPC selbst nicht an.

Sie können auch die folgenden optionalen Parameter angeben:

- `--target-role { READ_WRITE | READ_ONLY }`. Dieser Parameter lautet standardmäßig `READ_WRITE`. Der Wert `READ_ONLY` hat nur Auswirkungen auf von Aurora bereitgestellte Cluster, die eine oder mehrere Reader-DB-Instances enthalten. Wenn der Proxy einem Aurora-Cluster zugeordnet ist, der nur eine Writer-DB-Instance enthält, können Sie `READ_ONLY` nicht angeben. Weitere Informationen zur beabsichtigten Verwendung von schreibgeschützten Endpunkten mit Aurora-Clustern finden Sie unter Verwenden von Reader-Endpunkten mit Aurora-Clustern .
- `--vpc-security-group-ids` *`value`*. Trennen Sie die Sicherheitsgruppe IDs durch Leerzeichen. Wenn Sie diesen Parameter weglassen, verwendet RDS-Proxy die Standardsicherheitsgruppe für die VPC. RDS Proxy bestimmt die VPC auf der Grundlage des Subnetzes IDs , das Sie für den `--vpc-subnet-ids` Parameter angeben.
- `--endpoint-network-type { IPV4 | IPV6 | DUAL }`. Dieser Parameter gibt die IP-Version für den Proxy-Endpunkt an. Der Standardwert ist `IPV4`. Um `IPV6` oder `DUAL` zu verwenden, müssen Ihre VPC und Subnetze so konfiguriert sein, dass sie den ausgewählten Netzwerktyp unterstützen.

Example

Im folgenden Beispiel wird ein Proxy-Endpunkt mit dem Namen erstell `my-endpoint`.

Für Linux, macOS oder Unix:

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id \
  --endpoint-network-type DUAL
```

Für Windows:

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id ^
  --endpoint-network-type DUAL
```

RDS-API

Verwenden Sie die Aktion „Endpunkt erstellen“ der RDS-API, um einen DBProxyProxy-Endpunkt zu erstellen.

Anzeigen von Proxy-Endpunkten

Gehen Sie wie folgt vor, um vorhandene Proxy-Endpunkte anzuzeigen:

Konsole

Anzeigen der Details für einen Proxy-Endpunkt

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Proxies (Proxys).
3. Wählen Sie in der Liste den Proxy aus, dessen Endpunkt Sie anzeigen möchten. Klicken Sie auf den Proxy-Namen, um die Detailseite anzuzeigen.

4. Wählen Sie im Abschnitt Proxy-Endpunkte den Endpunkt aus, den Sie anzeigen möchten. Klicken Sie auf seinen Namen, um die Detailseite aufzurufen.
5. Untersuchen Sie die Parameter, für deren Werten Sie sich interessieren. Sie können Eigenschaften wie die Folgenden überprüfen:
   - Ob der Endpunkt lese-/schreibgeschützt oder schreibgeschützt ist.
   - Die Endpunkt-Adresse, die Sie in einer Datenbank-Verbindungszeichenfolge verwenden.
   - Die VPC-Subnetze und -Sicherheitsgruppen, die einem Endpunkt zugeordnet sind.

AWS CLI

Um einen oder mehrere Proxy-Endpunkte anzuzeigen, verwenden Sie den AWS CLI-Befehl describe-db-proxy-endpoints.

Sie können die folgenden optionalen Parameter angeben:

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

Das folgende Beispiel beschreibt den `my-endpoint`-Proxy-Endpunkt.

Example

Für Linux, macOS oder Unix:

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

Für Windows:

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

RDS-API

Um einen oder mehrere Proxy-Endpunkte zu beschreiben, verwenden Sie die Aktion RDS-API DescribeDBProxyEndpoints.

Ändern eines Proxy-Endpunkts

Gehen Sie wie folgt vor, um die Proxy-Endpunkte zu ändern:

Konsole

So ändern Sie einen oder mehrere Proxy-Endpunkte

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Proxies (Proxys).
3. Wählen Sie in der Liste den Proxy aus, dessen Endpunkt Sie ändern möchten. Klicken Sie auf den Proxy-Namen, um die Detailseite anzuzeigen.
4. Wählen Sie m Abschnitt Proxy-Endpunkte den Endpunkt aus, den Sie ändern möchten. Sie können ihn in der Liste auswählen oder auf seinen Namen klicken, um die Detailseite anzuzeigen.
5. Auf der Seite mit den Proxy-Details unter dem Abschnitt Proxy-Endpunkte wählen Sie Bearbeiten. Oder wählen Sie auf der Detailseite des Proxy-Endpunkts für Aktionen Bearbeiten.
6. Ändern Sie wie gewünscht die Werte der Parameter.
7. Wählen Sie Änderungen speichern aus.

AWS CLI

Um einen DB-Proxy-Endpunkt zu ändern, verwenden Sie den AWS CLI-Befehl modify-db-proxy-endpoint mit den folgenden erforderlichen Parametern:

- `--db-proxy-endpoint-name`

Geben Sie Änderungen in den Endpunkteigenschaften an, indem Sie einen oder mehrere der folgenden Parameter verwenden:

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`. Trennen Sie die Sicherheitsgruppen-IDs durch Leerzeichen

Das folgende Beispiel benennt den `my-endpoint`-Proxy-Endpunkt in `new-endpoint-name`.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
  --new-db-proxy-endpoint-name new-endpoint-name
```

Für Windows:

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

RDS-API

Um einen Proxy-Endpunkt zu ändern, verwenden Sie die Aktion RDS-API ModifyDBProxyEndpoint.

Löschen eines Proxy-Endpunkts

Gehen Sie folgendermaßen vor, um einen Endpunkt für Ihren Proxy zu löschen:

> **Note**
>
> Sie können den Standardendpunkt des Proxy nicht löschen, den RDS-Proxy automatisch für jeden Proxy erstellt.
> Wenn Sie einen Proxy löschen, löscht RDS-Proxy automatisch alle zugehörigen Endpunkte.

Konsole

So löschen Sie einen Proxy-Endpunkt mit AWS-Managementkonsole

1. Wählen Sie im Navigationsbereich Proxies (Proxys).
2. Wählen Sie in der Liste den Proxy aus, dessen Endpunkt Sie löschen möchten. Klicken Sie auf den Proxy-Namen, um die Detailseite anzuzeigen.
3. Im Abschnitt Proxy-Endpunkte den Endpunkt aus, den Sie löschen möchten. Sie können einen oder mehrere Endpunkte in der Liste auswählen oder auf den Namen eines einzelnen Endpunkts klicken, um die Detailseite anzuzeigen.

4. Auf der Seite mit den Proxy-Details unter dem Abschnitt Proxy-Endpunkte, wählen Sie Löschen. Oder wählen Sie auf der Detailseite des Proxy-Endpunkts für Aktionen die Option Löschen aus.

AWS CLI

Um einen Proxy-Endpunkt zu löschen, führen Sie den Befehl delete-db-proxy-endpoint mit den folgenden erforderlichen Parametern:

- `--db-proxy-endpoint-name`

Der folgende Befehl löscht den Proxy-Endpunkt mit dem Namen `my-endpoint`.

Für Linux, macOS oder Unix:

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

Für Windows:

```
aws rds delete-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint
```

RDS-API

Um einen Proxy-Endpunkt mit der RDS-API zu löschen, führen Sie die Aktion DeleteDBProxyEndpoint aus. Geben Sie den Namen des Proxy-Endpunkts für den `DBProxyEndpointName`-Parameter an.

Überwachen von RDS-Proxy-Metriken mit Amazon CloudWatch

Sie können den RDS-Proxy mithilfe von Amazon CloudWatch überwachen. CloudWatch sammelt und verarbeitet Rohdaten von den Proxys zu lesbaren Metriken nahezu in Echtzeit. Um diese Metriken in der CloudWatch-Konsole zu finden, wählen Sie Metrics (Metriken), dann RDS, und dann Per-Proxy Metrics (Proxy-Metriken). Weitere Informationen finden Sie unter Verwenden von Amazon CloudWatch-Metriken im Amazon CloudWatch-Benutzerhandbuch.

> **Note**
>
> RDS veröffentlicht diese Metriken für jede zugrunde liegende Amazon EC2-Instance, die dem Proxy zugeordnet ist. Ein einzelner Proxy kann von mehr als einer EC2-Instance bedient werden. Verwenden Sie CloudWatch-Statistiken, um die Werte für einen Proxy über alle zugeordneten Instances hinweg zu aggregieren.
> Einige dieser Metriken sind möglicherweise erst nach der ersten erfolgreichen Verbindung durch einen Proxy sichtbar.

In den RDS-Proxy-Protokollen wird jedem Eintrag der Name des zugehörigen Proxy-Endpunkts vorangestellt. Dieser Name kann der Name sein, den Sie für einen benutzerdefinierten Endpunkt angegeben haben, oder der spezielle Name `default` für den Standardendpunkt eines Proxys, der Lese-/Schreibanforderungen durchführt.

Alle RDS-Proxy-Metriken befinden sich in der Gruppe `proxy`.

Jeder Proxy-Endpunkt hat seinen eigenen CloudWatch-Metriken. Sie können die Nutzung jedes Proxy-Endpunkts unabhängig überwachen. Weitere Informationen zu den Proxy-Endpunkten finden Sie unter Arbeiten mit Amazon-RDS-Proxy-Endpunkten.

Sie können die Werte für jede Metrik mit einem der folgenden Dimensionssätze aggregieren. Zum Beispiel können Sie, indem Sie den `ProxyName`-Dimensionssatz verwenden, den gesamten Datenverkehr für einen bestimmten Proxy analysieren. Durch die Verwendung der anderen Dimensionssätze können Sie die Metriken auf verschiedene Arten aufteilen. Sie können die Metriken basierend auf den verschiedenen Endpunkten oder Zieldatenbanken jedes Proxys oder dem Lese-/Schreib- und schreibgeschützten Datenverkehr zu jeder Datenbank aufteilen.

- -Dimensionssatz 1: `ProxyName`
- -Dimensionssatz 2: `ProxyName`, `EndpointName`
- -Dimensionssatz 3: `ProxyName`, `TargetGroup`, `Target`
- -Dimensionssatz 4: `ProxyName`, `TargetGroup`, `TargetRole`

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
`AvailabilityPercentage`	Der Prozentsatz der Zeit, für die die Zielgruppe in der durch die Dimension angegebenen Rolle verfügbar war. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute	Dimension set 4
`ClientConnections`	Die aktuelle Anzahl von Clientverbindungen. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute	Dimension set 1, Dimension set 2
`ClientConnectionsClosed`	Die Anzahl der geschlossenen Clientverbindungen. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 2
`ClientConnectionsInSetup`	Die aktuelle Anzahl der Clientverbindungen, die geöffnet sind, deren Einrichtung jedoch nicht abgeschlossen wurde. Diese Metrik wird jede	1 Minute	Dimension set 1, Dimension set 2

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
	Minute gemeldet. Die nützlichste Statistik für diese Metrik ist Summe.		
`ClientConnectionsNoTLS`	Die aktuelle Anzahl von Clientverbindungen ohne TLS (Transport Layer Security). Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute	Dimension set 1, Dimension set 2
`ClientConnectionsReceived`	Die Anzahl der empfangenen Clientverbindungsanforderungen. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 2
`ClientConnectionsSetupFailedAuth`	Die Anzahl der Clientverbindungsversuche, die aufgrund einer fehlerhaften Authentifizierungs- oder TLS-Konfiguration fehlgeschlagen sind. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 2

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
`ClientConnectionsSetupSucceeded`	Die Anzahl der Clientverbindungen, die erfolgreich mit einem Authentifizierungsmechanismus mit oder ohne TLS hergestellt wurden. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 2
`ClientConnectionsTLS`	Die aktuelle Anzahl von Clientverbindungen mit TLS. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute	Dimension set 1, Dimension set 2
`DatabaseConnectionRequests`	Die Anzahl der Anforderungen zum Erstellen einer Datenbankverbindung. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionRequestsWithTLS`	Die Anzahl der Anforderungen zum Erstellen einer Datenbankverbindung mit TLS. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 3, Dimension set 4

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
DatabaseConnections	Die aktuelle Anzahl von Datenbankverbindungen. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist Sum.	1 Minute	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsBorrowLatency	Die Zeit in Mikrosekunden, die der überwachte Proxy benötigt, um eine Datenbankverbindung zu erhalten. Die nützlichste Statistik für diese Metrik ist Sum.	1 Minute und höher	Dimension set 1, Dimension set 2
DatabaseConnectionsCurrentlyBorrowed	Die aktuelle Anzahl von Datenbankverbindungen im Ausleihstatus. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist Sum.	1 Minute	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsCurrentlyInTransaction	Die aktuelle Anzahl der Datenbankverbindungen in einer Transaktion. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist Sum.	1 Minute	Dimension set 1, Dimension set 3, Dimension set 4

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
`DatabaseConnectionsCurrentlySessionPinned`	Die aktuelle Anzahl von Datenbankverbindungen, die derzeit aufgrund von Operationen in Clientanforderungen, die den Sitzungsstatus ändern, angeheftet, bzw. fixiert, werden. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsSetupFailed`	Die Anzahl der fehlgeschlagenen Datenbankverbindungsanforderungen. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsSetupSucceeded`	Die Anzahl der erfolgreich aufgebauten Datenbankverbindungen mit oder ohne TLS. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 3, Dimension set 4

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
`DatabaseConnectionsWithTLS`	Die aktuelle Anzahl von Datenbankverbindungen mit TLS. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute	Dimension set 1, Dimension set 3, Dimension set 4
`MaxDatabaseConnectionsAllowed`	Die maximal zulässige Anzahl von Datenbankverbindungen. Diese Metrik wird jede Minute gemeldet. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute	Dimension set 1, Dimension set 3, Dimension set 4
`QueryDatabaseResponseLatency`	Die Zeit in Mikrosekunden, die die Datenbank benötigt hat, um auf die Abfrage zu antworten. Die nützlichste Statistik für diese Metrik ist `Average`.	1 Minute und höher	Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
`QueryRequests`	Die Anzahl der empfangenen Abfragen. Eine Abfrage, die mehrere Anweisungen enthält, wird als eine Abfrage gezählt. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 2
`QueryRequestsNoTLS`	Die Anzahl der Abfragen, die von Nicht-TLS-Verbindungen empfangen wurden. Eine Abfrage, die mehrere Anweisungen enthält, wird als eine Abfrage gezählt. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 2
`QueryRequestsTLS`	Die Anzahl der Abfragen, die von TLS-Verbindungen empfangen wurden. Eine Abfrage, die mehrere Anweisungen enthält, wird als eine Abfrage gezählt. Die nützlichste Statistik für diese Metrik ist `Sum`.	1 Minute und höher	Dimension set 1, Dimension set 2

Metrik	Beschreibung	Gültiger Zeitraum	CloudWatch-Dimensionssatz
`QueryResponseLatency`	Die Zeit in Mikrosekunden zwischen dem Abrufen einer Abfrageanforderung und dem darauf antwortenden Proxy. Die nützlichste Statistik für diese Metrik ist `Average`.	1 Minute und höher	Dimension set 1, Dimension set 2

Sie können Protokolle der RDS-Proxy-Aktivität unter CloudWatch in der AWS-Managementkonsole finden. Jeder Proxy hat einen Eintrag auf der Seite Log groups (Protokollgruppen) .

> **Important**
>
> Diese Protokolle sind zur menschlichen Verwendung zur Fehlerbehebung und nicht für den programmatischen Zugriff bestimmt. Das Format und der Inhalt der Protokolle können geändert werden.
> Insbesondere enthalten ältere Protokolle keine Präfixe, die den Endpunkt für jede Anfrage angeben. In neueren Protokollen wird jedem Eintrag der Name des zugehörigen Proxy-Endpunkts vorangestellt. Dieser Name kann der Name sein, den Sie für einen benutzerdefinierten Endpunkt angegeben haben, oder der spezielle Name `default` für Anforderungen mit dem Standard-Endpunkt eines Proxys.

Arbeiten mit RDS-Proxy-Ereignissen

Ein Ereignis weist auf eine Änderung in einer Umgebung hin, beispielsweise einer AWS-Umgebung, eines Service oder einer Anwendung eines SaaS-Partners (Software as a Service). Oder es kann sich um eine(n) Ihrer eigenen benutzerdefinierten Anwendungen oder Services handeln. Ein Beispiel: Amazon Aurora generiert ein Ereignis, wenn Sie einen RDS-Proxy erstellen oder ändern. Amazon Aurora überträgt Ereignisse nahezu in Echtzeit an Amazon EventBridge bereit. Nachfolgend finden

Sie eine Liste von RDS-Proxy-Ereignissen, die Sie abonnieren können, und ein Beispiel für ein RDS-Proxy-Ereignis.

Weitere Informationen zum Arbeiten mit Ereignissen finden Sie in den folgenden Abschnitten:

- Informationen zum Anzeigen von Ereignissen mithilfe der AWS-Managementkonsole, AWS CLI oder RDS API finden Sie unter Anzeigen von Amazon-RDS-Ereignissen.
- Informationen dazu, wie Sie Amazon Aurora dafür konfigurieren, Ereignisse an EventBridge zu senden, finden Sie unter Erstellen einer Regel, die bei einem Amazon Aurora-Ereignis ausgelöst wird.

RDS-Proxy-Ereignisse

Die folgende Tabelle zeigt die Ereigniskategorie und eine Liste von Ereignissen für den Fall, dass ein RDS-Proxy der Quelltyp ist.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0204	RDS hat den DB-Proxy *Name* geändert.	Keine
Konfigurationsänderung	RDS-EVENT-0207	RDS hat den Endpunkt des DB-Proxys *Name* geändert.	Keine
Konfigurationsänderung	RDS-EVENT-0213	RDS hat das Hinzufügen der DB-Instance erkannt und sie automatisch zur Zielgruppe des DB-Proxys *Name* hinzugefügt.	Keine
Konfigurationsänderung	RDS-EVENT-0214	RDS hat das Löschen der DB-Instance *Name* erkannt und sie automatisch aus der Zielgruppe *Name* des DB-Proxys *Name* entfernt.	Keine

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Konfigurationsänderung	RDS-EVENT-0215	RDS hat das Löschen des DB-Clusters *Name* erkannt und ihn automatisch aus der Zielgruppe *Name* des DB-Proxys *Name* entfernt.	Keine
Erstellung	RDS-EVENT-0203	RDS hat den DB-Proxy *Name* erstellt.	Keine
Erstellung	RDS-EVENT-0206	RDS hat den Endpunkt *Name* für den DB-Proxy *Name* erstellt.	Keine
Löschung	RDS-EVENT-0205	RDS hat den DB-Proxy *Name* erstellt.	Keine
Löschung	RDS-EVENT-0208	RDS hat den Endpunkt *Name* für den DB-Proxy *Name* gelöscht.	Keine
Ausfall	RDS-EVENT-0243	RDS konnte keine Kapazität für den Proxy *Name* bereitstellen, da in Ihren Subnetzen nicht genügend IP-Adressen verfügbar sind: *Name*. Stellen Sie sicher, dass Ihre Subnetze die Mindestanzahl unbenutzter IP-Adressen aufweisen, wie in der RDS-Proxy-Dokumentation empfohlen.	Informationen zur Bestimmung der empfohlenen Anzahl für Ihre Instance-Klasse finden Sie unter Planen der Kapazität von IP-Adressen.

Kategorie	RDS-Ereignis-ID	Fehlermeldung	Hinweise
Ausfall	RDS-EVENT-0275	RDS drosselte einige Verbindungen zum DB-Proxy *name*. Die Anzahl der gleichzeitigen Verbindungsanforderungen vom Client zum Proxy hat den Grenzwert überschritten.	Keine

Es folgt ein Beispiel eines RDS-Proxy-Ereignisses im JSON-Format. Das Ereignis zeigt, dass RDS den Endpunkt `my-endpoint` des RDS-Proxy `my-rds-proxy` geändert hat. Die Ereignis-ID lautet RDS-EVENT-0207.

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

Befehlszeilenbeispiele für RDS Proxy

Um zu sehen, wie Kombinationen von Verbindungsbefehlen und SQL-Anweisungen mit RDS Proxy interagieren, sehen Sie sich die folgenden Beispiele an.

Beispiele

- Preserving Connections to a MySQL Database Across a Failover
- Adjusting the max_connections Setting for an Aurora DB Cluster

Example Verbindungen zu einer MySQL-Datenbank über ein Failover beibehalten

Dieses MySQL-Beispiel zeigt, wie offene Verbindungen bei einem Failover weiterhin funktionieren. Ein Beispiel ist, wenn Sie eine Datenbank neu starten oder sie aufgrund eines Problems nicht mehr verfügbar ist. In diesem Beispiel werden ein Proxy mit dem Namen `the-proxy` und ein Aurora-DB-Cluster mit den DB-Instances `instance-8898` und `instance-9814` verwendet. Wenn Sie den `failover-db-cluster`-Befehl über die Linux-Befehlszeile ausführen, wechselt die Schreiber-Instance, mit der der Proxy verbunden ist, zu einer anderen DB-Instance. Sie können sehen, dass sich die dem Proxy zugeordnete DB-Instance ändert, während die Verbindung geöffnet bleibt.

```
$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-8898 is the writer.
$ fg
```

```
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-8898      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
+---------------+---------------+
| Variable_name | Value         |
+---------------+---------------+
| hostname      | ip-10-1-3-178 |
+---------------+---------------+
1 row in set (0.02 sec)
```

Example Anpassen der Einstellung max_connections für einen Aurora-DB-Cluster.

Dieses Beispiel zeigt, wie Sie die `max_connections`-Einstellung für einen Aurora MySQL-DB-Cluster anpassen können. Dazu erstellen Sie eine eigene DB-Cluster-Parametergruppe basierend auf den Standardparametereinstellungen für Cluster, die mit MySQL 5.7 kompatibel sind. Sie geben einen Wert für die `max_connections`-Einstellung an und überschreiben die Formel, die den Standardwert festlegt. Sie ordnen die DB-Clusterparametergruppe Ihrem DB-Cluster zu.

```
export REGION=us-east-1
```

```
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-57-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-57

aws rds create-db-parameter-group --region $REGION \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \
  --description "Aurora MySQL 5.7 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"

echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-
group-name $CLUSTER_PARAM_GROUP \
  --parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"

echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"
```

Fehlersuche für RDS-Proxy

Im Folgenden finden Sie Ideen zur Fehlerbehebung für einige häufig auftretende Probleme mit dem RDS-Proxy sowie Informationen zu CloudWatch Protokollen für RDS Proxy.

In den RDS-Proxy-Protokollen wird jedem Eintrag der Name des zugehörigen Proxy-Endpunkts vorangestellt. Dieser Name kann derjenige sein, den Sie für einen benutzerdefinierten Endpunkt angegeben haben. Es kann sich auch um den speziellen Namen `default` für den Standardendpunkt eines Proxys handeln, der read/write Anfragen ausführt. Weitere Informationen zu den Proxy-Endpunkten finden Sie unter Arbeiten mit Amazon-RDS-Proxy-Endpunkten.

Themen

- Überprüfen der Konnektivität für einen Proxy
- Häufige Probleme und Lösungen
- Beheben von RDS-Proxy-Problemen mit RDS für MySQL
- Beheben von RDS-Proxy-Problemen mit RDS für PostgreSQL

Überprüfen der Konnektivität für einen Proxy

Mit den folgenden Befehle können Sie überprüfen, ob alle Komponenten wie Proxy-, Datenbank- und Rechen-Instances in der Verbindung miteinander kommunizieren können.

Untersuchen Sie den Proxy selbst mit dem describe-db-proxiesBefehl. Untersuchen Sie auch die zugehörige Zielgruppe mit dem Befehl describe-db-proxy-target-groups. Überprüfen Sie, ob die Details der Ziele mit dem Aurora-Cluster übereinstimmen, den Sie dem Proxy zuordnen möchten. Verwenden Sie Befehle wie die folgenden.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

Um zu überprüfen, ob der Proxy eine Verbindung zur zugrunde liegenden Datenbank herstellen kann, überprüfen Sie mithilfe des describe-db-proxy-targetsBefehls die in den Zielgruppen angegebenen Ziele. Verwenden Sie einen Befehl wie den folgenden.

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

Die Ausgabe des describe-db-proxy-targetsBefehls enthält ein `TargetHealth` Feld. Sie können die Felder `State`, `Reason` und `Description` in `TargetHealth` überprüfen, um festzustellen, ob der Proxy mit der zugrunde liegenden DB-Instance kommunizieren kann.

- Der Wert `State` für `AVAILABLE` gibt an, dass der Proxy eine Verbindung mit der DB-Instance herstellen kann.

- Der Wert `State` für `UNAVAILABLE` weist auf ein temporäres oder dauerhaftes Verbindungsproblem hin. Überprüfen Sie in diesem Fall die Felder `Reason` und `Description`. Wenn beispielsweise der `Reason` den Wert `PENDING_PROXY_CAPACITY` hat, versuchen Sie erneut , eine Verbindung herzustellen, nachdem der Proxy seinen Skalierungsvorgang beendet hat. Wenn `Reason` den Wert `UNREACHABLE`, `CONNECTION_FAILED` oder `AUTH_FAILURE` hat, verwenden Sie die Erläuterung aus dem Feld `Description`, um das Problem leichter zu diagnostizieren.
- Das Feld `State` hat möglicherweise für kurze Zeit einen Wert von `REGISTERING`, bevor Sie zu `AVAILABLE` oder `UNAVAILABLE` wechseln.

Wenn der folgende Netcat-Befehl (`nc`) erfolgreich ist, können Sie von der EC2 Instanz oder einem anderen System aus, auf dem Sie angemeldet sind, auf den Proxy-Endpunkt zugreifen. Dieser Befehl meldet einen Fehler, wenn Sie sich nicht in derselben VPC wie der Proxy und die zugehörige Datenbank befinden. Möglicherweise können Sie sich direkt bei der Datenbank anmelden, ohne sich in derselben VPC zu befinden. Sie können sich jedoch nur am Proxy anmelden, wenn Sie sich in derselben VPC befinden.

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

Sie können die folgenden Befehle verwenden, um sicherzustellen, dass Ihre EC2 Instanz über die erforderlichen Eigenschaften verfügt. Insbesondere muss die VPC für die EC2 Instance mit der VPC für den Aurora-Cluster der RDS-DB-Instance identisch sein, mit dem der Proxy eine Verbindung herstellt.

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

Untersuchen Sie die für den Proxy verwendeten Secrets Manager-Secrets.

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

Stellen Sie sicher, dass das von `get-secret-value` angezeigte Feld `SecretString` als JSON-Zeichenfolge codiert ist, die die Felder `username` und `password` enthält. Das folgende Beispiel zeigt das Format des `SecretString`-Feldes.

```
{
  "ARN": "some_arn",
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

Überprüfen Sie bei der Behebung von Problemen mit der IAM-Authentifizierung Folgendes:

- In der Datenbank ist die IAM-Authentifizierung aktiviert.
- Der Proxy ist mit dem richtigen Authentifizierungsschema konfiguriert.
- Die IAM-Richtlinien in der IAM-Rolle, die dem Proxy zur Verfügung gestellt wurden, gewähren der entsprechenden Datenbank und ihrem Benutzernamen die erforderlichen `rds-db:connect` Berechtigungen.
- Für die end-to-end IAM-Authentifizierung existieren Datenbankbenutzer, die den IAM-Benutzer- oder Rollennamen entsprechen.
- SSL/TLS ist für die Verbindung aktiviert.

Häufige Probleme und Lösungen

In diesem Abschnitt werden einige häufig auftretende Probleme und mögliche Lösungen bei der Verwendung von RDS-Proxy beschrieben.

Wenn in der Beschreibung `TargetHealth` `Proxy does not have any registered credentials` angegeben ist, überprüfen Sie nach dem Ausführen des CLI-Befehls `aws rds describe-db-proxy-targets` Folgendes:

- Es sind Anmeldeinformationen registriert, damit der Benutzer auf den Proxy zugreifen kann.
- Die IAM-Rolle für den Zugriff auf das vom Proxy verwendete Secret von Secrets Manager ist gültig.

Beim Erstellen eines DB-Proxys oder beim Herstellen einer Verbindung mit einem DB-Proxy können die folgenden RDS-Ereignisse auftreten.

Kategorie	RDS-Ereignis-ID	Description
Ausfall	RDS-EVENT-0243	RDS konnte keine Kapazität für den Proxy bereitstellen, da in Ihren Subnetzen nicht genügend IP-Adressen verfügbar sind. Stellen Sie sicher, dass Ihre Subnetze die Mindestanzahl unbenutzter IP-Adressen aufweisen, um das Problem zu lösen. Informationen zur Bestimmung der empfohlenen Anzahl für Ihre Instance-Klasse finden Sie unter Planen der Kapazität von IP-Adressen.
Ausfall	RDS-EVENT-0275	RDS hat einige Verbindungen zum DB-Proxy gedrosselt. *name* Die Anzahl der gleichzeitigen Verbindungsanforderungen vom Client zum Proxy hat den Grenzwert überschritten.

Beim Erstellen eines neuen Proxys oder beim Herstellen einer Verbindung mit einem Proxy können die folgenden Probleme auftreten.

Fehler	Ursachen oder Problemumgehungen
`403: The security token included in the request is invalid`	Wählen Sie eine vorhandene IAM-Rolle aus, anstatt eine neue zu erstellen.

Beheben von RDS-Proxy-Problemen mit RDS für MySQL

Beim Herstellen einer Verbindung mit einem MySQL-Proxy können die folgenden Probleme auftreten.

Fehler	Ursachen oder Problemumgehungen
`ERROR 1040 (HY000): Connections rate limit exceeded (`*`limit_value`*` )`	Die Rate der Verbindungsanforderungen vom Client zum Proxy hat den Grenzwert überschritten.
`ERROR 1040 (HY000): IAM authentication rate limit exceeded`	Die Anzahl der gleichzeitigen Anforderungen mit IAM-Authentifizierung vom Client an den Proxy hat den Grenzwert überschritten.
`ERROR 1040 (HY000): Number simultaneous connections exceeded (`*`limit_value`*` )`	Die Anzahl der gleichzeitigen Verbindungsanforderungen vom Client zum Proxy hat den Grenzwert überschritten.
`ERROR 1045 (28000): Access denied for user '`*`DB_USER`*`'@'%' (using password: YES)`	Das vom Proxy verwendete Secrets Manager-Geheimnis stimmt nicht mit dem Benutzernamen und dem Passwort eines vorhandenen Datenbankbenutzers überein. Aktualisieren Sie entweder die Anmeldeinformationen im Secrets Manager-Geheimnis oder stellen Sie sicher, dass der Datenbankbenutzer vorhanden ist und über das gleiche Passwort wie im Geheimnis verfügt.
`ERROR 1105 (HY000): Unknown error`	Es ist ein unbekannter Fehler aufgetreten.

Fehler	Ursachen oder Problemumgehungen
`ERROR 1231 (42000): Variable ''character_set_client'' can't be set to the value of value`	Der für den `character_set_client` -Parameter gesetzte Wert ist ungültig. Beispielsweise ist der Wert `ucs2` ungültig, da er den MySQL-Server zum Absturz bringen kann.
`ERROR 3159 (HY000): This RDS Proxy requires TLS connections.`	Sie haben die Einstellung Transport Layer Security erfordern im Proxy aktiviert, aber Ihre Verbindung enthielt den Parameter `ssl-mode=DISABLED` im MySQL-Client. Führen Sie eine der folgenden Aufgaben aus: • Deaktivieren Sie die Einstellung Transport Layer Security erfordern. • Stellen Sie eine Verbindung mit der Datenbank mit der Mindestei nstellung `ssl-mode=REQUIRED` im MySQL-Client her.
`ERROR 2026 (HY000): SSL connection error: Internal Server Error`	Der TLS-Handshake an den Proxy ist fehlgeschlagen. Einige mögliche Gründe sind: • SSL ist erforderlich, aber der Server unterstützt dies nicht. • Es ist ein interner Serverfehler aufgetreten. • Es ist ein fehlerhafter Handshake aufgetreten.
`ERROR 9501 (HY000): Timed-out waiting to acquire database connection`	Auf dem Proxy ist beim Herstellen einer Datenbankverbindung eine Zeitüberschreitung aufgetreten. Einige mögliche Gründe sind: • Der Proxy kann keine Datenbankverbindung herstellen, da die maximale Anzahl an Verbindungen erreicht wurde. • Der Proxy kann keine Datenbankverbindung herstellen, da die Datenbank nicht verfügbar ist.

Beheben von RDS-Proxy-Problemen mit RDS für PostgreSQL

Beim Herstellen einer Verbindung mit einem PostgreSQL-Proxy können die folgenden Probleme auftreten.

Fehler	Ursache	Lösung
`ERROR 28000: IAM authentication is allowed only with SSL connections.`	Der Benutzer hat versucht, mithilfe der IAM-Authentifizierung mit der Einstellung `sslmode=disable` im PostgreSQL-Client eine Verbindung zur Datenbank herzustellen.	Der Benutzer muss eine Verbindung mit der Datenbank mit der Mindesteinstellung `sslmode=require` im PostgreSQL-Client herstellen. Weitere Informationen finden Sie in der PostgreSQL SSL Support-Dokumentation.
`ERROR 28000: This RDS proxy has no credentials for the role` *`role_name`* `. Check the credentials for this role and try again.`	Es gibt kein Secrets Manager-Secret für diese Rolle.	Fügen Sie ein Secrets Manager-Secret diese Rolle hinzu. Weitere Informationen finden Sie unter Konfigurieren der IAM-Authentifizierung für RDS-Proxy.
`ERROR 28000: RDS supports only IAM, MD5, or SCRAM authentication.`	Der Datenbank-Client, der für die Verbindung mit dem Proxy verwendet wird, nutzt einen Authentifizierungsmechanismus, der derzeit vom Proxy nicht unterstützt wird.	Wenn Sie keine IAM-Authentifizierung verwenden, verwenden Sie die MD5 oder die SCRAM-Passwortauthentifizierung.
`ERROR 28000: A user name is missing from the connection startup packet. Provide a user name for this connection.`	Der Datenbankclient, der zum Herstellen einer Verbindung mit dem Proxy verwendet wird, sendet beim Versuch, eine Verbindung herzustellen, keinen Benutzernamen.	Stellen Sie sicher, dass Sie beim Einrichten einer Verbindung zum Proxy mit dem PostgreSQL-Client Ihrer Wahl einen Benutzernamen definieren.

Fehler	Ursache	Lösung
`ERROR 28000: IAM is allowed only with SSL connections.`	Ein Client hat versucht, eine Verbindung über die IAM-Authentifizierung herzustellen, aber SSL war nicht aktiviert.	Aktivieren Sie SSL im PostgreSQL-Client.
`ERROR 28000: This RDS Proxy requires TLS connections.`	Der Benutzer hat die Option Transport Layer Security erfordern aktiviert, hat aber versucht, auf dem PostgreSQL-Client eine Verbindung mit `sslmode=disable` herzustellen.	Führen Sie einen der folgenden Schritte aus, um diesen Fehler zu beheben: • Deaktivieren Sie die Option Transport Layer Security erfordern des Proxys. • Herstellen einer Verbindung mit der Datenbank mit der Mindesteinstellung `sslmode=allow` im PostgreSQL-Client.

Fehler	Ursache	Lösung
`ERROR 28P01: IAM authentication failed for user` *`user_name`* `. Check the IAM token for this user and try again.`	Dies kann folgende Ursachen haben: • Der Client hat den falschen IAM-Benutzernamen angegeben. • Der Client hat ein falsches IAM-Autorisierungstoken für den Benutzer angegeben. • Der Client verwendet eine IAM-Richtlinie, die nicht über die erforderlichen Berechtigungen verfügt. • Der Client hat ein abgelaufenes IAM-Autorisierungstoken für den Benutzer bereitgestellt.	Gehen Sie folgendermaßen vor, um diesen Fehler zu beheben: 1. Bestätigen Sie, dass der bereitgestellte IAM-Benutzer vorhanden ist. 2. Vergewissern Sie sich, dass das IAM-Autorisierungstoken dem bereitgestellten IAM-Benutzer gehört. 3. Bestätigen Sie, dass die IAM-Richtlinie über ausreichende Berechtigungen für RDS verfügt. 4. Überprüfen Sie die Gültigkeit des verwendeten IAM-Autorisierungstokens.
`ERROR 28P01: The password that was provided for the role` *`role_name`* `is wrong.`	Das Passwort für diese Rolle stimmt nicht mit dem Secrets Manager-Secret überein.	Überprüfen Sie den Schlüssel für diese Rolle in Secrets Manager, um festzustellen, ob das Passwort mit dem in Ihrem PostgreSQL-Client verwendeten Passwort übereinstimmt.
`ERROR 28P01: The IAM authentication failed for the role` *`role_name`* `. Check the IAM token for this role and try again.`	Es gibt ein Problem mit dem IAM-Token, das für die IAM-Authentifizierung verwendet wird.	Generieren Sie ein neues Authentifizierungstoken und verwenden Sie es in einer neuen Verbindung.

Fehler	Ursache	Lösung
`ERROR 0A000: Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.`	Der PostgreSQL-Client, mit dem eine Verbindung zum Proxy hergestellt wird, verwendet ein Protokoll, das älter als 3.0 ist.	Verwenden Sie einen neueren PostgreSQL-Client, der das 3.0-Messaging-Prot okoll unterstützt. Wenn Sie die `psql` PostgreSQL-CLI verwenden, verwenden Sie Version 7.4 oder höher.
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support streaming replication mode.`	Der PostgreSQL-Client, der für die Verbindung mit dem Proxy verwendet wird, versucht, den Streaming-Replikationsmodus zu verwenden, der derzeit vom RDS-Proxy nicht unterstützt wird.	Deaktivieren Sie den Streaming-Replikationsmodus im PostgreSQL-Client, der für die Verbindung verwendet wird.
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support the option` *`option_name`* `.`	Über die Startmeldung fordert der PostgreSQL-Client, der für die Verbindung mit dem Proxy verwendet wird, eine Option an, die derzeit vom RDS-Proxy nicht unterstützt wird.	Deaktivieren Sie die Option, die in der obigen Nachricht als nicht unterstützt angezeigt wird, in dem PostgreSQL-Client, der für die Verbindung verwendet wird.
`ERROR 53300: The IAM authentication failed because of too many competing requests.`	Die Anzahl der gleichzeitigen Anforderungen mit IAM-Authe ntifizierung vom Client an den Proxy hat den Grenzwert überschritten.	Reduzieren Sie die Rate, mit der Verbindungen mit IAM-Authentifizierung von einem PostgreSQL-Client hergestellt werden.
`ERROR 53300: The maximum number of client connections to the proxy exceeded` *`number_value`* `.`	Die Anzahl der gleichzeitigen Verbindungsanforderungen vom Client zum Proxy hat den Grenzwert überschritten.	Reduzieren Sie die Anzahl der aktiven Verbindungen von PostgreSQL-Clients zu diesem RDS-Proxy.

Fehler	Ursache	Lösung
`ERROR 53300: Rate of connection to proxy exceeded` *`number_value`* `.`	Die Rate der Verbindungsanforderungen vom Client zum Proxy hat den Grenzwert überschritten.	Reduzieren Sie die Rate, mit der Verbindungen von einem PostgreSQL-Client hergestellt werden.
`ERROR XX000: Unknown error.`	Es ist ein unbekannter Fehler aufgetreten.	Wenden Sie sich an den AWS Support, um das Problem zu untersuchen.
`ERROR 08000: Timed-out waiting to acquire database connection.`	Auf dem Proxy ist beim Herstellen einer Datenbankverbindung innerhalb der in der Einstellung `ConnectionBorrowTimeout` angegebenen Dauer eine Zeitüberschreitung aufgetreten. Einige mögliche Gründe sind: • Der Proxy kann keine Datenbankverbindung herstellen, da die maximale Anzahl an Verbindungen erreicht wurde. • Der Proxy kann keine Datenbankverbindung herstellen, da die Datenbank nicht verfügbar ist oder wenn der Herstellen einer Verbindung zur Datenbank länger dauert als das konfigurierte `ConnectionBorrowTimeout` .	Folgende Lösungen sind möglich: • Vermeiden Sie das Pinning von Proxyverbindungen. Siehe Vermeiden von Pinning beim RDS-Proxy. • Überprüfen Sie die Einstellungen `ConnectionBorrowTimeout` und `MaxConnectionsPercent` . Siehe Überlegungen zu RDS-Proxy-Verbindungen. • Überprüfen Sie die Verfügbarkeit des Ziels. Weitere Informationen finden Sie unter `AvailabilityPercentage` in Überwachen von RDS-Proxy-Metriken mit Amazon CloudWatch.

Fehler	Ursache	Lösung
`ERROR XX000: Request returned an error: database_error .`	Die vom Proxy hergestellte Datenbankverbindung hat einen Fehler zurückgegeben.	Die Lösung hängt vom spezifischen Datenbank fehler ab. Ein Beispiel is: `Request returned an error: database "your-database-name" does not exist`. Das bedeutet, dass der angegebene Datenbankname nicht auf dem Datenbankserver existiert. Oder es bedeutet, dass der als Datenbankname verwendet e Benutzername (wenn kein Datenbankname angegeben ist) auf dem Server nicht vorhanden ist.
`ERROR 53300: The IAM authentication failed because of too many competing requests.`	Die Anzahl der gleichzeitigen Anforderungen mit IAM-Authe ntifizierung vom Client an den Proxy hat den Grenzwert überschritten.	Reduzieren Sie die Rate, mit der Verbindungen mit IAM-Authentifizierung von einem PostgreSQL-Client hergestellt werden.

Fehler	Ursache	Lösung
`ERROR 28000: Enable IAM authentication for the client connection to the proxy and try again.`	Der RDS-Proxy kann keine Verbindung zur Datenbank herstellen, da die IAM-Authentifizierung für die Client-Verbindung zum Proxy nicht aktiviert ist. Dies tritt auf, wenn der `DefaultAuthScheme` Parameter des Proxys auf `IAM_AUTH` mit einem registrierten Benutzer gesetzt ist, der Client jedoch die Kennwortauthentifizierung anstelle der IAM-Authentifizierung verwendet.	Aktivieren Sie die IAM-Authentifizierung für die Client-Verbindung zum Proxy und versuchen Sie es erneut.
`ERROR 28000: Configure IAM authentication as the DefaultAuthScheme in your proxy and try again.`	Der RDS-Proxy kann keine Verbindung zur Datenbank herstellen, da der `DefaultAuthScheme` nicht auf `IAM_AUTH` eingestellt ist. Der `DefaultAuthScheme` Parameter des Proxys ist auf gesetzt`NONE`, aber der Client versucht, die IAM-Authentifizierung zu verwenden.	Stellen Sie `IAM_AUTH` für `DefaultAuthScheme` Ihren Proxy auf ein und versuchen Sie es erneut.

Problembehandlung bei gelöschter **`postgres`**-Datenbank

Wenn Sie die `postgres`-Datenbank versehentlich aus Ihrer Instance löschen, müssen Sie die Datenbank wiederherstellen, um die Verbindung zu Ihrer Instance wiederherzustellen. Führen Sie die folgenden Befehle mit Ihrer DB-Instance aus:

```
CREATE DATABASE postgres;
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

Verwenden von RDS-Proxy mit AWS CloudFormation

Sie können RDS-Proxy mit AWS CloudFormation nutzen. Auf diese Weise können Sie Gruppen verwandter Ressourcen erstellen. Eine solche Gruppe kann einen Proxy enthalten, der eine Verbindung mit einem neu erstellten Aurora-DB-Cluster herstellen kann. Die RDS-Proxy-Unterstützung in CloudFormation umfasst zwei neue Registrierungstypen: `DBProxy` und `DBProxyTargetGroup`.

Die folgende Auflistung zeigt eine CloudFormation-Beispielvorlage für RDS-Proxy.

```
Resources:
 DBProxy:
   Type: AWS::RDS::DBProxy
   Properties:
     DBProxyName: CanaryProxy
     EngineFamily: MYSQL
     RoleArn:
      Fn::ImportValue: SecretReaderRoleArn
     Auth:
       - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
     VpcSubnetIds:
       Fn::Split: [",", "Fn::ImportValue": SubnetIds]

 ProxyTargetGroup:
   Type: AWS::RDS::DBProxyTargetGroup
   Properties:
     DBProxyName: CanaryProxy
     TargetGroupName: default
     DBInstanceIdentifiers:
       - Fn::ImportValue: DBInstanceName
   DependsOn: DBProxy
```

Weitere Informationen zu den Ressourcen in diesem Beispiel finden Sie unter `DBProxy` und `DBProxyTargetGroup`.

Weitere Informationen zu den Ressourcen, die Sie mit CloudFormation erstellen können, finden Sie in der Referenz zu RDS-Ressourcentypen.

Verwenden von RDS Proxy mit globalen Aurora-Datenbanken

Eine globale Aurora-Datenbank ist eine einzige Datenbank, die mehrere AWS-Regionen abdeckt, um globale Lesezugriffe mit niedriger Latenz und eine Notfallwiederherstellung nach regionalen Ausfällen zu ermöglichen. Es bietet eine integrierte Fehlertoleranz für Ihre Bereitstellung, da die DB-Instance nicht auf eine einzelne AWS-Region, sondern auf mehreren Regionen und verschiedenen Availability Zones zurückgreifen kann. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Global Database.

Sie können RDS Proxy mit jedem DB-Cluster in einer globalen Aurora-Datenbank verwenden. Bevor Sie beginnen, diese Funktionen zusammen zu verwenden, sollten Sie sich mit den folgenden Informationen vertraut machen.

> **Important**
>
> Wenn der DB-Cluster Teil einer globalen Datenbank mit aktivierter Schreibweiterleitung ist, reduzieren Sie den `MaxConnectionsPercent`-Wert Ihres Proxys um das Kontingent, das der Schreibweiterleitung zugewiesen ist. Das Kontingent für die Schreibweiterleitung ist im DB-Cluster-Parameter `aurora_fwd_writer_max_connections_pct` festgelegt. Informationen zur Schreibweiterleitung finden Sie unter Verwenden der Schreibweiterleitung in einer Amazon Aurora globalen Datenbank.

Einschränkungen für RDS Proxy mit globalen Datenbanken

Wenn im Aurora-DB-Cluster die Schreibweiterleitung aktiviert ist, unterstützt RDS Proxy den `SESSION`-Wert für die `aurora_replica_read_consistency`-Variable nicht. Durch das Festlegen dieses Werts kann unerwartetes Verhalten auftreten.

So funktionieren RDS-Proxy-Endpunkte mit globalen Datenbanken

Wenn Sie verstehen, wie RDS-Proxy-Endpunke mit globalen Datenbanken funktionieren, können Sie Ihre Anwendungen, die Aurora-Datenbanken verwenden, mit diesen beiden Funktionen besser verwalten.

Bei einem Proxy mit dem primären Cluster einer globalen Datenbank als registriertem Ziel funktionieren die Proxy-Endpunkte genauso wie bei jedem beliebigen Aurora-DB-Cluster. Die Lese-/Schreib-Endpunkte des Proxy senden alle Anforderungen an die Writer-Instance des Clusters.

Die schreibgeschützten Endpunkte des Proxys senden alle Anfragen an die Reader-Instances. Wenn ein Reader nicht mehr verfügbar ist, während eine Verbindung geöffnet ist, leitet RDS Proxy nachfolgende Abfragen in der Verbindung an eine andere Reader-Instance weiter. Bei einem Proxy mit einem sekundären Cluster als registriertem Ziel werden Anfragen, die an die schreibgeschützten Endpunkte des Proxys gesendet werden, auch an die Reader-Instances gesendet. Da der Cluster keine Writer-Instances hat, schlagen Anfragen, die an Lese/Schreib-Endpunkte gesendet werden, mit dem Fehler „`The target group doesn't have any associated read/write instances`“ fehl.

Eine globale Datenbank-Umstellung und Failover-Vorgänge umfassen einen Rollenwechsel zwischen dem primären und einem der sekundären DB-Cluster. Wenn der ausgewählte sekundäre Cluster zum neuen primären Cluster wird, wird eine seiner Reader-Instances zum Writer hochgestuft. Diese DB-Instance ist jetzt die neue Writer-Instance für den globalen Cluster. Stellen Sie sicher, dass Sie die Schreibvorgänge Ihrer Anwendung an den entsprechenden Lese/Schreib-Endpunkt des Proxys umleiten, der dem neuen primären Cluster zugeordnet ist. Dieser Proxy-Endpunkt kann der Standardendpunkt oder ein benutzerdefinierter Lese/Schreib-Endpunkt sein.

RDS Proxy stellt alle Anfragen über Lese/Schreib-Endpunkte in die Warteschlange und sendet sie an die Schreib-Iinstance des neuen primären Clusters, sobald dieser verfügbar ist. Dies geschieht unabhängig davon, ob die Umstellung oder der Failover-Vorgang abgeschlossen ist. Während der Umstellung oder des Failovers akzeptiert der Standardendpunkt des Proxys für den alten primären Cluster weiterhin Schreibvorgänge. Sobald dieser Cluster jedoch zu einem sekundären Cluster wird, schlagen alle Schreibvorgänge fehl. Informationen dazu, wie und wann bestimmte globale Umstellungs- oder Failover-Aufgaben ausgeführt werden müssen, finden Sie in den folgenden Themen:

- Globale Datenbank-Umstellung – Durchführen von Umstellungen für Amazon Aurora Global Databases
- Globales Datenbank-Failover – Wiederherstellen einer globalen Amazon Aurora-Datenbank nach einem ungeplanten Ausfall

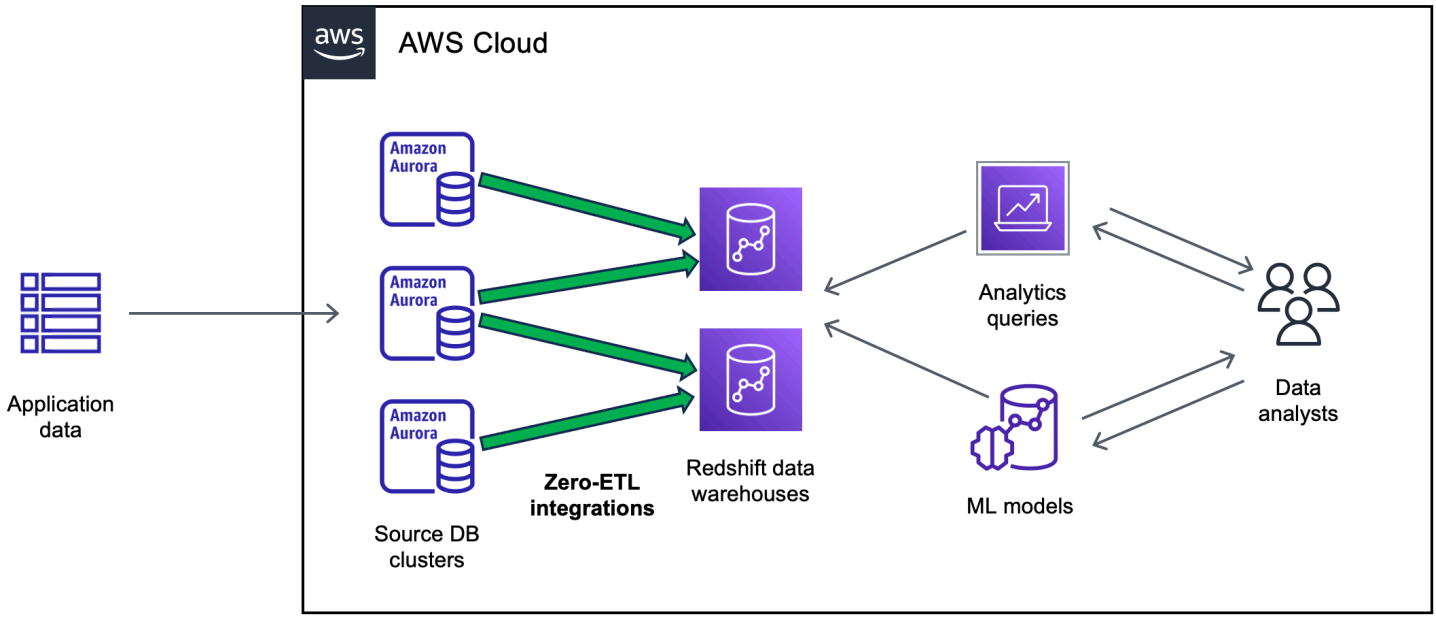
Null-ETL-Integrationen von Aurora

Es handelt sich um eine vollständig verwaltete Lösung, die Transaktionsdaten in Ihrem Analyseziel verfügbar macht, nachdem diese in einen Aurora-DB-Cluster geschrieben wurden. Extract, Transform, Load (ETL) bezeichnet die Kombination von Daten aus mehreren Quellen in einem großen, zentralen Data Warehouse.

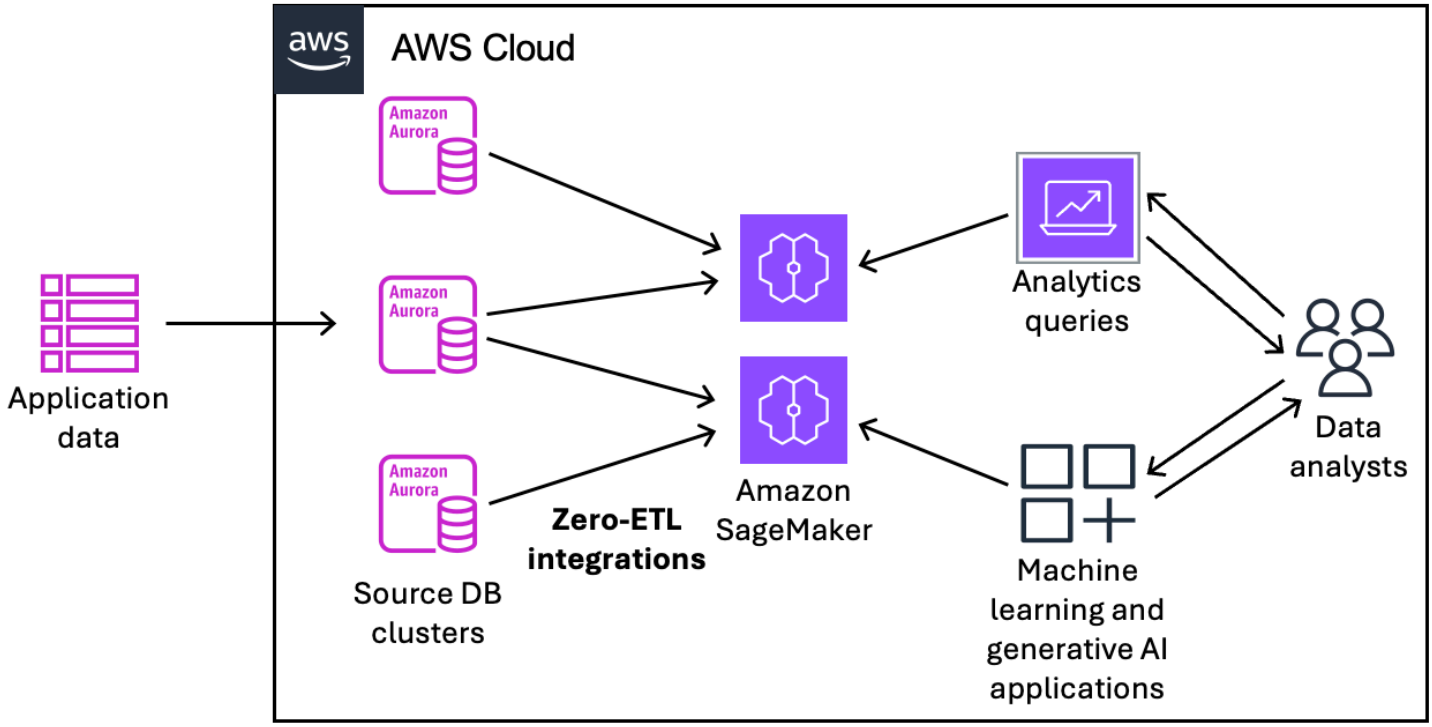
Eine Null-ETL-Integration macht die Daten in Ihrem Aurora-DB-Cluster in Amazon Redshift oder einem Amazon SageMaker AI -Lakehouse nahezu in Echtzeit verfügbar. Sobald sich diese Daten im Ziel-Data Warehouse oder Data Lake befinden, können Sie Ihre Analyse-, ML- und KI-Workloads mithilfe der integrierten Funktionen wie maschinelles Lernen, materialisierte Ansichten, gemeinsame Nutzung von Daten, Verbundzugriff auf mehrere Datenspeicher und Data Lakes sowie Integrationen mit Amazon SageMaker AI, Quick Suite und anderen unterstützen. AWS-Services

Um eine Null-ETL-Integration zu erstellen, geben Sie einen Aurora-DB-Cluster als Quelle und ein unterstütztes Data Warehouse oder Lakehouse als Ziel an. Bei der Integration werden Daten aus der Quelldatenbank in das Ziel-Data-Warehouse oder -Lakehouse repliziert.

Das folgende Diagramm veranschaulicht diese Funktion für die Null-ETL-Integration mit Amazon Redshift:



Das folgende Diagramm veranschaulicht diese Funktion für die Null-ETL-Integration mit einem Amazon SageMaker AI -Lakehouse:



Die Integration überwacht den Zustand der Datenpipeline und behebt nach Möglichkeit Probleme. Sie können Integrationen aus mehreren Aurora-DB-Clustern in ein einziges Ziel-Data-Warehouse oder -Lakehouse erstellen und dadurch Erkenntnisse aus mehreren Anwendungen gewinnen.

Weitere Informationen zu den Preisen für Null-ETL-Integrationen finden Sie unter Amazon Aurora – Preise und Amazon Redshift – Preise.

Themen

- Vorteile
- Die wichtigsten Konzepte
- Einschränkungen
- Kontingente
- Unterstützte Regionen
- Erste Schritte mit Null-ETL-Integrationen von Aurora
- Erstellen von Null-ETL-Integrationen von Aurora mit Amazon Redshift
- Erstellen von Null-ETL-Integrationen von Aurora mit einem Amazon SageMaker-Lakehouse
- Datenfilterung für Null-ETL-Integrationen für Aurora
- Hinzufügen von Daten zu einem Quell-DB-Cluster von Aurora und Abfragen der Daten
- Anzeigen und Überwachen von Null-ETL-Integrationen von Aurora

- Ändern der Null-ETL-Integrationen von Aurora
- Löschen von Null-ETL-Integrationen von Aurora
- Fehlerbehebung bei Null-ETL-Integrationen von Aurora

Vorteile

Null-ETL-Integrationen von Aurora bieten die folgenden Vorteile:

- Sie helfen Ihnen dabei, ganzheitliche Erkenntnisse aus mehreren Datenquellen zu gewinnen.
- Eliminieren Sie die Erfordernis zur Erstellung und Verwaltung komplexer Daten-Pipelines, die Extract, Transform, Load (ETL)-Operationen ausführen. Null-ETL-Integrationen beseitigen die Herausforderungen, die mit dem Aufbau und der Verwaltung von Pipelines einhergehen, indem sie diese für Sie bereitstellen und verwalten.
- Sie reduzieren den Betriebsaufwand und die Kosten, sodass Sie sich ganz auf die Verbesserung Ihrer Anwendungen konzentrieren können.
- Sie ermöglichen die Nutzung der Analyse- und ML-Funktionen des Ziels zum Gewinnen von Erkenntnissen aus Transaktions- und anderen Daten, um effektiv auf (zeit-)kritische Ereignisse reagieren zu können.

Die wichtigsten Konzepte

Wenn Sie mit Null-ETL-Integrationen beginnen, sollten Sie die folgenden Konzepte berücksichtigen:

Integration

Eine vollständig verwaltete Daten-Pipeline, die automatisch Transaktionsdaten und Schemata aus einem Aurora-DB-Cluster in ein Data Warehouse oder einen Katalog repliziert.

Quell-DB-Cluster

Der Aurora-DB-Cluster, von wo aus die Daten repliziert werden. Sie können einen DB-Cluster angeben, der bereitgestellte DB-Instances oder Aurora Serverless v2 DB-Instances als Quelle verwendet.

Target

Das Data Warehouse oder Lakehouse, in das die Daten repliziert werden. Es gibt zwei Arten von Data Warehouse: ein bereitgestelltes Cluster-Data-Warehouse und ein Serverless-

Data-Warehouse. Ein bereitgestelltes Cluster-Data-Warehouse ist eine Sammlung von Datenverarbeitungsressourcen, den sogenannten Knoten, die zu einer Gruppe, einem sogenannten Cluster, zusammengefasst werden. Ein Serverless-Data-Warehouse besteht aus einer Arbeitsgruppe, die Datenverarbeitungsressourcen speichert, und einem Namespace, in dem die Datenbankobjekte und Benutzer gespeichert sind. In beiden Data Warehouses wird eine Analyse-Engine ausgeführt und beide enthalten eine oder mehrere Datenbanken.

Ein Ziel-Lakehouse besteht aus Katalogen, Datenbanken, Tabellen und Ansichten. Weitere Informationen zur Lakehouse-Architektur finden Sie unter SageMaker Lakehouse components im Amazon SageMaker AI Unified Studio-Benutzerhandbuch.

Mehrere Quell-DB-Cluster können in dasselbe Ziel schreiben.

Weitere Informationen finden Sie unter Architektur des Data-Warehouse-Systems im Entwicklerhandbuch zu Amazon Redshift.

Einschränkungen

Die folgenden Einschränkungen gelten für Null-ETL-Integrationen von Aurora.

Themen

- Allgemeine Einschränkungen
- Einschränkungen von Aurora MySQL
- Einschränkungen bei Aurora PostgreSQL
- Einschränkungen für Amazon Redshift
- Amazon SageMaker AI Lakehouse-Einschränkungen

Allgemeine Einschränkungen

- Der Quell-DB-Cluster muss sich in derselben Region befinden wie das Ziel.
- Sie können einen DB-Cluster oder die zugehörigen Instances nicht umbenennen, wenn das Element über bestehende Integrationen verfügt.
- Sie können nicht mehrere Integrationen zwischen derselben Quell- und Zieldatenbank erstellen.
- Sie können einen DB-Cluster mit bestehenden Integrationen nicht löschen. Sie müssen zuerst alle zugehörigen Integrationen löschen.

- Wenn Sie den Quell-DB-Cluster anhalten, werden die letzten Transaktionen möglicherweise erst dann in das Ziel repliziert, wenn Sie den Cluster fortsetzen.
- Wenn der Cluster die Quelle einer Blau/Grün-Bereitstellung ist, können die blaue und die grüne Umgebung während der Umstellung keine vorhandenen Null-ETL-Integrationen haben. Sie müssen zuerst die Integration löschen und umstellen. Anschließend erstellen Sie die Integration neu.
- Ein DB-Cluster muss mindestens eine DB-Instance enthalten, um die Quelle einer Integration zu sein.
- Sie können keine Integration für einen Quell-DB-Cluster erstellen, bei dem es sich um einen kontoübergreifenden Klon handelt, wie z. B. solche, die mit AWS Resource Access Manager ()AWS RAM gemeinsam genutzt werden.
- Wenn der DB-Quell-Cluster in einer globalen Aurora-Datenbank verwendet wird und ein Failover zu einem der sekundären Cluster erfolgt, wird die Integration inaktiv. Sie müssen die Integration löschen und erneut erstellen.
- Sie können keine Integration für eine Quelldatenbank erstellen, für die aktiv eine andere Integration erstellt wird.
- Wenn Sie zum ersten Mal eine Integration erstellen oder wenn eine Tabelle erneut synchronisiert wird, kann das Seeding von Daten von der Quelle zum Ziel je nach Größe der Quelldatenbank 20 bis 25 Minuten oder länger dauern. Diese Verzögerung kann zu einer erhöhten Replikatverzögerung führen.
- Einige Datentypen werden nicht unterstützt. Weitere Informationen finden Sie unter the section called “Datentypunterschiede”.
- Systemtabellen, temporäre Tabellen und Ansichten werden nicht in Ziel-Warehouses repliziert.
- Das Ausführen von DDL-Befehlen (z. B. `ALTER TABLE`) für eine Quelltabelle kann eine erneute Synchronisierung der Tabelle auslösen, wobei die Tabelle während der erneuten Synchronisierung nicht mehr abgefragt werden kann. Weitere Informationen finden Sie unter the section called “Eine oder mehrere meiner Amazon-Redshift-Tabellen erfordern eine erneute Synchronisation”.

Einschränkungen von Aurora MySQL

- Auf Ihrem Quell-DB-Cluster muss eine unterstützte Version von Aurora MySQL ausgeführt werden. Eine Liste der unterstützten Versionen finden Sie unter the section called “Zero-ETL-Integrationen”.

- Null-ETL-Integrationen benötigen die MySQL-Binärprotokollierung (Binlog), um laufende Datenänderungen zu erfassen. Verwenden Sie keine Binlog-basierte Datenfilterung, da dies zu Dateninkonsistenzen zwischen der Quell- und der Zieldatenbank führen kann.
- Null-ETL-Integrationen werden nur für Datenbanken unterstützt, die für die Verwendung der InnoDB-Speicher-Engine konfiguriert sind.
- Fremdschlüsselverweise mit vordefinierten Tabellenaktualisierungen werden nicht unterstützt. Insbesondere werden `ON DELETE`- und `ON UPDATE`-Regeln mit `CASCADE`-, `SET NULL`- und `SET DEFAULT`-Aktionen nicht unterstützt. Der Versuch, eine Tabelle mit solchen Verweisen in einer anderen Tabelle zu erstellen oder zu aktualisieren, führt zu einem Fehlschlag der Tabelle.
- XA-Transaktionen, die auf dem Quell-DB-Cluster ausgeführt werden, führen dazu, dass die Integration in den Status `Syncing` wechselt.

Einschränkungen bei Aurora PostgreSQL

- Auf Ihrem Quell-DB-Cluster muss eine unterstützte Version von Aurora PostgreSQL ausgeführt werden. Eine Liste der unterstützten Versionen finden Sie unter the section called “Zero-ETL-Integrationen ”.
- Wenn Sie einen Quell-DB-Cluster von Aurora PostgreSQL auswählen, müssen Sie mindestens ein Datenfiltermuster angeben. Das Muster muss mindestens eine einzelne Datenbank (*`database-name`*`.*.*`) für die Replikation in das Ziel-Warehouse enthalten. Weitere Informationen finden Sie unter the section called “Datenfilterung für Null-ETL-Integrationen”.
- Alle Datenbanken, die im DB-Cluster von Aurora-PostgreSQL erstellt wurden, müssen UTF-8-Kodierung verwenden.
- Bei Verwendung der deklarativen Partitionierung werden die Tabellenpartitionen in Amazon Redshift repliziert. Die partitionierte Tabelle selbst wird jedoch nicht in Amazon Redshift repliziert.
- Zweiphasige Transaktionen werden nicht unterstützt.
- Wenn Sie alle DB-Instances aus einem DB-Cluster löschen, der die Quelle einer Integration ist, und dann erneut eine DB-Instance hinzufügen, wird die Replikation zwischen dem Quell- und dem Ziel-Cluster unterbrochen.
- Der Quell-DB-Cluster kann Aurora Limitless Database nicht verwenden.
- Primärschlüssel sind für alle im Datenfilter vorhandenen Tabellen erforderlich. Alle Tabellen ohne Primärschlüssel werden in den Status „failed“ versetzt.

Einschränkungen für Amazon Redshift

Eine Liste der Einschränkungen von Amazon Redshift im Zusammenhang mit Null-ETL-Integrationen finden Sie unter Überlegungen bei der Verwendung von Null-ETL-Integrationen mit Amazon Redshift im Amazon-Redshift-Managementleitfaden.

Amazon SageMaker AI Lakehouse-Einschränkungen

Im Folgenden finden Sie eine Einschränkung für Amazon SageMaker AI Lakehouse Zero-ETL-Integrationen.

- Katalognamen sind auf von 64 Zeichen beschränkt.

Kontingente

Für Ihr Konto gelten die folgenden Kontingente in Bezug auf Null-ETL-Integrationen von Aurora. Jedes Kontingent gilt pro Region, sofern nicht anders angegeben.

Name	Standard	Description
Integrationen	100	Die Gesamtzahl der Integrationen innerhalb eines AWS-Konto.
Integrationen pro Ziel	50	Die Anzahl der Integrationen, die Daten an ein einzelnes Ziel-Data-Warehouse oder -Lakehouse senden.
Integrationen pro Quell-Cluster	5	Die Anzahl der Integrationen, die Daten von einem einzelnen DB-Cluster senden.

Darüber hinaus legt das Ziel-Warehouse bestimmte Einschränkungen für die Anzahl der zulässigen Tabellen in jeder DB-Instance oder jedem Cluster-Knoten fest. Weitere Informationen finden Sie unter Kontingente und Limits in Amazon Redshift im Amazon-Redshift-Managementleitfaden.

Unterstützte Regionen

Aurora Zero-ETL-Integrationen sind in einer Teilmenge von verfügbar. AWS-Regionen Eine Liste der unterstützten -Regionen finden Sie unter the section called “Zero-ETL-Integrationen ”.

Erste Schritte mit Null-ETL-Integrationen von Aurora

Bevor Sie eine Null-ETL-Integration erstellen, konfigurieren Sie Ihren Aurora-DB-Cluster und Ihr Data Warehouse mit den erforderlichen Parametern und Berechtigungen. Während der Einrichtung führen Sie die folgenden Schritte aus:

1. Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe.
2. Erstellen Sie einen Quell-DB-Cluster.
3. Erstellen Sie ein Ziel-Data Warehouse für Amazon Redshift oder Erstellen Sie ein Amazon SageMaker AI-Ziel-Lakehouse.

Wenn Sie diese Aufgaben abgeschlossen haben, fahren Sie mit the section called “Erstellen von Null-ETL-Integrationen mit Amazon Redshift” oder the section called “Erstellen von Null-ETL-Integrationen mit einem Amazon SageMaker-Lakehouse” fort.

Sie können das verwenden AWS SDKs , um den Einrichtungsprozess für Sie zu automatisieren. Weitere Informationen finden Sie unter the section called “Richten Sie eine Integration mit dem ein AWS SDKs”.

> **Tip**
>
> Sie können RDS diese Einrichtungsschritte für Sie ausführen lassen, während Sie die Integration erstellen, anstatt sie manuell durchzuführen. Um sofort mit der Erstellung einer Integration zu beginnen, siehe the section called “Erstellen von Null-ETL-Integrationen mit Amazon Redshift”.

In Schritt 3 können Sie je nach Ihren Anforderungen entweder ein Ziel-Data-Warehouse (Schritt 3a) oder ein Ziel-Lakehouse (Schritt 3b) erstellen:

- Entscheiden Sie sich für ein Data Warehouse, wenn Sie herkömmliche Data-Warehousing-Funktionen mit SQL-basierten Analysen benötigen.
- Entscheiden Sie sich für ein Amazon SageMaker AI Lakehouse, wenn Sie Funktionen für maschinelles Lernen benötigen und Lakehouse-Funktionen für Data-Science- und ML-Workflows verwenden möchten.

Schritt 1: Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe

Null-ETL-Integrationen von Aurora erfordern bestimmte Werte für die DB-Cluster-Parameter, die die Replikation steuern. Ganz konkret erfordert Aurora MySQL ein erweitertes Binärprotokoll (`aurora_enhanced_binlog`), Aurora PostgreSQL erfordert eine erweiterte logische Replikation (`aurora.enhanced_logical_replication`).

Um die Binärprotokollierung oder logische Replikation zu konfigurieren, müssen Sie zunächst eine benutzerdefinierte DB-Cluster-Parametergruppe erstellen und sie dann dem Quell-DB-Cluster zuordnen.

Aurora MySQL (aurora-mysql8.0-Familie):

- `aurora_enhanced_binlog=1`
- `binlog_backup=0`
- `binlog_format=ROW`
- `binlog_replication_globaldb=0`
- `binlog_row_image=full`
- `binlog_row_metadata=full`

Stellen Sie außerdem sicher, dass der `binlog_transaction_compression`-Parameter nicht auf `ON` und der `binlog_row_value_options`-Parameter nicht auf `PARTIAL_JSON` gesetzt ist.

Weitere Informationen über das erweiterte Aurora-MySQL-Binärprotokoll finden Sie unter the section called “Einrichten eines erweiterten Binärprotokolls”.

Aurora PostgreSQL (aurora-postgresql16-Familie):

- `rds.logical_replication=1`
- `aurora.enhanced_logical_replication=1`
- `aurora.logical_replication_backup=0`
- `aurora.logical_replication_globaldb=0`

Wenn Sie die erweiterte logische Replikation (`aurora.enhanced_logical_replication`) aktivieren, werden immer alle Spaltenwerte in das Write-Ahead-Protokoll (WAL) geschrieben, auch wenn `REPLICA IDENTITY FULL` nicht aktiviert ist. Dies kann die IOPS für Ihren Quell-DB-Cluster erhöhen.

> **Important**
>
> Wenn Sie den DB-Cluster-Parameter `aurora.enhanced_logical_replication` aktivieren oder deaktivieren, macht die primäre DB-Instance alle logischen Replikationsslots ungültig. Dadurch wird die Replikation von der Quelle in das Ziel angehalten und Sie müssen die Replikationsslots auf der primären DB-Instance neu erstellen. Um Unterbrechungen zu vermeiden, sollte der Parameterstatus während der Replikation gleich bleiben.

Schritt 2: Wählen oder Erstellen eines Quell-DB-Clusters

Nachdem Sie eine benutzerdefinierte DB-Cluster-Parametergruppe erstellt haben, wählen oder erstellen Sie einen Aurora-DB-Cluster. Dieser Cluster dient als Quelle der Datenreplikation in das Ziel-Data-Warehouse. Sie können einen DB-Cluster angeben, der bereitgestellte DB-Instances oder DB-Instances von Aurora Serverless v2 als Quelle verwendet. Anleitungen zum Erstellen eines DB-Clusters finden Sie unter the section called “Erstellen eines DB-Clusters” oder the section called “Erstellen eines Aurora Serverless v2-DB Clusters”.

In der Datenbank muss eine unterstützte DB-Engine-Version ausgeführt werden. Eine Liste der unterstützten Versionen finden Sie unter the section called “Zero-ETL-Integrationen ”.

Wenn Sie die Datenbank erstellen, ändern Sie unter Zusätzliche Konfiguration die Standard-DB-Cluster-Parametergruppe in die benutzerdefinierte Parametergruppe, die Sie im vorherigen Schritt erstellt haben.

> **Note**
>
> Wenn Sie die Parametergruppe mit dem DB-Cluster verknüpfen, nachdem der Cluster bereits erstellt wurde, müssen Sie die primäre DB-Instance im Cluster neu starten, damit die Änderungen angewendet werden, bevor Sie eine Null-ETL-Integration erstellen können. Anleitungen finden Sie unter the section called “Neustart eines Aurora DB-Clusters oder einer Instance”.

Schritt 3a: Erstellen eines Ziel-Data-Warehouse

Nachdem Sie den Quell-DB-Cluster erstellt haben, müssen Sie ein Ziel-Data-Warehouse erstellen und konfigurieren. Das Data Warehouse muss die folgenden Anforderungen erfüllen:

- Verwendung eines RA3 Knotentyps mit mindestens zwei Knoten oder Redshift Serverless.
- Es muss verschlüsselt sein (bei Verwendung eines bereitgestellten Clusters). Weitere Informationen finden Sie unter Datenbankverschlüsselung in Amazon Redshift.

Anweisungen zum Erstellen eines Data Warehouse finden Sie unter Erstellen eines Clusters für bereitgestellte Cluster oder Erstellen einer Arbeitsgruppe mit einem Namespace für Redshift Serverless.

Aktivieren Sie die Berücksichtigung von Groß- und Kleinschreibung im Data Warehouse

Damit die Integration erfolgreich ist, muss der Parameter für die Berücksichtigung von Groß- und Kleinschreibung (`enable_case_sensitive_identifier`) für das Data Warehouse aktiviert sein. Standardmäßig ist die Berücksichtigung von Groß- und Kleinschreibung auf allen bereitgestellten Clustern und Redshift-Serverless-Arbeitsgruppen deaktiviert.

Um die Berücksichtigung von Groß- und Kleinschreibung zu aktivieren, führen Sie je nach Data-Warehouse-Typ die folgenden Schritte aus:

- Bereitgestellter Cluster – Um die Berücksichtigung von Groß- und Kleinschreibung in einem bereitgestellten Cluster zu aktivieren, erstellen Sie eine benutzerdefinierte Parametergruppe mit aktiviertem `enable_case_sensitive_identifier`-Parameter. Ordnen Sie diese Parametergruppe dann dem Cluster zu. Anweisungen finden Sie unter Verwalten von Parametergruppen mit der Konsole oder Konfigurieren von Parameterwerten mit der AWS CLI.

  > **Note**
  >
  > Denken Sie daran, den Cluster neu zu starten, nachdem Sie ihm die benutzerdefinierte Parametergruppe zugeordnet haben.

- Serverless-Arbeitsgruppe – Um die Berücksichtigung von Groß- und Kleinschreibung in einer Redshift-Serverless-Arbeitsgruppe zu aktivieren, müssen Sie die AWS CLI verwenden. Die Amazon-Redshift-Konsole unterstützt derzeit nicht das Ändern von Redshift-Serverless-Parameterwerten. Senden Sie die folgende update-workgroup-Anfrage:

```
aws redshift-serverless update-workgroup \
  --workgroup-name target-workgroup \
```

```
 --config-parameters
parameterKey=enable_case_sensitive_identifier,parameterValue=true
```

Sie müssen eine Arbeitsgruppe nicht neu starten, nachdem Sie ihre Parameterwerte geändert haben.

Konfigurieren der Autorisierung für das Data Warehouse

Nachdem Sie ein Data Warehouse erstellt haben, müssen Sie den Quell-DB-Cluster von Aurora als autorisierte Integrationsquelle konfigurieren. Anleitungen finden Sie unter Konfigurieren der Autorisierung für Ihr Amazon-Redshift-Data-Warehouse.

Richten Sie eine Integration mit dem ein AWS SDKs

Anstatt jede Ressource manuell einzurichten, können Sie das folgende Python-Skript ausführen, um die erforderlichen Ressourcen automatisch einzurichten. Das Codebeispiel verwendet das AWS SDK for Python (Boto3), um einen Quell-DB-Cluster von Amazon Aurora und ein Ziel-Data-Warehouse zu erstellen, jeweils mit den erforderlichen Parameterwerten. Dann wird abgewartet, bis die Datenbanken verfügbar sind, bevor eine Null-ETL-Integration zwischen ihnen erstellt wird. Sie können verschiedene Funktionen auskommentieren, je nachdem, welche Ressourcen Sie einrichten müssen.

Führen Sie die folgenden Befehle aus, um die erforderlichen Abhängigkeiten zu installieren:

```
pip install boto3
pip install time
```

Innerhalb des Skripts können Sie optional die Namen der Quell-, Ziel- und Parametergruppen ändern. Die letzte Funktion erstellt eine Integration namens `my-integration` nach der Einrichtung der Ressourcen.

Python-Codebeispiel

Aurora MySQL

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
```

```
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora MySQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-mysql8.0',
        Description='For Aurora MySQL binary logging'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'aurora_enhanced_binlog',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_replication_globaldb',
```

```
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_metadata',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-mysql',
        EngineVersion='8.0.mysql_aurora.3.05.2',
        DatabaseName='myauroradb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']
['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-mysql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
```

```
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
```

```
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']
```

```
    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

if __name__ == "__main__":
    main()
```

Aurora PostgreSQL

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
```

```
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora PostgreSQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-postgresql16',
        Description='For Aurora PostgreSQL logical replication'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'rds.logical_replication',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.enhanced_logical_replication',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.logical_replication_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'aurora.logical_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])
```

```
    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-postgresql',
        EngineVersion='16.4.aurora-postgresql',
        DatabaseName='mypostgresdb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']
['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-postgresql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora PostgreSQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])
```

```
response = redshift.create_cluster(
    ClusterIdentifier=target_cluster_name,
    NodeType='ra3.4xlarge',
    NumberOfNodes=2,
    Encrypted=True,
    MasterUsername='username',
    MasterUserPassword='Password01**',
    ClusterParameterGroupName=target_param_group_name
)
print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

# Retrieve the target cluster ARN
response = redshift.describe_clusters(
    ClusterIdentifier=target_cluster_name
)
target_arn = response['Clusters'][0]['ClusterNamespaceArn']

# Retrieve the current user's account ID
response = sts.get_caller_identity()
account_id = response['Account']

# Create a resource policy specifying cluster ARN and account ID
response = redshift.put_resource_policy(
    ResourceArn=target_arn,
    Policy='''
    {
        \"Version\":\"2012-10-17\",
        \"Statement\":[
            {\"Effect\":\"Allow\",
            \"Principal\":{
                \"Service\":\"redshift.amazonaws.com\"
            },
            \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
            \"Condition\":{
                \"StringEquals\":{
                    \"aws:SourceArn\":\"%s\"}
                }
            },
            {\"Effect\":\"Allow\",
            \"Principal\":{
                \"AWS\":\"arn:aws:iam::%s:root\"},
            \"Action\":\"redshift:CreateInboundIntegration\"}
        ]
```

```
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance'
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
```

```
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

if __name__ == "__main__":
    main()
```

Schritt 3b: Erstellen Sie einen AWS Glue Katalog für die Amazon SageMaker AI Zero-ETL-Integration

Wenn Sie eine Zero-ETL-Integration mit einem Amazon SageMaker AI Lakehouse erstellen, müssen Sie einen AWS Glue verwalteten Katalog in erstellen. AWS Lake Formation Der Zielkatalog muss ein von Amazon Redshift verwalteter Katalog sein. Um einen von Amazon Redshift verwalteten Katalog zu erstellen, erstellen Sie zunächst die serviceverknüpfte Rolle `AWSServiceRoleForRedshift`. Fügen Sie in der Lake-Formation-Konsole `AWSServiceRoleForRedshift` als Administrator mit Leseberechtigung hinzu.

Weitere Informationen zu den oben genannten Aufgaben finden Sie in den folgenden Themen.

- Informationen zum Erstellen eines von Amazon Redshift verwalteten Katalogs finden Sie unter Creating an Amazon Redshift managed catalog in the AWS Glue Data Catalog im AWS Lake Formation -Entwicklerhandbuch.
- Informationen zur serviceverknüpften Rolle für Amazon Redshift finden Sie unter Verwenden serviceverknüpfter Rollen für Amazon Redshift im Amazon-Redshift-Verwaltungshandbuch.
- Informationen zu Leseberechtigungen für Administratoren für Lake Formation finden Sie unter Lake Formation personas and IAM permissions reference im AWS Lake Formation -Entwicklerhandbuch.

Konfigurieren Sie die Berechtigungen für den Zielkatalog AWS Glue

Bevor Sie einen Zielkatalog für die Zero-ETL-Integration erstellen, müssen Sie die Lake Formation Formation-Zielerstellungsrolle und die AWS Glue Datenübertragungsrolle erstellen. Verwenden Sie die Lake-Formation-Zielerstellungsrolle, um den Zielkatalog zu erstellen. Geben Sie bei der Erstellung

des Zielkatalogs die Glue-Datenübertragungsrolle in das Feld IAM-Rolle im Abschnitt „Zugriff von Engines“ ein.

Lake-Formation-Zielerstellungsrolle

Die Zielerstellungsrolle muss ein Lake-Formation-Administrator sein. Sie erfordert die folgenden Berechtigungen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "lakeformation:RegisterResource",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "s3:PutEncryptionConfiguration",
                "iam:PassRole",
                "glue:CreateCatalog",
                "glue:GetCatalog",
                "s3:PutBucketTagging",
                "s3:PutLifecycleConfiguration",
                "s3:PutBucketPolicy",
                "s3:CreateBucket",
                "redshift-serverless:CreateNamespace",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "redshift-serverless:CreateWorkgroup"
            ],
            "Resource": [
                "arn:aws:glue:*:111122223333:catalog",
                "arn:aws:glue:*:111122223333:catalog/*",
                "arn:aws:s3:::*",
                "arn:aws:redshift-serverless:*:111122223333:workgroup/*",
                "arn:aws:redshift-serverless:*:111122223333:namespace/*",
                "arn:aws:iam::111122223333:role/GlueDataCatalogDataTransferRole"
```

```
            ]
        }
    ]
}
```

Die Zielerstellungsrolle muss die folgende Vertrauensstellung aufweisen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
          "Effect": "Allow",
          "Principal": {
            "AWS": "arn:aws:iam::111122223333:user/Username"
          },
          "Action": "sts:AssumeRole"
        }
    ]
}
```

Glue-Datenübertragungsrolle

Die Glue-Datenübertragungsrolle ist für MySQL-Katalogoperationen erforderlich und muss über die folgenden Berechtigungen verfügen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "DataTransferRolePolicy",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "glue:GetCatalog",
                "glue:GetDatabase"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Die Glue-Datenübertragungsrolle muss die folgende Vertrauensstellung aufweisen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "glue.amazonaws.com",
                    "redshift.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Nächste Schritte

Mit einem Aurora-DB-Cluster für die und entweder einem Amazon Redshift Redshift-Ziel-Data Warehouse oder einem Amazon SageMaker AI Lakehouse können Sie eine Zero-ETL-Integration

erstellen und Daten replizieren. Detaillierte Anweisungen finden Sie unter the section called “Erstellen von Null-ETL-Integrationen mit Amazon Redshift”.

Erstellen von Null-ETL-Integrationen von Aurora mit Amazon Redshift

Wenn Sie eine Null-ETL-Integration von Aurora erstellen, geben Sie den Quell-DB-Cluster von Aurora und das Ziel-Data-Warehouse von Amazon Redshift an. Sie können auch die Verschlüsselungseinstellungen anpassen und Tags hinzufügen. Aurora erstellt eine Integration zwischen dem Quell-DB-Cluster und dem jeweiligen Ziel. Sobald die Integration aktiv ist, werden alle Daten, die Sie in den Quell-DB-Cluster einfügen, in das konfigurierte Amazon-Redshift-Ziel repliziert.

Voraussetzungen

Bevor Sie eine Null-ETL-Integration, müssen Sie einen Quell-DB-Cluster und ein Ziel-Data-Warehouse von Amazon Redshift erstellen. Außerdem müssen Sie die Replikation in das Data Warehouse zulassen, indem Sie den DB-Cluster als autorisierte Integrationsquelle hinzufügen.

Anleitungen zum Ausführen der einzelnen Schritte finden Sie unter the section called “Erste Schritte mit Null-ETL-Integrationen”.

Erforderliche Berechtigungen

Bestimmte IAM-Berechtigungen sind erforderlich, um eine Null-ETL-Integration zu erstellen. Sie benötigen mindestens die Berechtigungen, um die folgenden Aktionen durchführen zu können:

- Erstellen Sie Null-ETL-Integrationen für den Quell-DB-Cluster.
- Anzeigen und Löschen aller Null-ETL-Integrationen.
- Erstellen eingehender Integrationen in das Ziel-Data-Warehouse.

Die folgende Beispielrichtlinie zeigt die die geringsten Berechtigungen, die zum Erstellen und Verwalten von Integrationen erforderlich sind. Möglicherweise benötigen Sie nicht genau diese Berechtigungen, wenn Ihr Benutzer oder Ihre Rolle über umfassendere Berechtigungen verfügt, z. B. über eine von `AdministratorAccess` verwaltete Richtlinie.

> **Note**
>
> Redshift Amazon Resource Names (ARNs) haben das folgende Format. Beachten Sie die Verwendung eines Schrägstrichs (`/`) anstelle eines Doppelpunkts (`:`) vor der Serverless-Namespace-UUID.
>
> - Bereitgestellter Cluster – `arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid`
> - Serverless – `arn:aws:redshift-serverless:{region}:{account-id}:namespace/namespace-uuid`

Beispielrichtlinie für Redshift-Ziel

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CreateIntegration",
      "Effect": "Allow",
      "Action": [
        "rds:CreateIntegration"
      ],
      "Resource": [
        "arn:aws:rds:us-east-1:123456789012:db:source-db",
        "arn:aws:rds:us-east-1:123456789012:integration:*"
      ]
    },
    {
      "Sid": "DescribeIntegrationDetails",
      "Effect": "Allow",
      "Action": [
        "rds:DescribeIntegrations"
      ],
      "Resource": [
      "arn:aws:rds:us-east-1:123456789012:integration:*"
  ]
    },
```

```
    {
      "Sid": "ChangeIntegrationDetails",
      "Effect": "Allow",
      "Action": [
        "rds:DeleteIntegration",
        "rds:ModifyIntegration"
      ],
      "Resource": [
        "arn:aws:rds:us-east-1:123456789012:integration:*"
      ]
    },
    {
      "Sid": "AllowRedShiftIntegration",
      "Effect": "Allow",
      "Action": [
        "redshift:CreateInboundIntegration"
      ],
      "Resource": [
        "arn:aws:redshift:us-east-1:123456789012:namespace:namespace-uuid"
      ]
    }
  ]
}
```

Auswählen eines Ziel-Data-Warehouse in einem anderen Konto

Wenn Sie ein Amazon Redshift Redshift-Ziel-Data Warehouse angeben möchten, das sich in einem anderen befindet AWS-Konto, müssen Sie eine Rolle erstellen, die es Benutzern des aktuellen Kontos ermöglicht, auf Ressourcen im Zielkonto zuzugreifen. Weitere Informationen finden Sie unter Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , dem Sie gehören.

Die Rolle muss über die folgenden Berechtigungen verfügen, die es dem Benutzer ermöglichen, verfügbare von Amazon Redshift bereitgestellte Cluster und Redshift-Serverless-Namespaces im Zielkonto einzusehen.

Erforderliche Berechtigungen und Vertrauensrichtlinie

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Die Rolle muss über die folgende Vertrauensrichtlinie verfügen, die die Zielkonto-ID angibt.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Anleitungen zum Erstellen der Rolle finden Sie unter Erstellen einer Rolle mit benutzerdefinierten Vertrauensrichtlinien.

Erstellen von Null-ETL-Integrationen

Sie können eine Zero-ETL-Integration mithilfe der AWS-Managementkonsole, der oder der AWS CLI RDS-API erstellen.

> **Important**
>
> Null-ETL-Integrationen unterstützen keine Aktualisierungs- oder Resynchronisierungsvorgänge. Wenn Sie nach der Erstellung auf Probleme mit einer Integration stoßen, müssen Sie die Integration löschen und eine neue erstellen.

RDS-Konsole

So erstellen Sie eine Null-ETL-Integration

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Null-ETL-Integrationen aus.
3. Wählen Sie Null-ETL-Integration erstellen aus.
4. Geben Sie für Integrationskennung einen Namen für die Integration ein. Der Name kann bis zu 63 alphanumerische Zeichen umfassen und Bindestriche enthalten.

   > **Important**
   >
   > Katalognamen sind auf von 64 Zeichen beschränkt. Stellen Sie sicher, dass die Integrations-ID diese Anforderung erfüllt, wenn sie als Katalogname verwendet werden soll.

5. Wählen Sie Weiter aus.
6. Wählen Sie als Quelle des Ursprungs der Daten den Aurora DB-Cluster aus.

   > **Note**
   >
   > RDS benachrichtigt Sie, wenn die DB-Cluster-Parameter nicht korrekt konfiguriert sind. Wenn Sie diese Nachricht erhalten, können Sie entweder Reparieren wählen oder eine manuelle Konfiguration vornehmen. Anleitungen zur manuellen Behebung finden Sie unter the section called “Schritt 1: Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe”.
   > Das Ändern der DB-Cluster-Parameter erfordert einen Neustart. Bevor Sie die Integration erstellen können, müssen der Neustart abgeschlossen und die neuen Parameterwerte erfolgreich auf den Cluster angewendet werden.

7. (Optional) Wählen Sie Datenfilteroptionen anpassen aus und fügen Sie Ihrer Integration Datenfilter hinzu. Sie können Datenfilter verwenden, um den Umfang der Replikation in das Ziel-Data-Warehouse zu definieren. Weitere Informationen finden Sie unter the section called “Datenfilterung für Null-ETL-Integrationen”.
8. Sobald der Quell-DB-Cluster erfolgreich konfiguriert wurde, wählen Sie Weiter aus.
9. Gehen Sie bei Ziel wie folgt vor:

    1. (Optional) Um ein anderes AWS-Konto für das Amazon Redshift Redshift-Ziel zu verwenden, wählen Sie Anderes Konto angeben aus. Geben Sie dann den Namen einer IAM-Rolle mit Berechtigungen zur Anzeige Ihrer Data Warehouses ein. Anleitungen zum Erstellen der IAM-Rolle finden Sie unter the section called “Auswählen eines Ziel-Data-Warehouse in einem anderen Konto”.
    2. Wählen Sie für Amazon Redshift Data Warehouse das Ziel für replizierte Daten aus dem Quell-DB-Cluster aus. Sie können einen bereitgestellten Amazon-Redshift-Cluster oder einen Redshift-Serverless-Namespace als Ziel auswählen.

    > **Note**
    >
    > RDS benachrichtigt Sie, wenn die Ressourcenrichtlinie oder die Einstellungen zur Berücksichtigung der Groß- und Kleinschreibung für das angegebene Data Warehouse nicht korrekt konfiguriert sind. Wenn Sie diese Nachricht erhalten, können Sie entweder Reparieren wählen oder eine manuelle Konfiguration vornehmen. Anweisungen zur manuellen Behebung finden Sie unter Aktivieren der Groß- und Kleinschreibung für Ihr Data Warehouse und Konfigurieren der Autorisierung für Ihr Data Warehouse im Amazon Redshift Management Guide.
    > Das Ändern der Groß- und Kleinschreibung für einen bereitgestellten Redshift-Cluster erfordert einen Neustart. Bevor Sie die Integration erstellen können, muss der Neustart abgeschlossen und der neue Parameterwert erfolgreich auf den Cluster angewendet werden.
    > Wenn sich Ihre gewählte Quelle und Ihr Ziel in verschiedenen AWS-Konten befinden, kann Amazon RDS diese Einstellungen nicht für Sie korrigieren. Sie müssen zu dem anderen Konto navigieren und diese manuell in Amazon Redshift korrigieren.

10. Sobald Ihr Ziel-Data Warehouse korrekt konfiguriert ist, wählen Sie Weiter.
11. (Optional) Fügen Sie unter Tags ein oder mehrere Tags zu der Integration hinzu. Weitere Informationen finden Sie unter the section called “Taggen von Aurora- und RDS-Ressourcen”.

12. Geben Sie für Verschlüsselung an, wie Ihre Integration verschlüsselt werden soll. Standardmäßig verschlüsselt RDS alle Integrationen mit einem. AWS-eigener Schlüssel Um stattdessen einen vom Kunden verwalteten Schlüssel auszuwählen, aktivieren Sie die Option Verschlüsselungseinstellungen anpassen und wählen Sie einen KMS-Schlüssel aus, der für die Verschlüsselung verwendet werden soll. Weitere Informationen finden Sie unter the section called “Verschlüsseln von Amazon Aurora-Ressourcen”.

    Fügen Sie optional einen Verschlüsselungskontext hinzu. Weitere Informationen finden Sie unter Verschlüsselungskontext im AWS Key Management Service -Entwicklerhandbuch.

    > **Note**
    >
    > Amazon RDS fügt zusätzlich zu den von Ihnen hinzugefügten Paaren die folgenden Verschlüsselungskontextpaare hinzu:
    >
    > - `aws:redshift:integration:arn` - `IntegrationArn`
    > - `aws:servicename:id` - `Redshift`
    >
    > Dadurch wird die Gesamtzahl der Paare, die Sie hinzufügen können, von 8 auf 6 reduziert, was zur Gesamtzeichenbegrenzung der Erteilungsbeschränkung beiträgt. Weitere Informationen finden Sie unter Verwenden von Erteilungsbeschränkungen im Entwicklerhandbuch für AWS Key Management Service .

13. Wählen Sie Weiter aus.
14. Überprüfen Sie Ihre Integrationseinstellungen und wählen Sie Null-ETL-Integration erstellen aus.

    Wenn die Erstellung fehlschlägt, finden Sie Informationen zur Fehlerbehebung unter the section called “Ich kann keine Null-ETL-Integration erstellen”.

Der Status der Integration lautet während der Erstellung `Creating`. Das Ziel-Data-Warehouse von Amazon Redshift hat den Status `Modifying`. Während dieser Zeit können Sie das Data Warehouse nicht abfragen und keine Konfigurationsänderungen daran vornehmen.

Wenn die Integration erfolgreich erstellt wurde, ändern sich sowohl der Status der Integration als auch der Status des Ziel-Data-Warehouse von Amazon Redshift in `Active`.

AWS CLI

Um eine Zero-ETL-Integration mit dem zu erstellen AWS CLI, verwenden Sie den Befehl create-integration mit den folgenden Optionen:

> **Note**
>
> Denken Sie daran, dass Katalognamen auf 19 Zeichen beschränkt sind. Wählen Sie Ihren Integrationsnamen entsprechend aus, wenn er als Katalogname verwendet werden soll.

- `--integration-name` – Geben Sie einen Namen für die Integration an.
- `--source-arn` – Geben Sie als Quelle für die Integration den ARN des Aurora-DB-Clusters an.
- `--target-arn` – Geben Sie den ARN des Amazon Redshift Data Warehouse an, das das Ziel für die Integration sein soll.

Example

Für Linux, macOS oder Unix:

```
aws rds create-integration \
    --integration-name my-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

Für Windows:

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

RDS-API

Verwenden Sie die `CreateIntegration`-Operation mit den folgenden Parametern, um mithilfe der Amazon-RDS-API eine Null-ETL-Integration zu erstellen:

> **Note**
>
> Katalognamen sind auf 19 Zeichen beschränkt. Stellen Sie sicher, dass Ihr IntegrationName Parameter diese Anforderung erfüllt, wenn er als Katalogname verwendet werden soll.

- `IntegrationName` – Geben Sie einen Namen für die Integration an.
- `SourceArn` – Geben Sie als Quelle für die Integration den ARN des Aurora-DB-Clusters an.
- `TargetArn` – Geben Sie den ARN des Amazon Redshift Data Warehouse an, das das Ziel für die Integration sein soll.

Verschlüsseln von Integrationen mit einem vom Kunden verwalteten Schlüssel

Wenn Sie AWS-eigener Schlüssel beim Erstellen einer Integration einen benutzerdefinierten KMS-Schlüssel anstelle eines angeben, muss die Schlüsselrichtlinie dem Amazon Redshift Redshift-Serviceprinzipal Zugriff auf die `CreateGrant` Aktion gewähren. Darüber hinaus muss es dem aktuellen Benutzer die Ausführung der Aktionen `DescribeKey` und `CreateGrant` ermöglicht werden.

Die folgende Beispielrichtlinie demonstriert, wie die erforderlichen Berechtigungen in der Schlüsselrichtlinie bereitgestellt werden. Sie enthält Kontextschlüssel, um den Umfang der Berechtigungen weiter zu reduzieren.

Beispiel für eine Schlüsselrichtlinie

JSON

```
{
    "Version":"2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
```

```
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Redshift service principal to add a grant to a KMS
key",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}": "{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}": "{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
```

```
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about
 a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

Weitere Informationen finden Sie unter Erstellen einer Schlüsselrichtlinie im AWS Key Management Service -Entwicklerhandbuch.

Nächste Schritte

Nachdem Sie erfolgreich eine Null-ETL-Integration erstellt haben, müssen Sie eine Zieldatenbank in Ihrem Amazon-Redshift-Zielcluster oder Ihrer Zielarbeitsgruppe erstellen. Anschließend können Sie damit beginnen, Daten zum Quell-DB-Cluster von Aurora hinzuzufügen und in Amazon Redshift abzufragen. Anleitungen finden Sie unter Erstellen von Zieldatenbanken in Amazon Redshift.

Erstellen von Null-ETL-Integrationen von Aurora mit einem Amazon SageMaker-Lakehouse

Wenn Sie eine Aurora Zero-ETL-Integration mit einem Amazon SageMaker Lakehouse erstellen, geben Sie den Aurora-DB-Cluster der und den verwalteten AWS Glue Zielkatalog an. Sie können auch die Verschlüsselungseinstellungen anpassen und Tags hinzufügen. Aurora erstellt eine Integration zwischen dem Quell-DB-Cluster und dem jeweiligen Ziel. Sobald die Integration aktiv ist, werden alle Daten, die Sie in den Quell-DB-Cluster einfügen, in das konfigurierte Ziel repliziert.

Voraussetzungen

Bevor Sie eine Zero-ETL-Integration mit einem Amazon SageMaker Lakehouse erstellen, müssen Sie einen erstellen. AWS Glue Außerdem müssen Sie die Replikation in den Katalog zulassen, indem Sie den DB-Cluster als autorisierte Integrationsquelle hinzufügen.

Anleitungen zum Ausführen der einzelnen Schritte finden Sie unter the section called “Erste Schritte mit Null-ETL-Integrationen”.

Erforderliche Berechtigungen

Bestimmte IAM-Berechtigungen sind erforderlich, um eine Null-ETL-Integration mit einem Amazon SageMaker-Lakehouse zu erstellen. Sie benötigen mindestens die Berechtigungen, um die folgenden Aktionen durchführen zu können:

- Erstellen Sie Null-ETL-Integrationen für den Quell-DB-Cluster.
- Anzeigen und Löschen aller Null-ETL-Integrationen.
- Erstellen Sie eingehende Integrationen in den verwalteten Zielkatalog. AWS Glue
- Greifen Sie auf Amazon S3 S3-Buckets zu, die vom AWS Glue verwalteten Katalog verwendet werden.
- Verwenden Sie AWS KMS Schlüssel für die Verschlüsselung, wenn eine benutzerdefinierte Verschlüsselung konfiguriert ist.
- Registrieren Sie Ressourcen bei Lake Formation.
- Fügen Sie dem AWS Glue verwalteten Katalog eine Ressourcenrichtlinie hinzu, um eingehende Integrationen zu autorisieren.

Die folgende Beispielrichtlinie zeigt die geringsten Berechtigungen, die zum Erstellen und Verwalten von Integrationen in ein Amazon SageMaker-Lakehouse erforderlich sind. Möglicherweise benötigen Sie nicht genau diese Berechtigungen, wenn Ihr Benutzer oder Ihre Rolle über umfassendere Berechtigungen verfügt, z. B. über eine von `AdministratorAccess` verwaltete Richtlinie.

Darüber hinaus müssen Sie eine Ressourcenrichtlinie für den AWS Glue verwalteten Zielkatalog konfigurieren, um eingehende Integrationen zu autorisieren. Verwenden Sie den folgenden AWS CLI Befehl, um die Ressourcenrichtlinie anzuwenden.

AWS CLI Beispielbefehl zur Autorisierung eingehender Integrationen im Zielkatalog

```
aws glue put-resource-policy \
```

```
      --policy-in-json  '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "glue.amazonaws.com"
        },
        "Action": [
            "glue:AuthorizeInboundIntegration"
        ],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"],
        "Condition": {
            "StringEquals": {
                "aws:SourceArn": "arn:aws:rds:region:account_id:db:source_name"
            }
        }
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "account_id"
        },
        "Action": ["glue:CreateInboundIntegration"],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"]
    }
    ]
}' \
      --region region
```

> **Note**
>
> Amazon Resource Names (ARNs) für den Glue-Katalog haben das folgende Format:
>
> - Glue-Katalog – `arn:aws:glue:{region}:{account-id}:catalog/catalog-name`

Auswahl eines AWS Glue verwalteten Zielkatalogs in einem anderen Konto

Wenn Sie einen AWS Glue verwalteten Zielkatalog angeben möchten, der sich in einem anderen AWS-Konto befindet, müssen Sie eine Rolle erstellen, die es Benutzern im aktuellen Konto ermöglicht, auf Ressourcen im Zielkonto zuzugreifen. Weitere Informationen finden Sie unter Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , dem Sie gehören.

Die Rolle muss über die folgenden Berechtigungen verfügen, die es dem Benutzer ermöglichen, die verfügbaren AWS Glue Kataloge im Zielkonto einzusehen.

Erforderliche Berechtigungen und Vertrauensrichtlinie

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "glue:GetCatalog"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Die Rolle muss über die folgende Vertrauensrichtlinie verfügen, die die Zielkonto-ID angibt.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::111122223333:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

Anleitungen zum Erstellen der Rolle finden Sie unter Erstellen einer Rolle mit benutzerdefinierten Vertrauensrichtlinien.

Erstellen von Null-ETL-Integrationen mit einem Amazon SageMaker-Lakehouse

Sie können eine Zero-ETL-Integration mit einem Amazon SageMaker Lakehouse mithilfe der AWS-Managementkonsole, der oder der AWS CLI RDS-API erstellen.

> **Important**
>
> Null-ETL-Integrationen mit einem Amazon SageMaker Lakehouse unterstützen keine Aktualisierungs- oder Resynchronisierungsvorgänge. Wenn Sie nach der Erstellung auf Probleme mit einer Integration stoßen, müssen Sie die Integration löschen und eine neue erstellen.

RDS-Konsole

So erstellen Sie eine Null-ETL-Integration mit einem Amazon SageMaker Lakehouse

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Null-ETL-Integrationen aus.
3. Wählen Sie Null-ETL-Integration erstellen aus.
4. Geben Sie für Integrationskennung einen Namen für die Integration ein. Der Name kann bis zu 63 alphanumerische Zeichen umfassen und Bindestriche enthalten.
5. Wählen Sie Weiter aus.
6. Wählen Sie als Quelle des Ursprungs der Daten den Aurora DB-Cluster aus.

> **Note**
>
> RDS benachrichtigt Sie, wenn die DB-Cluster-Parameter nicht korrekt konfiguriert sind. Wenn Sie diese Nachricht erhalten, können Sie entweder Reparieren wählen oder eine manuelle Konfiguration vornehmen. Anleitungen zur manuellen Behebung finden Sie unter the section called “Schritt 1: Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe”.

Das Ändern der DB-Cluster-Parameter erfordert einen Neustart. Bevor Sie die Integration erstellen können, müssen der Neustart abgeschlossen und die neuen Parameterwerte erfolgreich auf den Cluster angewendet werden.

7. (Optional) Wählen Sie Datenfilteroptionen anpassen aus und fügen Sie Ihrer Integration Datenfilter hinzu. Sie können Datenfilter verwenden, um den Umfang der Replikation bis zum Ziel-Lakehouse von Amazon SageMaker zu definieren. Weitere Informationen finden Sie unter the section called “Datenfilterung für Null-ETL-Integrationen”.
8. Sobald der Quell-DB-Cluster erfolgreich konfiguriert wurde, wählen Sie Weiter aus.
9. Gehen Sie bei Ziel wie folgt vor:

   1. (Optional) Um ein anderes AWS-Konto für das Amazon SageMaker Lakehouse-Ziel zu verwenden, wählen Sie Anderes Konto angeben aus. Geben Sie dann den ARN einer IAM-Rolle mit Berechtigungen zum Anzeigen Ihrer AWS Glue Kataloge ein. Anleitungen zum Erstellen der IAM-Rolle finden Sie unter the section called “Auswahl eines AWS Glue verwalteten Zielkatalogs in einem anderen Konto”.
   2. Wählen Sie für AWS Glue -Katalog das Ziel für replizierte Daten aus dem Quell DB-Cluster aus. Sie haben die Möglichkeit einen vorhandenen verwalteten AWS Glue -Katalog als Ziel zu verwenden.
   3. Die Ziel-IAM-Rolle muss über Beschreibungsberechtigungen für den Zielkatalog sowie über die folgenden Berechtigungen verfügen:

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "glue:GetCatalog",
            "Resource": [
                "arn:aws:glue:us-east-1:111122223333:catalog/*",
                "arn:aws:glue:us-east-1:111122223333:catalog"
            ]
        }
    ]
}
```

Die Ziel-IAM-Rolle muss über die folgende Vertrauensstellung aufweisen:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

4. Sie müssen der Ziel-IAM-Rolle Beschreibungsberechtigungen für den AWS Glue verwalteten Zielkatalog mit der Lake Formation-Administratorrolle gewähren, die in Schritt 3b: Erstellen Sie einen AWS Glue Katalog für die Amazon SageMaker AI Zero-ETL-Integration erstellt wurde.

> **Note**
>
> RDS benachrichtigt Sie, wenn die Ressourcenrichtlinien oder die Konfigurationseinstellungen für den angegebenen AWS Glue verwalteten Katalog nicht korrekt konfiguriert sind. Wenn Sie diese Nachricht erhalten, können Sie entweder Reparieren wählen oder eine manuelle Konfiguration vornehmen.
> Wenn sich Ihre gewählte Quelle und Ihr Ziel in verschiedenen AWS-Konten befinden, kann Amazon RDS diese Einstellungen nicht für Sie korrigieren. Sie müssen zu dem anderen Konto navigieren und sie manuell in SageMaker Unified Studio korrigieren.

10. Sobald Ihr AWS Glue verwalteter Zielkatalog korrekt konfiguriert ist, wählen Sie Weiter.
11. (Optional) Fügen Sie unter Tags ein oder mehrere Tags zu der Integration hinzu. Weitere Informationen finden Sie unter the section called “Taggen von Aurora- und RDS-Ressourcen”.
12. Geben Sie für Verschlüsselung an, wie Ihre Integration verschlüsselt werden soll. Standardmäßig verschlüsselt RDS alle Integrationen mit einem. AWS-eigener Schlüssel Um stattdessen einen vom Kunden verwalteten Schlüssel auszuwählen, aktivieren Sie die Option Verschlüsselungseinstellungen anpassen und wählen Sie einen KMS-Schlüssel aus,

der für die Verschlüsselung verwendet werden soll. Weitere Informationen finden Sie unter the section called “Verschlüsseln von Amazon Aurora-Ressourcen”.

Fügen Sie optional einen Verschlüsselungskontext hinzu. Weitere Informationen finden Sie unter Verschlüsselungskontext im AWS Key Management Service -Entwicklerhandbuch.

> **Note**
>
> Amazon RDS fügt zusätzlich zu den von Ihnen hinzugefügten Paaren die folgenden Verschlüsselungskontextpaare hinzu:
>
> - `aws:glue:integration:arn` - `IntegrationArn`
> - `aws:servicename:id` - `glue`
>
> Dadurch wird die Gesamtzahl der Paare, die Sie hinzufügen können, von 8 auf 6 reduziert, was zur Gesamtzeichenbegrenzung der Erteilungsbeschränkung beiträgt. Weitere Informationen finden Sie unter Verwenden von Erteilungsbeschränkungen im Entwicklerhandbuch für AWS Key Management Service .

13. Wählen Sie Weiter aus.
14. Überprüfen Sie Ihre Integrationseinstellungen und wählen Sie Null-ETL-Integration erstellen aus.

    Wenn die Erstellung fehlschlägt, finden Sie Informationen zur Fehlerbehebung unter the section called “Fehlerbehebung bei Null-ETL-Integrationen”.

Der Status der Integration lautet während der Erstellung `Creating`. Das Ziel-Lakehouse von Amazon SageMaker hat den Status `Modifying`. Während dieser Zeit können Sie den Katalog weder abfragen noch Konfigurationsänderungen daran vornehmen.

Wenn die Integration erfolgreich erstellt wurde, ändern sich sowohl der Status der Integration als auch der Status des Ziel-Lakehouse von Amazon SageMaker in `Active`.

AWS CLI

Um einen AWS Glue verwalteten Zielkatalog für die Zero-ETL-Integration mit dem vorzubereiten AWS CLI, müssen Sie zunächst den create-integration-resource-propertyBefehl mit den folgenden Optionen verwenden:

- `--resource-arn`— Geben Sie den ARN des AWS Glue verwalteten Katalogs an, der das Ziel für die Integration sein soll.
- `--target-processing-properties`— Geben Sie den ARN der IAM-Rolle an, um auf den AWS Glue verwalteten Zielkatalog zuzugreifen

```
aws glue create-integration-resource-property --region us-east-1
 --resource-arn arn:aws:glue:region:account_id:catalog/catalog_name \
 --target-processing-properties '{"RoleArn" : "arn:aws:iam::account_id:role/
TargetIamRole"}'
```

Um mithilfe von eine Zero-ETL-Integration mit einem Amazon SageMaker Lakehouse zu erstellen AWS CLI, verwenden Sie den Befehl create-integration mit den folgenden Optionen:

- `--integration-name` – Geben Sie einen Namen für die Integration an.
- `--source-arn` – Geben Sie als Quelle für die Integration den ARN des Aurora-DB-Clusters an.
- `--target-arn`— Geben Sie den ARN des AWS Glue verwalteten Katalogs an, der das Ziel für die Integration sein soll.

Example

Für Linux, macOS oder Unix:

```
aws rds create-integration \
    --integration-name my-sagemaker-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:cluster:my-db \
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

Für Windows:

```
aws rds create-integration ^
    --integration-name my-sagemaker-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:cluster:my-db ^
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

RDS-API

Verwenden Sie den `CreateIntegration`-Vorgang mit den folgenden Parametern, um eine Null-ETL-Integration mit Amazon SageMaker über die Amazon-RDS-API zu erstellen:

> **Note**
>
> Katalognamen sind auf 19 Zeichen beschränkt. Stellen Sie sicher, dass Ihr IntegrationName Parameter diese Anforderung erfüllt, wenn er als Katalogname verwendet werden soll.

- `IntegrationName` – Geben Sie einen Namen für die Integration an.
- `SourceArn` – Geben Sie als Quelle für die Integration den ARN des Aurora-DB-Clusters an.
- `TargetArn`— Geben Sie den ARN des AWS Glue verwalteten Katalogs an, der das Ziel für die Integration sein soll.

Verschlüsseln von Integrationen mit einem vom Kunden verwalteten Schlüssel

Wenn Sie AWS-eigener Schlüssel beim Erstellen einer Integration mit einen benutzerdefinierten KMS-Schlüssel und nicht einen angebenAmazon SageMaker, muss die Schlüsselrichtlinie dem SageMaker Unified Studio Dienstprinzipal Zugriff auf die `CreateGrant` Aktion gewähren. Darüber hinaus muss es dem aktuellen Benutzer die Ausführung der Aktionen `DescribeKey` und `CreateGrant` ermöglicht werden.

Die folgende Beispielrichtlinie demonstriert, wie die erforderlichen Berechtigungen in der Schlüsselrichtlinie bereitgestellt werden. Sie enthält Kontextschlüssel, um den Umfang der Berechtigungen weiter zu reduzieren.

Beispiel für eine Schlüsselrichtlinie

JSON

```
{
    "Version":"2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "EnablesIAMUserPermissions",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:root"
            },
```

```
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "GlueServicePrincipalAddGrant",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "AllowsCurrentUserRoleAddGrantKMSKey",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
```

```
            }
        },
        {
            "Sid": "AllowsCurrentUserRoleRetrieveKMSKeyInformation",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

Weitere Informationen finden Sie unter Erstellen einer Schlüsselrichtlinie im AWS Key Management Service -Entwicklerhandbuch.

Nächste Schritte

Nachdem Sie erfolgreich eine Null-ETL-Integration mit Amazon SageMaker erstellt haben, können Sie damit beginnen, dem Quell-Aurora-DB-Cluster Daten hinzuzufügen und in Ihrem Amazon SageMaker-Lakehouse abzufragen. Die Daten werden automatisch repliziert und für Analysen und Machine Learning zur Verfügung gestellt.

Datenfilterung für Null-ETL-Integrationen für Aurora

Null-ETL-Integrationen von Aurora unterstützen die Datenfilterung, mit der Sie steuern können, welche Daten aus Ihrer Quelle, also der dem Aurora-DB-Cluster, in Ihr Ziel-Data-Warehouse repliziert werden. Anstatt die gesamte Datenbank zu replizieren, können Sie einen oder mehrere Filter anwenden, um bestimmte Tabellen ein- oder auszuschließen. Auf diese Weise können Sie die Speicher- und Abfrageleistung optimieren, da nur relevante Daten übertragen werden. Derzeit ist die Filterung auf Datenbank- und Tabellenebene beschränkt. Das Filtern auf Spalten- und Zeilenebene wird nicht unterstützt.

Die Datenfilterung kann nützlich sein, wenn Sie:

- bestimmte Tabellen aus zwei oder mehr verschiedenen Quell-Clustern verknüpfen möchten und nicht alle Daten aus den jeweiligen Clustern benötigen;

- Kosten sparen möchten, indem Sie Analysen nur mit einer Teilmenge von Tabellen und nicht mit einer ganzen Flotte von Datenbanken durchführen;
- vertrauliche Informationen wie Telefonnummern, Adressen oder Kreditkarteninformationen aus bestimmten Tabellen herausfiltern möchten.

Sie können Datenfilter zu einer Zero-ETL-Integration hinzufügen AWS-Managementkonsole, indem Sie die AWS Command Line Interface (AWS CLI) oder die Amazon RDS-API verwenden.

Wenn die Integration einen bereitgestellten Cluster als Ziel hat, muss sich der Cluster auf Patch 180 oder höher befinden, um die Datenfilterung verwenden zu können.

Themen

- Format eines Datenfilters
- Filterlogik
- Rangfolge der Filter
- Beispiele für Aurora MySQL
- Beispiele für Aurora PostgreSQL
- Hinzufügen von Datenfiltern zu einer Integration
- Entfernen von Datenfiltern aus einer Integration

Format eines Datenfilters

Sie können mehrere Filter für eine einzelne Integration definieren. Jeder Filter schließt alle vorhandenen und künftigen Datenbanktabellen, die einem der Muster im Filterausdruck entsprechen, entweder ein oder aus. Null-ETL-Integrationen von Aurora verwenden die Maxwell-Filtersyntax für die Datenfilterung.

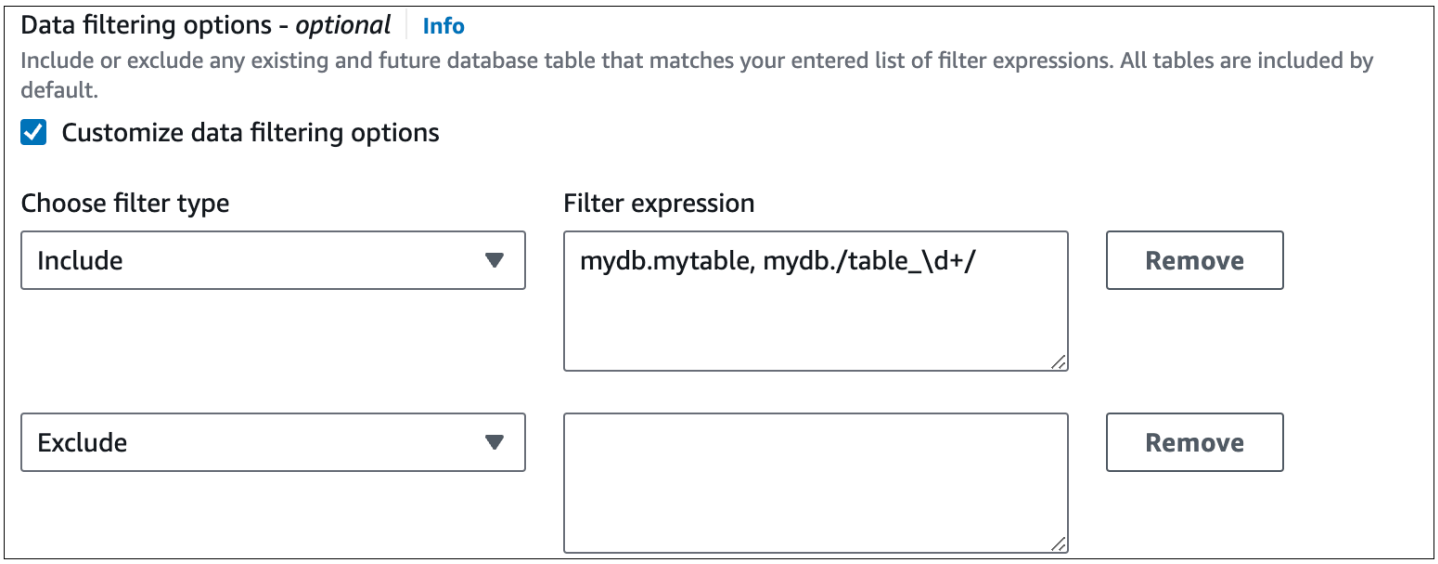
Jeder Filter enthält die folgenden Elemente:

Element	Description
Filtertyp	Der Filtertyp `Include` schließt alle Tabellen, die einem der Muster im Filterausdruck entsprechen, ein. Der Filtertyp `Exclude`

Element	Description
	schließt alle Tabellen, die einem der Muster entsprechen, aus.
Filterausdruck	Eine kommagetrennte Liste von Mustern. Ausdrücke müssen die Maxwell-Filtersyntax verwenden.
Muster	Ein Filtermuster im Format *database*.*table* für Aurora MySQL oder *database*.*schema*.*table* für Aurora PostgreSQL. Sie können Literalnamen angeben oder reguläre Ausdrücke definieren. **Note** Für Aurora MySQL werden reguläre Ausdrücke sowohl im Datenbank- als auch im Tabellennamen unterstützt. Für Aurora PostgreSQL werden reguläre Ausdrücke nur im Schema- und Tabellennamen unterstützt, nicht im Datenbanknamen. Sie können keine Filter oder Zugriffsv erweigerungslisten auf Spaltenebene einbeziehen. Eine einzelne Integration kann insgesamt maximal 99 Muster haben. In der Konsole können Sie Muster innerhalb eines einzelnen Filterausdrucks eingeben oder sie auf mehrere Ausdrücke verteilen. Ein einzelnes Muster darf maximal 256 Zeichen lang sein.

> **Important**
>
> Wenn Sie ein Quell-DB-Cluster von Aurora PostgreSQL auswählen, müssen Sie mindestens ein Datenfiltermuster angeben. Das Muster muss mindestens eine einzelne Datenbank (*`database-name`*`.*.*`) für die Replikation in das Ziel-Data-Warehouse enthalten.

Die folgende Abbildung zeigt die Struktur der Datenfilter für Aurora MySQL in der Konsole:



> **Important**
>
> Geben Sie in Ihren Filtermustern keine persönlich identifizierenden, vertraulichen oder sensiblen Informationen an.

Datenfilter im AWS CLI

Wenn Sie den verwenden AWS CLI , um einen Datenfilter hinzuzufügen, unterscheidet sich die Syntax geringfügig von der der Konsole. Sie müssen jedem Muster einzeln einen Filtertyp (`Include` oder `Exclude`) zuweisen, sodass Sie nicht mehrere Muster unter einem Filtertyp gruppieren können.

In der Konsole können Sie beispielsweise die folgenden durch Kommas getrennten Muster unter einer einzigen `Include`-Anweisung gruppieren:

Aurora MySQL

```
mydb.mytable, mydb./table_\d+/
```

Aurora PostgreSQL

```
mydb.myschema.mytable, mydb.myschema./table_\d+/
```

Wenn Sie den verwenden AWS CLI, muss derselbe Datenfilter jedoch das folgende Format haben:

Aurora MySQL

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

Aurora PostgreSQL

```
'include: mydb.myschema.mytable, include: mydb.myschema./table_\d+/'
```

Filterlogik

Wenn Sie in Ihrer Integration keine Datenfilter angeben, geht Aurora vom Standardfilter `include:*.*` aus, der alle Tabellen in das Ziel-Data-Warehouse repliziert. Wenn Sie jedoch mindestens einen Filter hinzufügen, wechselt die Standardlogik zu `exclude:*.*`, wodurch standardmäßig alle Tabellen ausgeschlossen werden. Auf diese Weise können Sie explizit definieren, welche Datenbanken und Tabellen in die Replikation einbezogen werden sollen.

Sie definieren beispielsweise folgenden Filter:

```
'include: db.table1, include: db.table2'
```

Aurora bewertet den Filter wie folgt:

```
'exclude:*.*, include: db.table1, include: db.table2'
```

Daher repliziert Aurora nur `table1` und `table2` von der angegebenen Datenbank db in das Ziel-Data-Warehouse.

Rangfolge der Filter

Aurora wertet Datenfilter in der von Ihnen angegebenen Reihenfolge aus. In der AWS-Managementkonsole verarbeitet er Filterausdrücke von links nach rechts und von oben nach unten. Ein zweiter Filter oder ein individuelles Muster, das dem ersten folgt, kann ihn/es überschreiben.

Wenn der erste Filter beispielsweise `Include books.stephenking` lautet, schließt er nur die Tabelle `stephenking`aus der Datenbank `books` ein. Wenn Sie jedoch einen zweiten Filter `Exclude books.*` hinzufügen, überschreibt dieser den ersten Filter. Dadurch wird verhindert, dass Tabellen aus dem Index `books` in das Ziel-Data-Warehouse repliziert werden.

Wenn Sie mindestens einen Filter angeben, geht die Logik standardmäßig zunächst von `exclude:*.*` aus, wodurch automatisch alle Tabellen von der Replikation ausgeschlossen werden. Als bewährte Methode empfiehlt es sich, Filter von allgemein nach spezifisch zu definieren. Beginnen Sie mit einer oder mehreren `Include`-Anweisungen, um die zu replizierenden Daten anzugeben, und fügen Sie dann `Exclude`-Filter hinzu, um bestimmte Tabellen selektiv zu entfernen.

Das gleiche Prinzip gilt für Filter, die Sie mit der AWS CLI definieren. Aurora wertet diese Filtermuster in der Reihenfolge aus, in der Sie sie angeben, sodass ein Muster möglicherweise ein Muster überschreibt, das Sie zuvor angegeben haben.

Beispiele für Aurora MySQL

Die folgenden Beispiele zeigen, wie die Datenfilterung für Null-ETL-Integrationen in Beispielen für Aurora MySQL funktioniert:

- Alle Datenbanken und alle Tabellen einschließen:

```
'include: *.*'
```

- Alle Tabellen in der Datenbank `books` einschließen:

```
'include: books.*'
```

- Alle Tabellen mit dem Namen `mystery` ausschließen:

```
'include: *.*, exclude: *.mystery'
```

- Zwei bestimmte Tabellen in der Datenbank `books` einschließen:

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- Alle Tabellen in der Datenbank `books` einschließen, außer denen, die die Teilzeichenfolge `mystery` enthalten:

```
'include: books.*, exclude: books./.*mystery.*/'
```

- Alle Tabellen in der Datenbank `books` einschließen, außer denen, die mit `mystery` beginnen:

```
'include: books.*, exclude: books./mystery.*/'
```

- Alle Tabellen in der Datenbank `books` einschließen, außer denen, die mit `mystery` enden:

```
'include: books.*, exclude: books./.*mystery/'
```

- Alle Tabellen in der Datenbank `books` einschließen, die mit `table_` beginnen, außer der, die den Namen `table_stephen_king` trägt. Zum Beispiel würde `table_movies` oder `table_books` repliziert werden, `table_stephen_king` aber nicht.

```
'include: books./table_.*/, exclude: books.table_stephen_king'
```

Beispiele für Aurora PostgreSQL

Die folgenden Beispiele zeigen, wie die Datenfilterung für Null-ETL-Integrationen für Aurora PostgreSQL funktioniert:

- Alle Tabellen in der Datenbank `books` einschließen:

```
'include: books.*.*'
```

- Alle Tabellen mit dem Namen `mystery` in der Datenbank `books` ausschließen:

```
'include: books.*.*, exclude: books.*.mystery'
```

- Eine Tabelle in der Datenbank `books` im Schema `mystery` und eine Tabelle in der Datenbank `employee` im Schema `finance` einschließen:

```
'include: books.mystery.stephen_king, include: employee.finance.benefits'
```

- Alle Tabellen in der Datenbank `books` und im Schema `science_fiction` einschließen, außer denen, die die Teilzeichenfolge `king` enthalten:

```
'include: books.science_fiction.*, exclude: books.*./.*king.*/
```

- Alle Tabellen in der Datenbank `books` einschließen, außer denen, deren Schemaname mit `sci` beginnt:

```
'include: books.*.*, exclude: books./sci.*/.*'
```

- Alle Tabellen in der Datenbank `books` einschließen, außer denen im Schema `mystery`, die mit `king` enden:

```
'include: books.*.*, exclude: books.mystery./.*king/'
```

- Alle Tabellen in der Datenbank `books` einschließen, die mit `table_` beginnen, außer der, die den Namen `table_stephen_king` trägt. Zum Beispiel werden `table_movies` im Schema `fiction` und `table_books` im Schema `mystery` repliziert, `table_stephen_king` jedoch in keinem Schema:

```
'include: books.*./table_.*/, exclude: books.*.table_stephen_king'
```

Hinzufügen von Datenfiltern zu einer Integration

Sie können die Datenfilterung mithilfe der AWS-Managementkonsole AWS CLI, der oder der Amazon RDS-API konfigurieren.

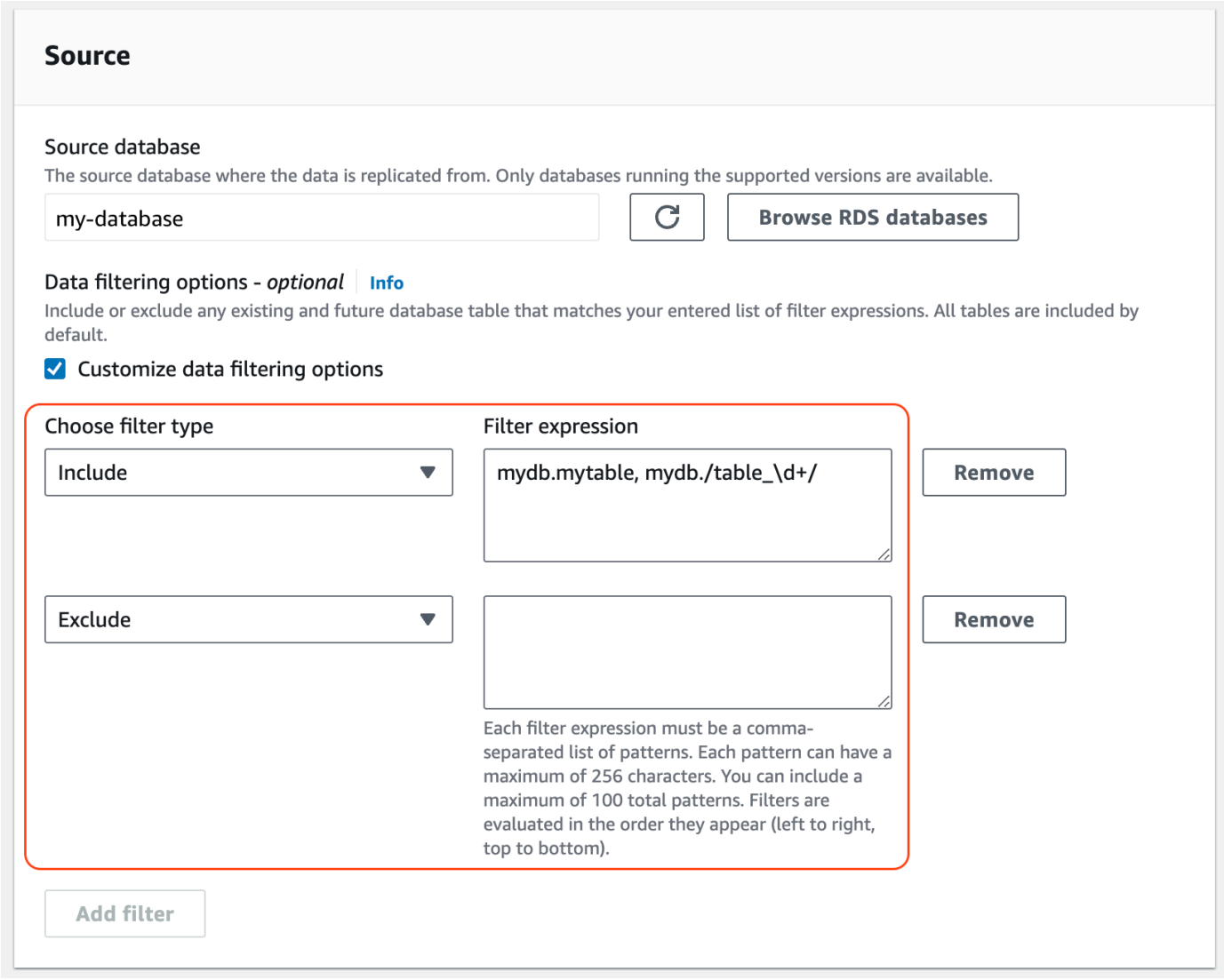
> **Important**
>
> Wenn Sie einen Filter hinzufügen, nachdem Sie eine Integration erstellt haben, behandelt Aurora ihn so, als ob er schon immer existiert hätte. Alle Daten im Ziel-Data-Warehouse, die nicht den neuen Filterkriterien entsprechen, werden entfernt und alle betroffenen Tabellen erneut synchronisiert.

RDS-Konsole

So fügen Sie Datenfilter zur einer Null-ETL-Integration hinzu

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Null-ETL-Integrationen aus. Wählen Sie die Integration aus, zu der Sie Datenfilter hinzufügen möchten, und klicken Sie dann auf Ändern.
3. Fügen Sie unter Quelle eine oder mehrere `Include`- und `Exclude`-Anweisungen hinzu.

   Die folgende Abbildung zeigt ein Beispiel für Datenfilter für eine MySQL-Integration:



4. Wenn Sie mit den Änderungen zufrieden sind, wählen Sie Weiter und Änderungen speichern.

AWS CLI

Rufen Sie den Befehl modify-integration auf, um einer Zero-ETL-Integration mithilfe von Datenfilter hinzuzufügen. AWS CLI Geben Sie den Parameter `--data-filter` zusätzlich zur Integrations-ID mit einer kommagetrennten Liste von `Include`- und `Exclude`-Maxwell-Filtern an.

Example

Im folgenden Beispiel werden Filtermuster zu `my-integration` hinzugefügt.

Für Linux, macOS oder Unix:

```
aws rds modify-integration \
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

Für Windows:

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

RDS-API

Rufen Sie den Vorgang auf, um eine Zero-ETL-Integration mithilfe der RDS-API zu ändern. ModifyIntegration Geben Sie die Integrations-ID an und stellen Sie eine kommagetrennte Liste der Filtermuster bereit.

Entfernen von Datenfiltern aus einer Integration

Wenn Sie einen Datenfilter aus einer Integration entfernen, wertet Aurora die verbleibenden Filter neu aus, als ob der entfernte Filter nie existiert hätte. Anschließend repliziert es alle zuvor ausgeschlossenen Daten, die jetzt die Kriterien erfüllen, in das Ziel-Data-Warehouse. Dadurch wird eine erneute Synchronisierung aller betroffenen Tabellen ausgelöst.

Hinzufügen von Daten zu einem Quell-DB-Cluster von Aurora und Abfragen der Daten

Zum Erstellen einer Null-ETL-Integration, die Daten von Amazon Aurora in Amazon Redshift repliziert, müssen Sie eine Datenbank im Ziel erstellen.

Stellen Sie für Verbindungen mit Amazon Redshift zunächst eine Verbindung mit Ihrem Amazon-Redshift-Cluster oder Ihrer Arbeitsgruppe her und erstellen Sie eine Datenbank mit einem Verweis auf Ihre Integrations-ID. Anschließend können Sie Daten zum Quell-DB-Cluster von Aurora hinzufügen und diese in Amazon Redshift oder Amazon SageMaker replizieren.

Themen

- Erstellen einer Zieldatenbank
- Hinzufügen von Daten zum Quell-DB-Cluster
- Abfragen Ihrer Aurora-Daten in Amazon Redshift
- Datentypunterschiede zwischen Aurora und Amazon Redshift-Datenbanken
- DDL-Vorgänge für Aurora PostgreSQL

Erstellen einer Zieldatenbank

Bevor Sie nach der Erstellung einer Integration mit der Replikation von Daten in Amazon Redshift beginnen können, müssen Sie in Ihrem Ziel-Data-Warehouse eine Datenbank erstellen. Diese Datenbank muss einen Verweis auf die Integrations-ID enthalten. Sie können die Amazon-Redshift-Konsole oder Query Editor v2 verwenden, um die Datenbank zu erstellen.

Anleitungen zum Erstellen einer Zieldatenbank finden Sie unter Erstellen einer Zieldatenbank in Amazon Redshift.

Hinzufügen von Daten zum Quell-DB-Cluster

Nachdem Sie Ihre Integration konfiguriert haben, können Sie den Quell-DB-Cluster von Aurora mit Daten füllen, die Sie in Ihr Data Warehouse replizieren möchten.

> **Note**
>
> Es gibt Unterschiede zwischen den Datentypen in Amazon Aurora und dem Ziel-Analyse-Warehouse. Eine Tabelle mit Datentypzuordnungen finden Sie unter the section called “Datentypunterschiede”.

Stellen Sie zunächst mit dem MySQL- oder PostgreSQL-Client Ihrer Wahl eine Verbindung zum Quell-DB-Cluster her. Anleitungen finden Sie unter the section called “Herstellen einer Verbindung mit einem DB-Cluster”.

Erstellen Sie dann eine Tabelle und fügen Sie eine Zeile mit Beispieldaten ein.

> **Important**
>
> Stellen Sie sicher, dass die Tabelle über einen Primärschlüssel verfügt. Andernfalls kann sie nicht in das Ziel-Data-Warehouse repliziert werden.

Die PostgreSQL-Dienstprogramme pg_dump und pg_restore erstellen zunächst Tabellen ohne Primärschlüssel und fügen ihn anschließend hinzu. Wenn Sie eines dieser Dienstprogramme verwenden, empfehlen wir, zuerst ein Schema zu erstellen und dann Daten in einem separaten Befehl zu laden.

MySQL

Im folgenden Beispiel wird das Dienstprogramm MySQL Workbench verwendet.

```
CREATE DATABASE my_db;

USE my_db;

CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

PostgreSQL

Das folgende Beispiel verwendet das interaktive PostgreSQL-Terminal psql. Schließen Sie beim Herstellen einer Verbindung mit dem Cluster die benannte Datenbank ein, die Sie beim Erstellen der Integration angegeben haben.

```
psql -h mycluster.cluster-123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username
 -d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));
```

```
named_db=> INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977,
 'Supernatural fiction');
```

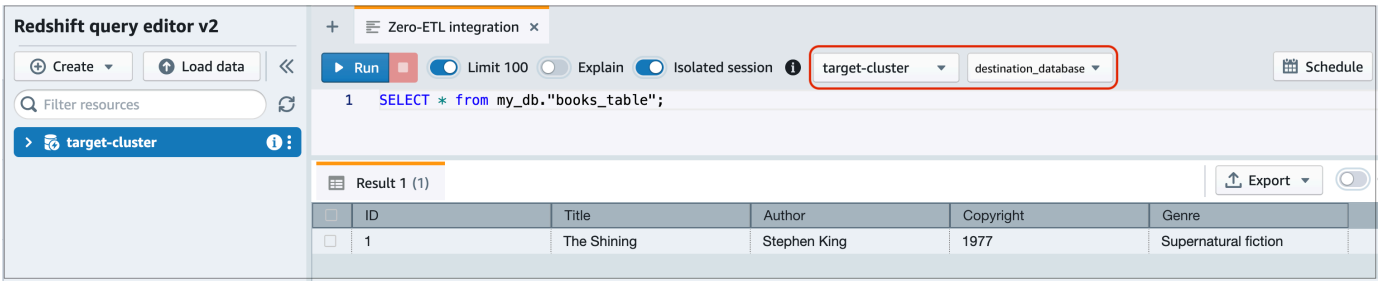
Abfragen Ihrer Aurora-Daten in Amazon Redshift

Nachdem Sie Daten zum Aurora-DB-Cluster hinzugefügt haben, werden sie in die Zieldatenbank repliziert und können abgefragt werden.

So fragen Sie die replizierten Daten ab

1. Navigieren Sie zur Amazon Redshift-Konsole und wählen Sie im linken Navigationsbereich die Option Query Editor v2 aus.
2. Stellen Sie eine Verbindung mit Ihrem Cluster oder Ihrer Arbeitsgruppe her und wählen Sie Ihre aus der Integration erstellte Datenbank im Dropdown-Menü aus (destination_database in diesem Beispiel). Anleitungen zum Erstellen einer Zieldatenbank finden Sie unter Erstellen einer Zieldatenbank in Amazon Redshift.
3. Verwenden Sie eine SELECT-Anweisung, um Ihre Daten abzufragen. In diesem Beispiel können Sie den folgenden Befehl ausführen, um alle Daten aus der Tabelle auszuwählen, die Sie im Quell-DB-Cluster von Aurora erstellt haben:

   ```
   SELECT * from my_db."books_table";
   ```

   - *my_db* ist der Name des Aurora-Datenbankschemas. Diese Option wird nur für MySQL-Datenbanken benötigt.
   - *books_table* ist der Name der Aurora-Tabelle.

Sie können die Daten auch mit einem Befehlszeilen-Client abfragen. Zum Beispiel:

```
destination_database=# select * from my_db."books_table";
```

```
 ID |       Title |        Author |   Copyright |                  Genre |  txn_seq |
 txn_id
----+-------------+---------------+-------------+------------------------+----------
+--------+
  1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
 12192
```

> **Note**
>
> Um zwischen Groß- und Kleinschreibung zu unterscheiden, verwenden Sie doppelte Anführungszeichen (" ") für Schema-, Tabellen- und Spaltennamen. Weitere Informationen finden Sie unter enable_case_sensitive_identifier.

Datentypunterschiede zwischen Aurora und Amazon Redshift-Datenbanken

Die folgenden Tabellen zeigen die Zuordnungen der Datentypen von Aurora MySQL und Aurora PostgreSQL zu den entsprechenden Zieldatentypen. Amazon Aurora unterstützt derzeit nur diese Datentypen für Null-ETL-Integrationen.

Wenn eine Tabelle in dem Quell-DB-Cluster einen nicht unterstützten Datentyp enthält, wird die Tabelle nicht synchronisiert und kann vom Ziel nicht genutzt werden. Das Streaming von der Quelle zum Ziel wird fortgesetzt, aber die Tabelle mit dem nicht unterstützten Datentyp ist nicht verfügbar. Um die Tabelle zu reparieren und im Ziel verfügbar zu machen, müssen Sie die grundlegende Änderung manuell rückgängig machen und dann die Integration aktualisieren, indem Sie `ALTER DATABASE...INTEGRATION REFRESH` ausführen.

> **Note**
>
> Null-ETL-Integrationen mit einem Amazon SageMaker-Lakehouse können nicht aktualisiert werden. Löschen Sie stattdessen die Integration und erstellen Sie sie erneut.

Themen

- Aurora MySQL
- Aurora PostgreSQL

Aurora MySQL

Aurora-MySQL-Datentyp	Zieldatentyp	Beschreibung	Einschränkungen
INT	INTEGER	4-Byte-Ganzzahl mit Vorzeichen	Keine
SMALLINT	SMALLINT	2-Byte-Ganzzahl mit Vorzeichen	Keine
TINYINT	SMALLINT	2-Byte-Ganzzahl mit Vorzeichen	Keine
MEDIUMINT	INTEGER	4-Byte-Ganzzahl mit Vorzeichen	Keine
BIGINT	BIGINT	8-Byte-Ganzzahl mit Vorzeichen	Keine
INT UNSIGNED	BIGINT	8-Byte-Ganzzahl mit Vorzeichen	Keine
TINYINT UNSIGNED	SMALLINT	2-Byte-Ganzzahl mit Vorzeichen	Keine
MEDIUMINT UNSIGNED	INTEGER	4-Byte-Ganzzahl mit Vorzeichen	Keine
BIGINT UNSIGNED	DECIMAL(20,0)	Genauer Zahlenwert mit wählbarer Genauigkeit	Keine
DECIMAL(P,S) = NUMERIC(P,S)	DECIMAL (p,s)	Genauer Zahlenwert mit wählbarer Genauigkeit	Eine Genauigkeit von mehr als 38 und eine Skalierung von mehr als 37

Aurora-MySQL-Datentyp	Zieldatentyp	Beschreibung	Einschränkungen
			werden nicht unterstützt.
DECIMAL(P,S) UNSIGNED = NUMERIC(P,S) UNSIGNED	DECIMAL (p,s)	Genauer Zahlenwert mit wählbarer Genauigkeit	Eine Genauigkeit von mehr als 38 und eine Skalierung von mehr als 37 werden nicht unterstützt.
FLOAT4/REAL	REAL	Gleitkommazahl mit einfacher Genauigkeit	Keine
FLOAT4/REAL UNSIGNED	REAL	Gleitkommazahl mit einfacher Genauigkeit	Keine
DOUBLE/REAL/FLOAT8	DOUBLE PRECISION	Double (Gleitkommazahl mit doppelter Genauigkeit)	Keine
DOUBLE/REAL/FLOAT8 UNSIGNED	DOUBLE PRECISION	Double (Gleitkommazahl mit doppelter Genauigkeit)	Keine
BIT(N)	VARBYTE(8)	Binärwert mit variabler Länge	Keine
BINARY(n)	VARBYTE(n)	Binärwert mit variabler Länge	Keine

Aurora-MySQL-Datentyp	Zieldatentyp	Beschreibung	Einschränkungen
VARBINARY (n)	VARBYTE(n)	Binärwert mit variabler Länge	Keine
CHAR(n)	VARCHAR (n)	Zeichenkettenwert mit variabler Länge	Keine
VARCHAR (n)	VARCHAR (n)	Zeichenkettenwert mit variabler Länge	Keine
TEXT	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
TINYTEXT	VARCHAR(255)	Zeichenkettenwert variabler Länge bis zu 255 Zeichen	Keine
MEDIUMTEXT	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
LONGTEXT	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
ENUM	VARCHAR(1020)	Zeichenkettenwert variabler Länge bis zu 1.020 Zeichen	Keine

Aurora-MySQL-Datentyp	Zieldatentyp	Beschreibung	Einschränkungen
SET	VARCHAR(1020)	Zeichenkettenwert variabler Länge bis zu 1.020 Zeichen	Keine
DATUM	DATUM	Kalenderdatum (Jahr, Monat, Tag)	Keine
DATETIME	TIMESTAMP (ZEITSTEMPEL)	Datum und Uhrzeit (ohne Zeitzone)	Keine
TIMESTAMP(p)	TIMESTAMP (ZEITSTEMPEL)	Datum und Uhrzeit (ohne Zeitzone)	Keine
TIME	VARCHAR(18)	Zeichenkettenwert variabler Länge bis zu 18 Zeichen	Keine
JAHR	VARCHAR(4)	Zeichenkettenwert variabler Länge bis zu 4 Zeichen	Keine
JSON	SUPER	Semistrukturierte Daten oder Dokumente als Werte	Keine

Aurora PostgreSQL

Null-ETL-Integrationen für Aurora PostgreSQL unterstützen keine benutzerdefinierten Datentypen oder Datentypen, die durch Erweiterungen erstellt wurden.

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
Array	SUPER	Semistrukturierte Daten oder Dokumente als Werte	Keine
bigint	BIGINT	8-Byte-Ganzzahl mit Vorzeichen	Keine
BIGSERIAL	BIGINT	8-Byte-Ganzzahl mit Vorzeichen	Keine
bit varying(n)	VARBYTE(n)	Binärwert variabler Länge bis zu 16.777.216 Byte	Keine
BIT(N)	VARBYTE(n)	Binärwert variabler Länge bis zu 16.777.216 Byte	Keine
BIT, BIT VARYING	VARBYTE(16777216)	Binärwert variabler Länge bis zu 16.777.216 Byte	Keine
Boolean	BOOLEAN	Logischer/Boolescher Wert (wahr/falsch)	Keine

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
bytea	VARBYTE(16777216)	Binärwert variabler Länge bis zu 16.777.216 Byte	Keine
CHAR(n)	CHAR(n)	Zeichenkettenwert mit fester Länge bis zu 65.535 Byte	Keine
char varying(n)	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
cid	BIGINT	8-Byte-Ganzzahl mit Vorzeichen	Keine
CIDR	VARCHAR(19)	Zeichenkettenwert variabler Länge bis zu 19 Zeichen	Keine
date	DATUM	Kalenderdatum (Jahr, Monat, Tag)	Werte, die größer als 294.276 A.D. sind, werden nicht unterstützt
double precision	DOUBLE PRECISION	Double (Gleitkommazahlen mit doppelter Genauigkeit)	Subnormale Werte werden nicht vollständig unterstützt

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
gtsvector	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
INET	VARCHAR(19)	Zeichenkettenwert variabler Länge bis zu 19 Zeichen	Keine
Ganzzahl	INTEGER	4-Byte-Ganzzahl mit Vorzeichen	Keine
int2vector	SUPER	Semistrukturierte Daten oder Dokumente als Werte	Keine
Intervall	INTERVAL	Dauer	Es werden nur INTERVAL-Typen unterstützt, die einen Qualifier des Typs „Jahr zu Monat“ oder „Tag zu Sekunde“ angeben.
json	SUPER	Semistrukturierte Daten oder Dokumente als Werte	Keine

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
JSONB	SUPER	Semistrukturierte Daten oder Dokumente als Werte	Keine
jsonpath	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
MACADDR	VARCHAR(17)	Zeichenkettenwert variabler Länge bis zu 17 Zeichen	Keine
macaddr8	VARCHAR(23)	Zeichenkettenwert variabler Länge bis zu 23 Zeichen	Keine
money	DECIMAL(20,3)	Währungsbetrag	Keine
Name	VARCHAR(64)	Zeichenkettenwert variabler Länge bis zu 64 Zeichen	Keine

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
numeric(p,s)	DECIMAL (p,s)	Benutzerdefinierter Wert mit fester Präzision	• NaN-Werte werden nicht unterstützt • Genauigkeit und Maßstab müssen explizit definiert werden und dürfen nicht größer als 38 (Genauigkeit) und 37 (Skala) sein. • Negative Skala wird nicht unterstützt
oid	BIGINT	8-Byte-Ganzzahl mit Vorzeichen	Keine
oidvector	SUPER	Semistrukturierte Daten oder Dokumente als Werte	Keine
pg_brin_bloom_summary	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
pg_dependencies	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
PG_LSN	VARCHAR(17)	Zeichenkettenwert variabler Länge bis zu 17 Zeichen	Keine
pg_mcv_list	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
pg_ndistinct	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
pg_node_tree	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
pg_snapshot	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
real	REAL	Gleitkommazahl mit einfacher Genauigkeit	Subnormale Werte werden nicht vollständig unterstützt

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
REFCURSOR	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
smallint	SMALLINT	2-Byte-Ganzzahl mit Vorzeichen	Keine
SMALLSERIAL	SMALLINT	2-Byte-Ganzzahl mit Vorzeichen	Keine
serial	INTEGER	4-Byte-Ganzzahl mit Vorzeichen	Keine
text	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
tid	VARCHAR(23)	Zeichenkettenwert variabler Länge bis zu 23 Zeichen	Keine
time [(p)] without time zone	VARCHAR(19)	Zeichenkettenwert variabler Länge bis zu 19 Zeichen	Die Werte `Infinity` und `-Infinity` werden nicht unterstützt.
time [(p)] with time zone	VARCHAR(22)	Zeichenkettenwert variabler Länge bis zu 22 Zeichen	Die Werte `Infinity` und `-Infinity` werden nicht unterstützt.

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
timestamp [(p)] without time zone	TIMESTAMP (ZEITSTEMPEL)	Datum und Uhrzeit (ohne Zeitzone)	• Die Werte `Infinity` und `-Infinity` werden nicht unterstützt. • Werte, die größer als `9999-12-31` sind, werden nicht unterstützt. • B.C.-Werte werden nicht unterstützt.
timestamp [(p)] with time zone	TIMESTAMPTZ	Datum und Uhrzeit (mit Zeitzone)	• Die Werte `Infinity` und `-Infinity` werden nicht unterstützt. • Werte, die größer als `9999-12-31` sind, werden nicht unterstützt. • B.C.-Werte werden nicht unterstützt.

Datentyp von Aurora PostgreSQL	Amazon-Redshift-Datentyp	Beschreibung	Einschränkungen
TSQUERY	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
TSVECTOR	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
TXID_SNAPSHOT	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine
uuid	VARCHAR(36)	Zeichenkette variabler Länge mit 36 Zeichen	Keine
xid	BIGINT	8-Byte-Ganzzahl mit Vorzeichen	Keine
xid8	DECIMAL(20, 0)	Dezimalzahl mit fester Präzision	Keine
xml	VARCHAR(65535)	Zeichenkettenwert variabler Länge bis zu 65.535 Zeichen	Keine

DDL-Vorgänge für Aurora PostgreSQL

Amazon Redshift wird von PostgreSQL abgeleitet und hat daher aufgrund der gemeinsamen PostgreSQL-Architektur mehrere Features mit Aurora PostgreSQL gemeinsam. Null-ETL-Integrationen nutzen diese Ähnlichkeiten, um die Datenreplikation von Aurora PostgreSQL in Amazon

Redshift zu optimieren, indem sie Datenbanken nach Namen zuordnen und die Datenbank, das Schema und die Tabellenstruktur gemeinsam nutzen.

Beachten Sie bei der Verwaltung von Null-ETL-Integrationen von Aurora PostgreSQL die folgenden Punkte:

- Die Isolation wird auf Datenbankebene verwaltet.
- Die Replikation erfolgt auf Datenbankebene.
- Datenbanken von Aurora PostgreSQL werden Amazon-Redshift-Datenbanken nach Namen zugeordnet, wobei Daten in die entsprechende umbenannte Redshift-Datenbank fließen, wenn das Original umbenannt wird.

Trotz ihrer Ähnlichkeiten weisen Amazon Redshift und Aurora PostgreSQL wesentliche Unterschiede auf. In den folgenden Abschnitten werden die Systemantworten von Amazon Redshift für gängige DDL-Vorgänge beschrieben.

Themen

- Datenbankvorgänge
- Schemavorgänge
- Tabellenoperationen

Datenbankvorgänge

Die folgende Tabelle zeigt die Systemantworten für Datenbank-DDL-Vorgänge.

DDL-Vorgang	Redshift-Systemantwort
`CREATE DATABASE`	Kein Vorgang
`DROP DATABASE`	Amazon Redshift entfernt alle Daten in der Redshift-Zieldatenbank.
`RENAME DATABASE`	Amazon Redshift entfernt alle Daten in der ursprünglichen Zieldatenbank und synchroni siert die Daten in der neuen Zieldatenbank erneut. Wenn die neue Datenbank nicht existiert, müssen Sie sie manuell erstellen.

DDL-Vorgang	Redshift-Systemantwort
	Anleitungen finden Sie unter Eine Zieldaten bank in Amazon Redshift erstellen.

Schemavorgänge

Die folgende Tabelle zeigt die Systemantworten für Schema-DDL-Vorgänge.

DDL-Vorgang	Redshift-Systemantwort
`CREATE SCHEMA`	Kein Vorgang
`DROP SCHEMA`	Amazon Redshift entfernt das ursprüngliche Schema.
`RENAME SCHEMA`	Amazon Redshift entfernt das ursprüngliche Schema und synchronisiert dann die Daten im neuen Schema erneut.

Tabellenoperationen

Die folgende Tabelle zeigt die Systemantworten auf Tabellen-DDL-Vorgänge.

DDL-Vorgang	Redshift-Systemantwort
`CREATE TABLE`	Amazon Redshift erstellt die Tabelle. Einige Vorgänge führen dazu, dass die Tabellenerstellung fehlschlägt, z. B. das Erstellen einer Tabelle ohne Primärschlüssel oder das Durchführen einer deklarativen Partitionierung. Weitere Informationen erhalten Sie unter the section called “Einschränkungen” und the section called “Fehlerbehebung bei Null-ETL-Integrationen”.

DDL-Vorgang	Redshift-Systemantwort
`DROP TABLE`	Amazon Redshift entfernt die Tabelle.
`TRUNCATE TABLE`	Amazon Redshift schneidet die Tabelle ab.
`ALTER TABLE (RENAME... )`	Amazon Redshift benennt die Tabelle oder Spalte um.
`ALTER TABLE (SET SCHEMA)`	Amazon Redshift entfernt die Tabelle im ursprünglichen Schema und synchronisiert dann die Tabelle im neuen Schema erneut.
`ALTER TABLE (ADD PRIMARY KEY)`	Amazon Redshift fügt einen Primärschlüssel hinzu und synchronisiert die Tabelle erneut.
`ALTER TABLE (ADD COLUMN)`	Amazon Redshift fügt eine Spalte zur Tabelle hinzu.
`ALTER TABLE (DROP COLUMN)`	Amazon Redshift entfernt die Spalte, wenn es sich nicht um eine Primärschlüsselspalte handelt. Andernfalls wird die Tabelle erneut synchronisiert.
`ALTER TABLE (SET LOGGED/UNLOGGED )`	Wenn Sie die Tabelle in protokolliert ändern, synchronisiert Amazon Redshift die Tabelle erneut. Wenn Sie die Tabelle in nicht protokoll iert ändern, entfernt Amazon Redshift die Tabelle.

Anzeigen und Überwachen von Null-ETL-Integrationen von Aurora

Sie können die Details einer Null-ETL-Integration von Amazon Aurora anzeigen, um die zugehörigen Konfigurationsinformationen und den aktuellen Status einzusehen. Sie können außerdem den Status Ihrer Integration überwachen, indem Sie bestimmte Systemansichten in Amazon Redshift abfragen. Darüber hinaus veröffentlicht Amazon Redshift bestimmte integrationsbezogene Metriken auf Amazon CloudWatch, die Sie in der Amazon Redshift Redshift-Konsole einsehen können.

Themen

- Anzeigen von Integrationen
- Überwachen von Integrationen mithilfe von Systemtabellen für Amazon Redshift
- Überwachung von Integrationen mit Amazon EventBridge für Amazon Redshift

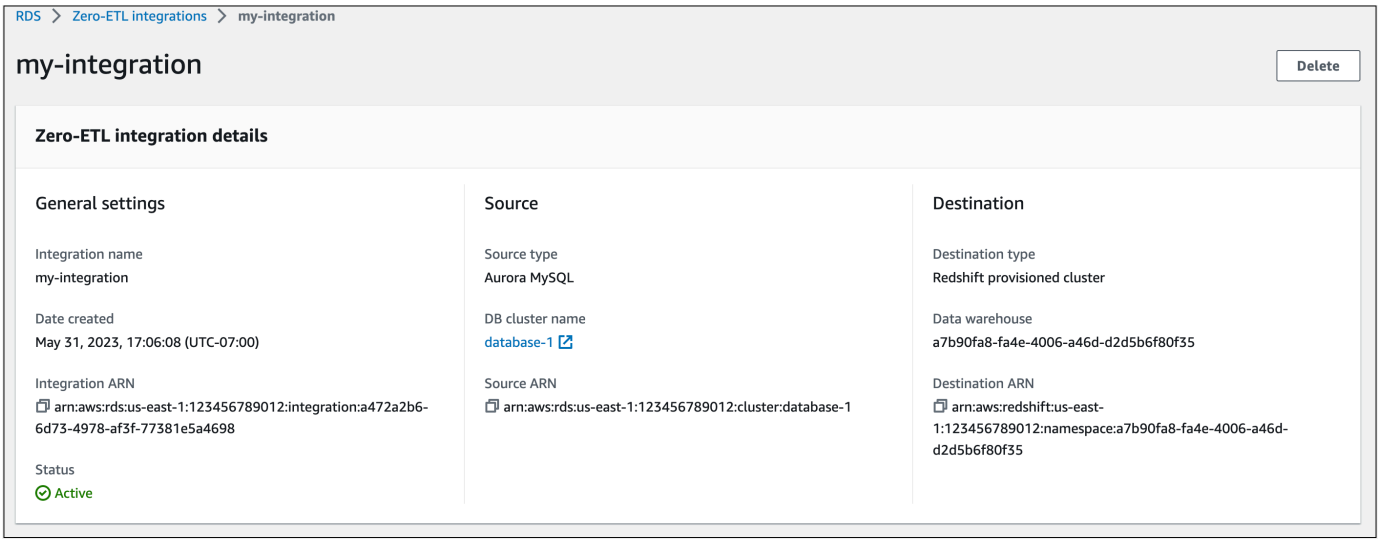
Anzeigen von Integrationen

Sie können Aurora Zero-ETL-Integrationen mithilfe der AWS-Managementkonsole, der oder der AWS CLI RDS-API anzeigen.

Konsole

So zeigen Sie die Details einer Null-ETL-Integration an

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Null-ETL-Integrationen aus.
3. Wählen Sie eine Integration aus, um weitere Details dazu anzuzeigen, z. B. den Quell-DB-Cluster und das Ziel-Data-Warehouse.



Eine Integration kann folgende Status aufweisen:

- `Creating` – Die Integration wird erstellt.
- `Active` – Die Integration sendet Transaktionsdaten an das Ziel-Data-Warehouse.

- `Syncing` – Bei der Integration ist ein behebbarer Fehler aufgetreten und die Daten werden erneut gesendet. Betroffene Tabellen sind erst für Abfragen verfügbar, wenn die erneute Synchronisierung abgeschlossen ist.
- `Needs attention` – Bei der Integration ist ein Ereignis oder ein Fehler aufgetreten, für dessen Behebung ein manuelles Eingreifen erforderlich ist. Befolgen Sie zur Behebung des Problems die Anweisungen in der Fehlermeldung auf der Seite mit den Integrationsdetails.
- `Failed` – Bei der Integration ist ein nicht behebbares Ereignis oder ein Fehler aufgetreten, der nicht behoben werden kann. Sie müssen die Integration löschen und erneut erstellen.
- `Deleting` – Die Integration wird gelöscht.

AWS CLI

Um alle Zero-ETL-Integrationen im aktuellen Konto mit dem anzuzeigen AWS CLI, verwenden Sie den Befehl describe-integrations und geben Sie die Option an. `--integration-identifier`

Example

Für Linux, macOS oder Unix:

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Für Windows:

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS-API

Um die Null-ETL-Integration mithilfe der Amazon-RDS-API anzuzeigen, verwenden Sie die Operation `DescribeIntegrations` mit dem Parameter `IntegrationIdentifier`.

Überwachen von Integrationen mithilfe von Systemtabellen für Amazon Redshift

Amazon Redshift verfügt über Systemtabellen und Ansichten, die Informationen zur Funktionsweise des Systems enthalten. Sie können diese Systemtabellen und Ansichten

genauso abfragen wie andere Datenbanktabellen. Weitere Informationen zu Systemtabellen und Ansichten in Amazon Redshift finden Sie in der Referenz zu Systemtabellen und Ansichten im Datenbankentwicklerhandbuch zu Amazon Redshift.

Sie können die folgenden Systemansichten und Tabellen abfragen, um Informationen zu Ihren Null-ETL-Integrationen von Aurora in abzurufen:

- SVV_INTEGRATION – Stellt Konfigurationsdetails für Ihre Integrationen bereit.
- SVV_INTEGRATION_TABLE_STATE – Beschreibt den Status jeder Tabelle innerhalb einer Integration.
- SYS_INTEGRATION_TABLE_STATE_CHANGE – Zeigt Protokolle zur Statusänderung von Tabellen für eine Integration an.
- SYS_INTEGRATION_ACTIVITY – Stellt Informationen zu abgeschlossenen Integrationsausführungen bereit.

Alle integrationsbezogenen CloudWatch Amazon-Metriken stammen von Amazon Redshift. Weitere Informationen finden Sie unter Metriken für Null-ETL-Integrationen im Amazon-Redshift-Verwaltungshandbuch. Derzeit veröffentlicht Amazon Aurora keine Integrationsmetriken für CloudWatch.

Überwachung von Integrationen mit Amazon EventBridge für Amazon Redshift

Amazon Redshift sendet integrationsbezogene Ereignisse an Amazon. EventBridge Eine Liste der Ereignisse und der entsprechenden Ereignisse finden Sie unter Benachrichtigungen über Zero-ETL-Integrationsereignisse mit Amazon EventBridge im Amazon Redshift Management Guide. IDs

Ändern der Null-ETL-Integrationen von Aurora

Sie können nur den Namen, die Beschreibung und die Datenfilteroptionen für eine Null-ETL-Integration in einem unterstützten Data Warehouse ändern. Den zur Verschlüsselung der Integration verwendeten AWS KMS-Schlüssel oder die Quell- oder Zieldatenbanken können Sie nicht ändern.

Wenn Sie einer vorhandenen Integration einen Datenfilter hinzufügen, wertet Aurora den Filter neu aus, als ob er schon immer existiert hätte. Es entfernt alle Daten, die sich derzeit im Ziel-Data-Warehouse befinden und nicht den neuen Filterkriterien entsprechen. Wenn Sie einen Datenfilter aus einer Integration entfernen, werden alle Daten, die zuvor nicht den Filterkriterien entsprachen

(dies jetzt aber tun), in das Ziel-Data-Warehouse repliziert. Weitere Informationen finden Sie unter the section called “Datenfilterung für Null-ETL-Integrationen”.

Sie können eine Null-ETL-Integration mit der AWS-Managementkonsole, der AWS CLI oder der Amazon-RDS-API erstellen.

RDS-Konsole

So ändern Sie eine Null-ETL-Integration

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Null-ETL-Integrationen und dann die Integration aus, die Sie ändern möchten.
3. Wählen Sie Ändern und nehmen Sie Änderungen an allen verfügbaren Einstellungen vor.
4. Nachdem Sie die gewünschten Änderungen vorgenommen haben, klicken Sie auf Ändern.

AWS CLI

Um eine Null-ETL-Integration mithilfe der AWS CLI zu ändern, rufen Sie den Befehl modify-integration auf. Geben Sie zusammen mit der `--integration-identifier` eine der folgenden Optionen an:

- `--integration-name` – Geben Sie einen neuen Namen für die Integration an.
- `--description` – Geben Sie eine neue Beschreibung für die Integration an.
- `--data-filter` – Geben Sie Datenfilteroptionen für die Integration an. Weitere Informationen finden Sie unter the section called “Datenfilterung für Null-ETL-Integrationen”.

Example

Die folgende Anfrage ändert eine bestehende Integration.

Für Linux, macOS oder Unix:

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

Für Windows:

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

RDS-API

Rufen Sie die Operation ModifyIntegration auf, um eine Null-ETL-Integration mithilfe der RDS-API zu ändern. Geben Sie die Integrations-ID und die Parameter an, die geändert werden sollen.

Löschen von Null-ETL-Integrationen von Aurora

Wenn Sie eine Null-ETL-Integration löschen, entfernt Amazon Aurora sie aus dem Quell-DB-Cluster von Aurora. Ihre Transaktionsdaten werden nicht aus Amazon Aurora oder dem Analyseziel gelöscht, Aurora sendet jedoch keine neuen Daten an Amazon Redshift oder Amazon SageMaker.

Sie können eine Integration nur löschen, wenn sie den Status `Active`, `Failed`, `Syncing` oder `Needs attention` hat.

Sie können Null-ETL-Integrationen mithilfe der AWS-Managementkonsole, der AWS CLI oder der RDS-API löschen.

Konsole

So löschen Sie eine Null-ETL-Integration

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im linken Navigationsbereich Null-ETL-Integrationen aus.
3. Wählen Sie die Null-ETL-Integration aus, die Sie löschen möchten.
4. Klicken Sie auf Aktionen, Löschen und bestätigen Sie den Löschvorgang.

AWS CLI

Verwenden Sie zum Löschen einer Null-ETL-Integration den Befehl delete-integration und geben Sie die Option `--integration-identifier` an.

Example

Für Linux, macOS oder Unix:

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Für Windows:

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS-API

Verwenden Sie zum Löschen einer Null-ETL-Integration mithilfe der Amazon-RDS-API die Operation `DeleteIntegration` mit dem Parameter `IntegrationIdentifier`.

Fehlerbehebung bei Null-ETL-Integrationen von Aurora

Sie können den Status einer Null-ETL-Integration überprüfen, indem Sie die Systemtabelle SVV_INTEGRATION im Analyseziel abfragen. Wenn die Spalte `state` den Wert `ErrorState` aufweist, bedeutet das, dass ein Fehler vorliegt. Weitere Informationen finden Sie unter the section called “Überwachen mithilfe von Systemtabellen”.

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme bei Null-ETL-Integrationen von Aurora zu beheben.

> **Important**
>
> Resynchronisierungs- und Aktualisierungsvorgänge sind für Null-ETL-Integrationen mit einem Amazon SageMaker AI-Lakehouse nicht verfügbar. Wenn Probleme mit einer Integration auftreten, müssen Sie die Integration löschen und eine neue Integration erstellen. Sie können eine bestehende Integration nicht aktualisieren oder erneut synchronisieren.

Themen

- Ich kann keine Null-ETL-Integration erstellen
- Meine Integration steckt im Status Syncing fest

- Meine Tabellen werden nicht in Amazon Redshift repliziert
- Eine oder mehrere meiner Amazon-Redshift-Tabellen erfordern eine erneute Synchronisation
- Fehlgeschlagene Integration bei Null-ETL-Integrationen von Lakehouse in Amazon SageMaker AI
- DDL-Änderungen werden in Amazon Redshift vorgenommen, bevor die DDL-Transaktion für Aurora PostgreSQL abgeschlossen ist

Ich kann keine Null-ETL-Integration erstellen

Wenn Sie keine Null-ETL-Integration erstellen können, stellen Sie Folgendes für Ihre Quelldatenbank sicher:

- In Ihrer Quelldatenbank muss eine unterstützte DB-Engine-Version ausgeführt werden. Eine Liste der unterstützten Versionen finden Sie unter the section called “Zero-ETL-Integrationen ”.
- Sie haben die DB-Parameter korrekt konfiguriert. Wenn die erforderlichen Parameter falsch festgelegt oder nicht mit der Datenbank verknüpft sind, schlägt die Erstellung fehl. Siehe the section called “Schritt 1: Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe”.

Stellen Sie außerdem Folgendes für Ihr Ziel-Data-Warehouse sicher:

- Die Unterscheidung zwischen Groß- und Kleinschreibung ist aktiviert. Siehe Aktivieren der Unterscheidung zwischen Groß- und Kleinschreibung für Ihr Data Warehouse.
- Sie haben den richtigen autorisierten Prinzipal und die richtige Integrationsquelle hinzugefügt. Weitere Informationen finden Sie unter Konfigurieren der Autorisierung für Ihr Amazon Redshift Data Warehouse.
- Das Data Warehouse ist verschlüsselt (wenn es sich um einen bereitgestellten Cluster handelt). Weitere Informationen finden Sie unter Verschlüsselung von Amazon-Redshift-Datenbanken.

Meine Integration steckt im Status **`Syncing`** fest

Ihre Integration zeigt möglicherweise konsistent den Status `Syncing` an, wenn Sie den Wert eines erforderlichen DB-Parameters ändern.

Um dieses Problem zu beheben, überprüfen Sie die Werte der Parameter in der Parametergruppe, die mit dem Quell-DB-Cluster verknüpft ist, und stellen Sie sicher, dass sie den erforderlichen Werten entsprechen. Weitere Informationen finden Sie unter the section called “Schritt 1: Erstellen einer benutzerdefinierten DB-Cluster-Parametergruppe”.

Wenn Sie Parameter ändern, stellen Sie sicher, dass Sie den DB-Cluster neu starten, damit die Änderungen übernommen werden.

Meine Tabellen werden nicht in Amazon Redshift repliziert

Wenn Sie eine oder mehrere Tabellen in Amazon Redshift nicht sehen, können Sie den folgenden Befehl ausführen, um sie erneut zu synchronisieren:

```
ALTER DATABASE dbname INTEGRATION REFRESH TABLES table1, table2;
```

Weitere Informationen finden Sie unter ALTER DATABASE in der SQL-Referenz zu Amazon Redshift.

Ihre Daten werden möglicherweise nicht repliziert, da eine oder mehrere Quelltabellen nicht über einen Primärschlüssel verfügen. Das Überwachungs-Dashboard in Amazon Redshift zeigt den Status dieser Tabellen als `Failed` an und der Status der gesamten Null-ETL-Integration ändert sich in `Needs attention`. Um dieses Problem zu lösen, können Sie einen vorhandenen Schlüssel in Ihrer Tabelle identifizieren, der zu einem Primärschlüssel werden kann, oder Sie können einen synthetischen Primärschlüssel hinzufügen. Ausführliche Lösungen finden in den folgenden Ressourcen:

- Handle tables without primary keys while creating Amazon Aurora MySQL or Amazon RDS für MySQL zero-ETL integrations with Amazon Redshift
- Handle tables without primary keys while creating Amazon Aurora PostgreSQL zero-ETL integrations with Amazon Redshift

Eine oder mehrere meiner Amazon-Redshift-Tabellen erfordern eine erneute Synchronisation

Wenn Sie bestimmte Befehle auf Ihrer Quelldatenbank ausführen, müssen Ihre Tabellen möglicherweise erneut synchronisiert werden. In diesen Fällen zeigt die Systemansicht SVV_INTEGRATION_TABLE_STATE für `table_state` den Wert `ResyncRequired` an. Dies bedeutet, dass die Integration die Daten für diese spezifische Tabelle vollständig neu von MySQL in Amazon Redshift laden muss.

Wenn die Tabelle mit der erneuten Synchronisation beginnt, wechselt sie in den Status `Syncing`. Sie müssen keine manuellen Maßnahmen ergreifen, um eine Tabelle erneut zu synchronisieren.

Während die Tabellendaten erneut synchronisiert werden, können Sie in Amazon Redshift nicht darauf zugreifen.

Im Folgenden finden Sie einige Beispieloperationen, mit denen eine Tabelle in den Status `ResyncRequired` versetzt werden kann, sowie mögliche Alternativen, die Sie in Betracht ziehen sollten.

Operation	Beispiel	Alternative
Hinzufügen einer Spalte an einer bestimmten Position	`ALTER TABLE table_name ADD COLUMN column_name INTEGER NOT NULL first;`	Amazon Redshift unterstützt nicht das Hinzufügen von Spalten an bestimmten Positionen mithilfe der Schlüsselwörter `first` oder `af` Wenn die Reihenfolge der Spalten in der Zieltabelle nicht wichtig ist, fügen Sie die Spalte mit einem einfacheren Befehl am Ende der Tabelle hinzu: `ALTER TABLE table_name ADD COLUMN column_na`

Operation	Beispiel	Alternative
		`me column_type ;`

Operation	Beispiel	Alternative
Hinzufügen einer Zeitstempelspalte mit dem Standardwert CURRENT_TIMESTAMP	`ALTER TABLE table_name ADD COLUMN column_name TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP;`	Der Wert CURRENT_TIMESTAMP für vorhandene Tabellenzeilen wird von Aurora MySQL berechnet und kann in Amazon Redshift ohne vollständige erneute Synchronisation der Tabellendaten nicht simuliert werden. Wenn möglich, ändern Sie den Standardwert in eine Literalkonstante wie 2023-01-01 00:00:15, um Latenzen bei der Tabellenv

Operation	Beispiel	Alternative
		erfügbarkeit zu vermeiden.
Ausführen mehrerer Spaltenoperationen innerhalb eines einzigen Befehls	`ALTER TABLE table_name ADD COLUMN column_1, RENAME COLUMN column_2 TO column_3;`	Überlegen Sie, ob Sie den Befehl in zwei separate Operationen, ADD und RENAME, aufteilen sollten, die keine erneute Synchronisation erfordern.

Fehlgeschlagene Integration bei Null-ETL-Integrationen von Lakehouse in Amazon SageMaker AI

Wenn Sie Probleme mit einer bestehenden Null-ETL-Integration mit einem Lakehouse in Amazon SageMaker AI haben, besteht die einzige Lösung darin, die Integration zu löschen und neu zu erstellen. Im Gegensatz zu anderen AWS-Services unterstützen Null-ETL-Integrationen keine Aktualisierungs- oder Resynchronisierungsvorgänge.

So lösen Sie Integrationsprobleme:

1. Löschen Sie die problematische Null-ETL-Integration mithilfe der Konsole, CLI oder API.
2. Stellen Sie sicher, dass die Konfiguration der Quelldatenbank und des Ziel-Data-Warehouse korrekt ist.
3. Erstellen Sie eine neue Null-ETL-Integration mit derselben oder einer aktualisierten Konfiguration.

Dieser Prozess führt zu einer vollständigen Neuinitialisierung der Daten-Pipeline, was je nach Größe Ihrer Quelldatenbank einige Zeit in Anspruch nehmen kann.

DDL-Änderungen werden in Amazon Redshift vorgenommen, bevor die DDL-Transaktion für Aurora PostgreSQL abgeschlossen ist

DDL-Änderungen können in Amazon Redshift angezeigt werden, bevor eine DDL-Operation in Null-ETL-Integrationen von Aurora PostgreSQL abgeschlossen ist. Weitere Informationen finden Sie unter DDL-Vorgänge für Aurora PostgreSQL.

Verwenden von Aurora Serverless v2

Aurora Serverless v2 ist eine On-Demand-Konfiguration mit automatischer Skalierung für Amazon Aurora. Aurora Serverless v2 hilft dabei, die Prozesse zur Überwachung der Workload und zur Anpassung der Kapazität Ihrer Datenbanken zu automatisieren. Die Kapazität wird automatisch basierend auf dem Anwendungsbedarf angepasst. Berechnet werden Ihnen nur Ressourcen, die Ihre DB-Cluster verbrauchen. Daher kann Aurora Serverless v2 Ihnen helfen, innerhalb des Budgets zu bleiben und zu vermeiden, für Computerressourcen zu bezahlen, die Sie nicht verwenden.

Diese Art der Automatisierung ist besonders wertvoll für mehrmandantenfähige Datenbanken, verteilte Datenbanken, Entwicklungs- und Testsysteme sowie andere Umgebungen mit sehr variablen und unvorhersehbaren Workloads.

Themen

- Anwendungsfälle für Aurora Serverless v2
- Vorteile von Aurora Serverless v2
- Funktionsweise von Aurora Serverless v2
- Anforderungen und Einschränkungen für Aurora Serverless v2
- Erstellen eines DB-Clusters, der Aurora Serverless v2 verwendet
- Verwalten von Aurora Serverless v2-DB-Clustern
- Performance und Skalierung für Aurora Serverless v2
- Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2
- Migration zu Aurora Serverless v2

Anwendungsfälle für Aurora Serverless v2

Aurora Serverless v2 unterstützt viele verschiedene Datenbank-Workloads. Diese reichen von Entwicklungs- und Testumgebungen über Websites und Anwendungen mit unvorhersehbaren Workloads bis hin zu den anspruchsvollsten, geschäftskritischen Anwendungen, die eine hohe Skalierung und Verfügbarkeit erfordern.

Aurora Serverless v2 ist besonders nützlich für die folgenden Anwendungsfälle:

- Variable Workloads – Sie führen Workloads mit plötzlichen und unvorhersehbaren Aktivitätszuwächsen aus. Ein Beispiel dafür ist eine Verkehrs-Website, für die Aktivitätsspitzen

entstehen, wenn es zu regnen beginnt. Eine andere ist eine E-Commerce-Website mit erhöhtem Datenverkehr, wenn Sie Verkaufs- oder Sonderaktionen anbieten. Mit Aurora Serverless v2 wird Ihre Datenbank automatisch auf die Kapazität skaliert, mit der die Spitzenlast Ihrer Anwendung erfüllt werden kann, und wieder herunterskaliert, wenn die Aktivitätsspitze vorbei ist. Mit Aurora Serverless v2 müssen Sie nicht mehr Spitzen- oder durchschnittliche Kapazitäten bereitstellen. Sie können eine obere Kapazitätsgrenze angeben, um die schlimmste Situation zu bewältigen. Diese Kapazität wird nur genutzt, wenn sie benötigt wird.

Die Granularität der Skalierung in Aurora Serverless v2 hilft Ihnen, die Kapazität genau an die Anforderungen Ihrer Datenbank anzupassen. Für einen bereitgestellten Cluster erfordert die Hochskalierung das Hinzufügen einer ganz neuen DB-Instance. Für ein Aurora Serverless v1-Cluster erfordert die Hochskalierung eine Verdoppelung der Anzahl der Aurora-Kapazitätseinheiten (ACUs) für den Cluster, z. B. von 16 auf 32 oder 32 auf 64. Im Gegensatz dazu kann Aurora Serverless v2 eine halbe ACU hinzufügen, wenn nur ein wenig mehr Kapazität benötigt wird. Es kann 0,5, 1, 1,5, 2 oder zusätzliche halbe ACUs hinzufügen, basierend auf der zusätzlichen Kapazität, die für einen Anstieg der Workload erforderlich ist. Und es kann 0,5, 1, 1,5, 2 oder zusätzliche halbe ACUs entfernen, wenn die Workload abnimmt und diese Kapazität nicht mehr benötigt wird.

- Anwendungen für mehrere Mandanten – Mit Aurora Serverless v2 müssen Sie die Datenbankkapazität nicht für jede Anwendung in Ihrer Flotte einzeln verwalten. Aurora Serverless v2 verwaltet die individuelle Datenbankkapazität für Sie.

  Sie können einen Cluster für jeden Mandanten erstellen. Auf diese Weise können Sie Funktionen wie Klonen, Snapshot-Wiederherstellung und globale Aurora-Datenbanken verwenden, um die hohe Verfügbarkeit und Notfallwiederherstellung für jeden Mandanten zu verbessern.

  Jeder Mandant kann je nach Tageszeit, Jahreszeit, Werbeveranstaltungen usw. bestimmte Perioden mit hoher und geringer Aktivität haben. Jeder Cluster kann einen großen Kapazitätsbereich aufweisen. Auf diese Weise fallen für Cluster mit geringer Aktivität minimale DB-Instance-Gebühren an. Jeder Cluster kann schnell hochskaliert werden, um Perioden mit hoher Aktivität zu bewältigen.
- Neue Anwendungen – Sie stellen eine neue Anwendung bereit und sind sich nicht sicher, welche DB-Instance-Größe Sie benötigen. Durch die Verwendung von Aurora Serverless v2 können Sie einen Cluster mit einer oder mehreren DB-Instances einrichten und es der Datenbank überlassen, gemäß den Kapazitätsanforderungen Ihrer Anwendung automatisch zu skalieren.
- Anwendungen mit gemischter Verwendung – Angenommen, Sie haben eine Anwendung für die Online-Transaktionsverarbeitung (OLTP), erleben aber regelmäßig Spitzen im Abfrageverkehr.

Durch Angabe von Hochstufungsstufen für die Aurora Serverless v2-DB-Instances in einem Cluster können Sie Ihren Cluster so konfigurieren, dass die Reader-DB-Instances unabhängig von der Writer-DB-Instance skalieren können, um die zusätzliche Last zu bewältigen. Wenn die Nutzungsspitze nachlässt, skalieren die Reader-DB-Instances wieder herunter, um der Kapazität der Writer-DB-Instance zu entsprechen.

- Planung der Kapazität – Angenommen, Sie passen normalerweise Ihre Datenbankkapazität an oder überprüfen die optimale Datenbankkapazität für Ihre Workload, indem Sie die DB-Instance-Klassen aller DB-Instances in einem Cluster ändern. Mit Aurora Serverless v2 können Sie diesen Verwaltungsaufwand vermeiden. Sie können die entsprechende minimale und maximale Kapazität ermitteln, indem Sie die Workload ausführen und überprüfen, wie stark die DB-Instances tatsächlich skalieren.

  Sie können vorhandene DB-Instances von bereitgestellt in Aurora Serverless v2 oder von Aurora Serverless v2 in bereitgestellt ändern. In solchen Fällen müssen Sie keinen neuen Cluster und keine neue DB-Instance erstellen.

  Bei einer globalen Aurora-Datenbank benötigen Sie möglicherweise nicht so viel Kapazität für die sekundären Cluster wie im primären Cluster. Sie können Aurora Serverless v2-DB-Instances in den sekundären Clustern verwenden. Auf diese Weise kann die Cluster-Kapazität hochskalieren, wenn eine sekundäre Region hochgestuft wird und die Workload Ihrer Anwendung übernimmt.
- Entwicklung und Tests – Sie können nicht nur Ihre anspruchsvollsten Anwendungen ausführen, sondern Aurora Serverless v2 auch für Entwicklungs- und Testumgebungen verwenden. Mit Aurora Serverless v2 können Sie DB-Instances mit geringer Mindestkapazität erstellen, statt spitzenlastleistungsfähige db.t*-DB-Instance-Klassen zu verwenden. Sie können die maximale Kapazität so hoch einstellen, dass diese DB-Instances weiterhin erhebliche Workloads ausführen können, ohne dass der Speicher knapp wird. Wenn die Datenbank nicht verwendet wird, skalieren alle DB-Instances herunter, um unnötige Gebühren zu vermeiden.

> **Tip**
>
> Damit Aurora Serverless v2 in Entwicklungs- und Testumgebungen bequem verwendet werden kann, bietet die AWS-Managementkonsole die Verknüpfung Easy create (Einfach erstellen), wenn Sie einen neuen Cluster erstellen. Wenn Sie die Option Dev/Test (Entwicklung/Test) auswählen, erstellt Aurora einen Cluster mit einer Aurora Serverless v2-DB-Instance und einem Kapazitätsbereich, der für ein Entwicklungs- und Testsystem typisch ist.

Verwenden von Aurora Serverless v2 für vorhandene bereitgestellte Workloads

Angenommen, Sie haben bereits eine Aurora-Anwendung, die auf einem bereitgestellten Cluster ausgeführt wird. Sie können überprüfen, wie die Anwendung mit Aurora Serverless v2 funktionieren würde, indem Sie dem vorhandenen Cluster eine oder mehrere Aurora Serverless v2-DB-Instances als Reader-DB-Instances hinzufügen. Sie können überprüfen, wie oft die Reader-DB-Instances herauf- und herunterskalieren. Sie können den Aurora-Failover-Mechanismus verwenden, um eine Aurora Serverless v2-DB-Instance zum Writer hochzustufen, und überprüfen, wie sie die Lese-/Schreib-Workload bewältigt. Auf diese Weise können Sie mit minimalen Ausfallzeiten und ohne Ändern des Endpunkts, den Ihre Clientanwendungen verwenden, umstellen. Weitere Informationen zum Verfahren zum Konvertieren vorhandener Cluster in Aurora Serverless v2 finden Sie unter Migration zu Aurora Serverless v2.

Vorteile von Aurora Serverless v2

Aurora Serverless v2 ist für variable oder „Spitzen-“Workloads vorgesehen. Bei solch unvorhersehbaren Workloads haben Sie möglicherweise Schwierigkeiten zu planen, wann Ihre Datenbankkapazität geändert werden soll. Möglicherweise haben Sie auch Probleme, Kapazitätsänderungen schnell genug vorzunehmen, indem Sie die bekannten Mechanismen wie das Hinzufügen von DB-Instances oder das Ändern von DB-Instance-Klassen verwenden. Aurora Serverless v2 bietet die folgenden Vorteile zugunsten solcher Anwendungsfälle:

- Einfacheres Kapazitätsmanagement als bereitgestellt – Aurora Serverless v2 reduziert den Aufwand für die Planung von DB-Instance-Größen und die Größenänderung von DB-Instances, wenn sich die Workload ändert. Es reduziert auch den Aufwand, konsistente Kapazität für alle DB-Instances in einem Cluster aufrechtzuerhalten.
- Schnellere und einfachere Skalierung in Zeiten hoher Aktivität – Aurora Serverless v2 skaliert Rechen- und Speicherkapazität nach Bedarf, ohne dass die Client-Transaktionen oder Ihre gesamte Workload gestört werden. Die Möglichkeit, Reader-DB-Instances mit Aurora Serverless v2 zu verwenden, hilft Ihnen, zusätzlich zur vertikalen Skalierung die horizontale Skalierung zu nutzen. Die Möglichkeit, globale Aurora-Datenbanken zu verwenden, bedeutet, dass Sie Ihre Aurora Serverless v2-Lese-Workload über mehrere AWS-Regionen verteilen können. Diese Fähigkeit ist bequemer als die Skalierungsmechanismen für bereitgestellte Cluster. Sie ist auch schneller und stärker granular als die Skalierungsfunktionen in Aurora Serverless v1.

- Kostengünstig in Zeiten geringer Aktivität – Aurora Serverless v2 hilft Ihnen, eine Überbereitstellung Ihrer DB-Instances zu vermeiden. Aurora Serverless v2 fügt Ressourcen in granularen Schritten hinzu, wenn DB-Instances hochskalieren. Sie zahlen nur für die Datenbankressourcen, die Sie verbrauchen. Die Nutzung der Aurora Serverless v2-Ressourcen wird auf Sekundenbasis abgerechnet. Wenn eine DB-Instance herunterskaliert, wird so sofort die reduzierte Ressourcennutzung registriert.
- Größere Funktionsparität mit bereitgestellt – Sie können viele Aurora-Funktionen mit Aurora Serverless v2 verwenden, die für Aurora Serverless v1 nicht verfügbar sind. Mit Aurora Serverless v2 können Sie z. B. Reader-DB-Instances, globale Datenbanken , AWS Identity and Access Management (IAM)-Datenbankauthentifizierung und Performance Insights verwenden. Sie können auch viel mehr Konfigurationsparameter verwenden als mit Aurora Serverless v1.

  Mit Aurora Serverless v2 können Sie insbesondere die folgenden Funktionen aus bereitgestellten Clustern nutzen:

  - Reader-DB-Instances – Aurora Serverless v2 kann Reader-DB-Instances für die horizontale Skalierung nutzen. Wenn ein Cluster eine oder mehrere Reader-DB-Instances enthält, kann der Cluster bei Problemen mit der Writer-DB-Instance sofort ein Failover ausführen. Diese Fähigkeit ist mit Aurora Serverless v1 nicht verfügbar.
  - Multi-AZ-Cluster – Sie können die Aurora Serverless v2-DB-Instances eines Clusters auf mehrere Availability Zones (AZs) verteilen. Die Einrichtung eines Multi-AZ-Clusters trägt dazu bei, die Geschäftskontinuität auch in seltenen Fällen von Problemen sicherzustellen, die eine gesamte AZ betreffen. Diese Fähigkeit ist mit Aurora Serverless v1 nicht verfügbar.
  - Globale Datenbanken – Sie können Aurora Serverless v2 in Kombination mit globalen Aurora-Datenbanken verwenden, um zusätzliche schreibgeschützte Kopien Ihres Clusters in anderen AWS-Regionen für Notfallwiederherstellungszwecke zu erstellen.
  - RDS Proxy – Mit Amazon RDS Proxy können Sie Ihren Anwendungen erlauben, Datenbankverbindungen zu bündeln und gemeinsam zu nutzen, um ihre Skalierbarkeit zu verbessern.
- Schnellere, stärker granulare, weniger störende Skalierung als Aurora Serverless v1 – Aurora Serverless v2 kann schneller hoch- und herunterskalieren. Die Skalierung kann die Kapazität um nur 0,5 ACU ändern, anstatt die Anzahl der ACU zu verdoppeln oder zu halbieren. Die Skalierung erfolgt normalerweise ohne Unterbrechung der Verarbeitung. Die Skalierung umfasst im Gegensatz zu Aurora Serverless v1 kein Ereignis, das Sie beachten müssen. Die Skalierung kann erfolgen, während SQL-Anweisungen ausgeführt werden und Transaktionen geöffnet sind, ohne auf einen Ruhepunkt warten zu müssen.

Funktionsweise von Aurora Serverless v2

Die folgende Übersicht beschreibt die Funktionsweise von Aurora Serverless v2.

Themen

- Übersicht über Aurora Serverless v2
- Konfigurationen für Aurora-DB-Cluster
- Kapazität von Aurora Serverless v2
- Aurora Serverless v2-Skalierung
- Aurora Serverless v2 und hohe Verfügbarkeit
- Aurora Serverless v2 und Speicher
- Konfigurationsparameter für Aurora-Cluster

Übersicht über Aurora Serverless v2

Amazon Aurora Serverless v2 eignet sich für die anspruchsvollsten, sehr variablen Workloads. Zum Beispiel könnte Ihre Datenbanknutzung für einen kurzen Zeitraum sehr hoch sein, gefolgt von langen Zeiträumen mit nur leichter oder gar keiner Aktivität. Beispiele hierfür sind Einzelhandels-Websites, Spiele oder Sportveranstaltungen mit regelmäßigen Werbeveranstaltungen sowie Datenbanken, die nach Bedarf Berichte erstellen. Andere Beispiele sind Entwicklungs- und Testumgebungen sowie neue Anwendungen, in denen die Nutzung schnell ansteigen könnte. In Fällen wie diesen und vielen anderen ist es mit dem bereitgestellten Modell nicht immer möglich, die Kapazität vorab korrekt zu konfigurieren. Es können auch höhere Kosten entstehen, wenn Sie zu viel Kapazität bereitstellen, die Sie dann nicht benötigen.

Im Gegensatz dazu sind von Aurora bereitgestellte Cluster für stetige Workloads geeignet. Bei bereitgestellten Clustern wählen Sie eine DB-Instance-Klasse mit einer vordefinierten Menge an Speicher, CPU-Leistung, I/O-Bandbreite usw aus. Wenn sich Ihre Workload ändert, ändern Sie die Instance-Klasse Ihres Writers und Ihrer Readers manuell. Das bereitgestellte Modell funktioniert gut, wenn Sie die Kapazität im Vorfeld der erwarteten Verbrauchsmustern anpassen können. Es ist akzeptabel, dass kurze Ausfälle auftreten, während Sie die Instance-Klasse des Writers und der Reader in Ihrem Cluster ändern.

Aurora Serverless v2 ist von Grund auf so konzipiert, dass serverlose DB-Cluster unterstützt werden, die sofort skalierbar sind. Aurora Serverless v2 wurde entwickelt, um das gleiche Maß an Sicherheit und Isolation zu bieten wie mit bereitgestellten Writern und Readern. Diese Aspekte

sind in serverlosen Cloud-Umgebungen mit mehreren Mandanten entscheidend. Der dynamische Skalierungsmechanismus umfasst sehr wenig Aufwand, sodass er schnell auf Änderungen der Datenbank-Workload reagieren kann. Er bietet auch genügend Leistung, um einen drastischen Anstieg beim Bedarf nach Rechenleistung zu meistern.

Durch die Verwendung von Aurora Serverless v2 können Sie einen Aurora-DB-Cluster erstellen, ohne für jeden Writer und Reader an eine bestimmte Datenbankkapazität gebunden zu sein. Sie geben den minimalen und maximalen Bereich für die Kapazität an. Aurora skaliert jeden Aurora Serverless v2-Writer oder -Reader im Cluster innerhalb dieses Kapazitätsbereichs. Durch die Verwendung eines Multi-AZ-Clusters, in dem jeder Writer oder Reader dynamisch skalieren kann, können Sie die dynamische Skalierung und Hochverfügbarkeit nutzen.

Aurora Serverless v2 skaliert die Datenbankressourcen automatisch basierend auf Ihren minimalen und maximalen Kapazitätsspezifikationen. Die Skalierung ist schnell, da die meisten Skalierungsereignisoperationen den Writer oder Reader auf demselben Host halten. In seltenen Fällen, in denen ein Aurora Serverless v2-Writer oder -Reader von einem Host zu einem anderen verschoben wird, verwaltet Aurora Serverless v2 die Verbindungen automatisch. Sie müssen Ihren Datenbank-Client-Anwendungscode oder Ihre Datenbank-Verbindungszeichenfolgen nicht ändern.

Wie bei bereitgestellten Clustern sind bei Aurora Serverless v2 Speicherkapazität und Rechenkapazität getrennt. Wenn wir uns auf Aurora Serverless v2-Kapazität und -Skalierung beziehen, geht es immer um die Rechenkapazität, die zunimmt oder abnimmt. Somit kann Ihr Cluster viele Terabyte an Daten enthalten, auch wenn die CPU und Speicherkapazität auf ein niedriges Niveau herunterskalieren.

Anstelle der Bereitstellung und Verwaltung von Datenbankservern geben Sie die Datenbankkapazität an. Details zur Aurora Serverless v2-Kapazität finden Sie unter Kapazität von Aurora Serverless v2. Die tatsächliche Kapazität jedes Aurora Serverless v2-Writers oder -Readers variiert im Laufe der Zeit, je nach Workload. Details zu diesem Mechanismus finden Sie unter Aurora Serverless v2-Skalierung.

> **Important**
>
> Mit Aurora Serverless v1 verfügt Ihr Cluster über eine einzige Maßeinheit für die Rechenkapazität, die zwischen den minimalen und maximalen Kapazitätswerten skaliert werden kann. Mit Aurora Serverless v2 kann Ihr Cluster zusätzlich zum Writer auch Reader enthalten. Jeder Aurora Serverless v2-Writer und -Reader kann zwischen minimalen und maximalen Kapazitätswerten skalieren. Somit hängt die Gesamtkapazität Ihres Aurora Serverless v2-Clusters sowohl vom Kapazitätsbereich ab, den Sie für Ihren DB-Cluster

definieren, als auch von der Anzahl der Writer und Reader im Cluster. Zu jedem Zeitpunkt wird Ihnen nur die Aurora Serverless v2-Kapazität in Rechnung gestellt, die in Ihrem Aurora-DB-Cluster aktiv genutzt wird.

Konfigurationen für Aurora-DB-Cluster

Für jeden Ihrer Aurora DB-Cluster können Sie eine beliebige Kombination von Aurora Serverless v2-Kapazität, bereitgestellter Kapazität oder beides auswählen.

Sie können einen Cluster einrichten, der beides enthält, Aurora Serverless v2 und bereitgestellte Kapazität. Dies wird als Cluster mit gemischter Konfiguration bezeichnet. Nehmen Sie zum Beispiel an, dass Sie mehr Lese- und Schreibkapazität benötigen, als für einen Aurora Serverless v2-Writer verfügbar ist. In diesem Fall können Sie den Cluster mit einem sehr großen bereitgestellten Writer einrichten. Sie können in diesem Fall immer noch Aurora Serverless v2 für die Reader verwenden. Oder nehmen Sie an, dass die Schreib-Workload für Ihren Cluster variiert, die Lese-Workload jedoch stabil ist. In diesem Fall können Sie Ihren Cluster mit einem Aurora Serverless v2-Writer und einem oder mehreren bereitgestellten Readern einrichten.

Sie können auch einen DB-Cluster einrichten, bei dem die gesamte Kapazität von Aurora Serverless v2 verwaltet wird. Dazu können Sie einen neuen Cluster erstellen und von Anfang an Aurora Serverless v2 verwenden. Oder Sie können die gesamte bereitgestellte Kapazität in einem vorhandenen Cluster durch Aurora Serverless v2 ersetzen. Einige der Upgrade-Pfade älterer Engine-Versionen erfordern z. B., mit einem bereitgestellten Writer zu beginnen und ihn durch einen Aurora Serverless v2-Writer zu ersetzen. Informationen zu den Verfahren zum Erstellen eines neuen DB-Clusters mit Aurora Serverless v2 oder zum Wechseln eines vorhandenen DB-Cluster auf Aurora Serverless v2 finden Sie unter Erstellen eines Aurora Serverless v2-DB Clusters und Umstellung von einem bereitgestellten Cluster zu Aurora Serverless v2.

Wenn Sie Aurora Serverless v2 in einem DB-Cluster überhaupt nicht verwenden, sind alle Writer und Reader im DB-Cluster bereitgestellt. Dies ist die älteste und gebräuchlichste Art von DB-Cluster, mit der die meisten Benutzer vertraut sind. Vor Einführung von Aurora Serverless gab es tatsächlich keinen besonderen Namen für diese Art von Aurora-DB-Cluster. Die bereitgestellte Kapazität ist konstant. Die Gebühren sind relativ einfach zu prognostizieren. Sie müssen jedoch im Vorfeld prognostizieren, wie viel Kapazität Sie benötigen. In einigen Fällen sind Ihre Prognosen möglicherweise ungenau oder Ihr Kapazitätsbedarf kann sich ändern. In diesen Fällen kann Ihr DB-Cluster unterdimensioniert (langsamer als gewünscht) oder überdimensioniert (teurer als gewünscht) sein.

Kapazität von Aurora Serverless v2

Die Maßeinheit für Aurora Serverless v2 ist die Aurora-Kapazitätseinheit (ACU). Die Kapazität von Aurora Serverless v2 ist nicht an die DB-Instance-Klassen gebunden, die Sie für bereitgestellte Cluster verwenden.

Jede ACU ist eine Kombination aus etwa 2 Gibibyte (GiB) Arbeitsspeicher, entsprechender CPU und Netzwerkleistung. Mit dieser Maßeinheit geben Sie den Kapazitätsbereich der Datenbank an. Die Metriken `ServerlessDatabaseCapacity` und `ACUUtilization` helfen Ihnen festzustellen, wie viel Kapazität Ihre Datenbank tatsächlich nutzt und wo diese Kapazität innerhalb des angegebenen Bereichs liegt.

Zu jedem Zeitpunkt verfügt jeder Aurora Serverless v2-DB-Writer oder -Reader über eine Kapazität. Die Kapazität wird als Gleitkommazahl dargestellt, die ACUs repräsentiert. Die Kapazität steigt oder nimmt ab, sobald der Writer oder Reader skaliert. Dieser Wert wird jede Sekunde gemessen. Für jeden DB-Cluster, für den Sie Aurora Serverless v2 verwenden möchten, definieren Sie einen Kapazitätsbereich: Die minimalen und maximalen Kapazitätswerte, zwischen denen jeder Aurora Serverless v2-Writer oder -Reader skalieren kann. Der Kapazitätsbereich ist für jeden Aurora Serverless v2-Writer oder -Reader in einem DB-Cluster gleich. Jeder Aurora Serverless v2-Writer oder -Reader hat eine eigene Kapazität, die in diesen Bereich fällt.

In der folgenden Tabelle werden die Aurora Serverless v2-Kapazitätsbereiche und die Unterstützung der Engine-Versionen für Aurora MySQL und Aurora PostgreSQL gezeigt.

Kapazitätsbereich (ACUs)	Unterstützte Versionen für Aurora MySQL	Unterstützte Versionen für Aurora PostgreSQL
0,5–128	3.02.0 und höher	13.6 und höher, 14.3 und höher, 15.2 und höher, 16.1 und höher
0,5–256	3.06.0 und höher	13.13 und höher, 14.10 und höher, 15.5 und höher, 16.1 und höher
0–256	3.08.0 und höher	13.15 und höher, 14.12 und höher, 15.7 und höher, 16.3 und höher

Die folgende Tabelle zeigt die Aurora Serverless v2-Plattformversionen mit ihren ACU-Bereichen und Leistungsmerkmalen.

Aurora Serverless v2-Plattformversion	ACU-Bereich	Leistung
1	0–128	Basisleistung
2	0–256	Basisleistung
3	0–256	Bis zu 30 % verbesserte Leistung im Vergleich zu Plattformversion 2

> **Note**
>
> Der verfügbare Skalierungsbereich für einen bestimmten Cluster wird sowohl von der Engine-Version als auch von der Plattformversion bestimmt. Es ist möglich, eine leistungsstärkere Engine-Version auf einer weniger leistungsstarken Plattformversion auszuführen und umgekehrt. Der Skalierungsbereich wird durch die Engine- oder Plattformversion mit der niedrigsten Kapazität bestimmt.

Sie können im Abschnitt zur Instance-Konfiguration der AWS-Managementkonsole oder über die API feststellen, auf welcher Plattformversion Ihr Cluster läuft, indem Sie sich `ServerlessV2PlatformVersion` für einen DBCluster ansehen.

Die kleinste Aurora Serverless v2-Kapazität, die Sie definieren können, sind 0 ACUs für Aurora Serverless v2-Versionen, die die Auto-Pause-Funktion unterstützen. Sie können eine größere Zahl angeben, wenn sie kleiner oder gleich Ihrem maximalen Kapazitätswert ist. Wenn Sie die Mindestkapazität auf eine kleine Zahl festlegen, können leicht geladene DB-Cluster minimale Rechenressourcen verbrauchen. Gleichzeitig bleiben sie bereit, Verbindungen sofort anzunehmen und hochzuskalieren, wenn ihre Aktivität ansteigt.

Wir empfehlen, das Minimum auf einen Wert festzulegen, der es jedem DB-Writer oder -Reader ermöglicht, den Arbeitssatz der Anwendung im Pufferpool zu halten. Auf diese Weise wird der Inhalt des Pufferpools in Zeiten geringer Aktivität nicht verworfen. Alle Überlegungen zur Auswahl des minimalen Kapazitätswerts finden Sie unter Auswählen der minimalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster. Alle Überlegungen zur Auswahl des maximalen Kapazitätswerts finden Sie unter Auswählen der maximalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster.

Je nachdem, wie Sie die Reader in einer Multi-AZ-Bereitstellung konfigurieren, können die Kapazitäten an die Kapazität des Writers gebunden oder davon unabhängig sein. Weitere Informationen zur Vorgehensweise finden Sie unter Aurora Serverless v2-Skalierung.

Die Überwachung von Aurora Serverless v2 umfasst die Messung der Kapazitätswerte für den Writer und die Reader in Ihrem DB-Cluster im Zeitverlauf. Wenn Ihre Datenbank nicht auf die Mindestkapazität herunterskaliert, können Sie Maßnahmen ergreifen, z. B. den minimalen Wert anpassen und Ihre Datenbankanwendung optimieren. Wenn Ihre Datenbank ihre maximale Kapazität konsequent erreicht, können Sie Maßnahmen wie die Erhöhung des maximalen Werts ergreifen. Sie können auch Ihre Datenbankanwendung optimieren und die Abfragelast auf mehr Reader verteilen.

Die Gebühren für die Aurora Serverless v2-Kapazität werden in Form von ACU-Stunden gemessen. Informationen darüber, wie Aurora Serverless v2-Gebühren berechnet werden, finden Sie auf der Seite Aurora-Preisübersicht.

Nehmen wir an, dass die Gesamtzahl der Writer und Reader in Ihrem Cluster *n* beträgt. In diesem Fall verbraucht der Cluster ungefähr *n* x *minimum ACUs*, wenn Sie keine Datenbankoperationen ausführen. Aurora selbst führt möglicherweise Überwachungs- oder Wartungsvorgänge durch, die eine geringe Last verursachen. Dieser Cluster verbraucht nicht mehr als *n* x *maximum ACUs*, wenn die Datenbank mit voller Kapazität läuft.

Weitere Informationen zur Auswahl geeigneter minimaler und maximaler ACU-Werte finden Sie unter Auswählen des Aurora Serverless v2-Kapazitätsbereichs für einen Aurora-Cluster. Die minimalen und maximalen ACU-Werte, die Sie angeben, wirken sich auch auf die Funktionsweise einiger Aurora-Konfigurationsparameter für Aurora Serverless v2 aus. Weitere Informationen zur Wechselwirkung zwischen dem Kapazitätsbereich und den Konfigurationsparametern finden Sie unter Arbeiten mit Parametergruppen für Aurora Serverless v2.

Aurora Serverless v2-Skalierung

Für jeden Aurora Serverless v2-Writer oder -Reader verfolgt Aurora kontinuierlich die Auslastung von Ressourcen wie CPU, Arbeitsspeicher und Netzwerk. Diese Messungen werden zusammen als Last bezeichnet. Die Last umfasst die von Ihrer Anwendung ausgeführten Datenbankoperationen. Außerdem umfasst sie auch die Hintergrundverarbeitung für den Datenbankserver und Aurora-Verwaltungsaufgaben. Wenn die Kapazität durch einen dieser Faktoren eingeschränkt wird, wird Aurora Serverless v2 hochskaliert. Aurora Serverless v2 skaliert auch hoch, wenn Leistungsprobleme erkannt werden, die dadurch gelöst werden können. Sie können die Ressourcenauslastung und deren Auswirkung auf die Aurora Serverless v2-Skalierung überwachen, indem Sie die Verfahren

in Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2 und Überwachung der Aurora Serverless v2-Leistung mit Performance Insights verwenden.

Die Last kann je nach Writer und Reader in Ihrem DB-Cluster variieren. Der Writer verarbeitet alle DDL-Anweisungen (Data Definition Language), wie `CREATE TABLE` ,`ALTER TABLE` und `DROP TABLE`. Außerdem verarbeitet der Writer alle DML-Anweisungen (Data Manipulation Language), wie `INSERT` und `UPDATE`. Reader können schreibgeschützte Anweisungen verarbeiten, z. B. `SELECT`-Abfragen.

Skalierung ist die Operation, die die Aurora Serverless v2-Kapazität für Ihre Datenbank erhöht oder senkt. Mit Aurora Serverless v2 verfügt jeder Writer und Reader über einen eigenen aktuellen Kapazitätswert, der in ACUs gemessen wird. Aurora Serverless v2 skaliert einen Writer oder Reader auf eine höhere Kapazität, wenn die aktuelle Kapazität zu niedrig ist, um die Last zu bewältigen. Der Writer oder Reader wird auf eine geringere Kapazität skaliert, wenn seine aktuelle Kapazität höher als erforderlich ist.

Im Gegensatz zu Aurora Serverless v1, bei der skaliert wird, indem die Kapazität bei jedem Erreichen eines Schwellenwerts verdoppelt wird, kann Aurora Serverless v2 die Kapazität inkrementell erhöhen. Wenn Ihr Workload-Bedarf die aktuelle Datenbankkapazität eines Writers oder Readers erreicht, erhöht Aurora Serverless v2 die Anzahl der ACUs für diesen Writer oder Reader. Aurora Serverless v2 skaliert die Kapazität in den inkrementellen Schritten, die erforderlich sind, um die beste Leistung für die genutzten Ressourcen zu bieten. Die Skalierung erfolgt in inkrementellen Schritten ab 0,5 ACU. Je größer die aktuelle Kapazität, desto größer ist das Skalierungsinkrement und desto schneller kann die Skalierung erfolgen.

Da die Aurora Serverless v2-Skalierung so häufig, granular und unterbrechungsfrei auftritt, werden keine diskreten Ereignisse in der AWS-Managementkonsole wie bei Aurora Serverless v1 verursacht. Stattdessen können Sie die Amazon-CloudWatch-Metriken wie `ServerlessDatabaseCapacity` und `ACUUtilization` messen und ihre minimalen, maximalen und durchschnittlichen Werte im Zeitverlauf verfolgen. Weitere Informationen über Aurora-Metriken finden Sie unter Überwachung von Metriken in einem Amazon-Aurora-Cluster. Tipps zur Überwachung von Aurora Serverless v2 finden Sie unter Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2.

Sie können wählen, ob ein Reader gleichzeitig mit dem zugehörigen Writer oder unabhängig vom Writer skalieren soll. Dazu geben Sie die Hochstufungsstufe für diesen Reader an.

- Reader in den Hochstufungsstufen 0 und 1 skalieren gleichzeitig mit dem Writer. Dieses Skalierungsverhalten macht Reader in den Prioritätsstufen 0 und 1 ideal verfügbar. Das liegt daran,

dass sie immer auf die richtige Kapazität dimensioniert sind, um die Workload des Writers im Falle eines Failovers zu übernehmen.

- Reader der Hochstufungsstufen 2 bis 15 skalieren unabhängig vom Writer. Jeder Reader bleibt innerhalb der minimalen und maximalen ACU-Werte, die Sie für Ihren Cluster angegeben haben. Wenn ein Reader unabhängig von der zugehörigen Writer-DB skaliert, kann er inaktiv werden und herunterskalieren, während der Writer weiterhin ein hohes Transaktionsvolumen verarbeitet. Er ist nach wie vor als Failover-Ziel verfügbar, wenn keine anderen Reader in niedrigeren Hochstufungsstufen zur Verfügung stehen. Wenn der Reader jedoch zum Writer hochgestuft wird, muss er möglicherweise hochskalieren, um die volle Workload des Writers zu bewältigen.

Weitere Informationen zu Hochstufungsstufen finden Sie unter Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader.

Die Begriffe Skalierungpunkte und zugehörige Timeout-Perioden von Aurora Serverless v1 finden in Aurora Serverless v2 keine Anwendung. Die Aurora Serverless v2-Skalierung kann auftreten, während Datenbankverbindungen geöffnet sind, während SQL-Transaktionen in Bearbeitung sind, während Tabellen gesperrt sind und während temporäre Tabellen verwendet werden. Aurora Serverless v2 wartet nicht auf einen Ruhepunkt, um mit der Skalierung zu beginnen. Die Skalierung unterbricht keine laufenden Datenbankoperationen.

Wenn Ihre Workload mehr Lesekapazität benötigt, als mit einem einzigen Writer und einem einzigen Reader verfügbar ist, können Sie dem Cluster mehrere Aurora Serverless v2-Reader hinzufügen. Jeder Aurora Serverless v2-Reader kann innerhalb der minimalen und maximalen Kapazitätswerte skalieren, die Sie für Ihren DB-Cluster angegeben haben. Sie können den Reader-Endpunkt des Clusters verwenden, um schreibgeschützte Sitzungen an die Reader zu leiten und die Last des Writers zu reduzieren.

Ob Aurora Serverless v2 die Skalierung durchführt und wie schnell die Skalierung nach dem Start erfolgt, hängt auch von den minimalen und maximalen ACU-Einstellungen für den Cluster ab. Darüber hinaus hängt es davon ab, ob ein Reader so konfiguriert ist, dass er zusammen mit dem Writer oder unabhängig davon skaliert wird. Details zu den Faktoren, die Auswirkungen auf die Aurora Serverless v2-Skalierung haben, finden Sie unter Performance und Skalierung für Aurora Serverless v2.

Skalierung auf 0

In den aktuellen Aurora-MySQL- und Aurora-PostgreSQL-Versionen können Aurora Serverless v2-Writer und -Reader auf 0 ACU herunterskalieren. Wir bezeichnen diese Funktion als automatisches

Anhalten und Fortsetzen oder Auto-Pause. Sie können wählen, ob Sie dieses Verhalten zulassen möchten, indem Sie als Mindestkapazität einen Wert von 0 oder einen Wert ungleich 0 angeben. Sie können auch wählen, wie lange gewartet werden soll, bis eine Aurora Serverless v2-Instance angehalten wird. Informationen darüber, welche Versionen über diese Funktion verfügen, finden Sie unter Kapazität von Aurora Serverless v2. Informationen zur effektiven Verwendung finden Sie unter Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2.

In älteren Aurora-MySQL- und Aurora-PostgreSQL-Versionen können inaktive Aurora Serverless v2-Writer und -Reader auf den minimalen ACU-Wert herunterskalieren, den Sie für den Cluster angegeben haben, aber nicht auf 0 ACUs. In diesem Fall stehen 0 ACUs bei der Festlegung des Kapazitätsbereichs nicht als Auswahlmöglichkeit zur Verfügung. Dieses Verhalten ist anders als bei Aurora Serverless v1, bei der nach einer Zeit der Inaktivität eine Pause erfolgen kann, aber dann einige Zeit benötigt wird, den Betrieb fortzusetzen, wenn Sie eine neue Verbindung öffnen.

Wenn Ihr DB-Cluster mit Aurora Serverless v2-Kapazität einige Zeit nicht benötigt wird, können Sie auch den gesamten Cluster genau wie bereitgestellte DB-Cluster anhalten und starten. Diese Technik eignet sich am besten für Entwicklungs- und Testsysteme, bei denen die Cluster möglicherweise viele Stunden am Stück nicht benötigt werden und die Geschwindigkeit, mit der sie fortgesetzt werden, nicht entscheidend ist. Die Funktion zum Stoppen/Starten von Clustern ist für alle Aurora Serverless v2-Versionen verfügbar. Weitere Informationen über diese Funktion finden Sie unter Stoppen und Starten eines Amazon Aurora-DB-Clusters.

Aurora Serverless v2 und hohe Verfügbarkeit

Hohe Verfügbarkeit für einen Aurora-DB-Cluster wird erreicht, indem Sie ihn zu einem Multi-AZ-DB-Cluster hochstufen. Ein Multi-AZ-Aurora-DB-Cluster verfügt jederzeit über Rechenkapazität in mehr als einer Availability Zone (AZ). Diese Konfiguration hält Ihre Datenbank auch im Falle eines größeren Ausfalls betriebsbereit. Aurora führt ein automatisches Failover im Falle eines Problems durch, das den Writer oder sogar die gesamte AZ betrifft. Mit Aurora Serverless v2 können Sie wählen, ob die Standby-Rechenkapazität zusammen mit der Kapazität des Schreibers hoch- oder herunterskaliert werden soll. Auf diese Weise ist die Rechenkapazität in der zweiten AZ bereit, die aktuelle Workload jederzeit zu übernehmen. Gleichzeitig kann die Rechenkapazität in allen AZs herunterskalieren, wenn die Datenbank inaktiv ist. Weitere Informationen darüber, wie Aurora mit AWS-Regionen und Availability Zones funktioniert, finden Sie unter Hohe Verfügbarkeit für Aurora-DB-Instances.

Die Aurora Serverless v2-Multi-AZ-Fähigkeit verwendet Reader zusätzlich zum Writer. Die Unterstützung für Reader ist neu für Aurora Serverless v2 im Gegensatz zu Aurora Serverless v1. Sie

können bis zu 15 Aurora Serverless v2-Reader hinzufügen, die sich über 3 AZs auf einen Aurora-DB-Cluster verteilen.

Für geschäftskritische Anwendungen, die auch im Falle eines Problems, das Ihren gesamten Cluster oder die gesamte AWS-Region betrifft, verfügbar bleiben müssen, können Sie eine globale Aurora-Datenbank einrichten. Sie können Aurora Serverless v2-Kapazität in den sekundären Clustern verwenden, damit sie im Falle einer Notfallwiederherstellung zur Übernahme bereit sind. Sie können auch herunterskalieren, wenn die Datenbank nicht ausgelastet ist. Weitere Informationen zu globalen Aurora-Datenbanken finden Sie unter Verwenden von Amazon Aurora Global Database.

Aurora Serverless v2 funktioniert wie für Failover und andere Hochverfügbarkeitsfunktionen bereitgestellt. Weitere Informationen finden Sie unter Hohe Verfügbarkeit für Amazon Aurora.

Angenommen, Sie möchten maximale Verfügbarkeit für Ihre Aurora Serverless v2-Cluster sicherstellen. Sie können zusätzlich zum Writer einen Reader erstellen. Wenn Sie dem Reader die Hochstufungsstufe 0 oder 1 zuweisen, geschieht jede Skalierung sowohl für den Writer als auch für den Reader. Auf diese Weise ist ein Reader mit identischer Kapazität im Falle eines Failovers immer zur Übernahme für den Writer bereit.

Angenommen, Sie möchten Quartalsberichte für Ihr Unternehmen erstellen, während Ihr Cluster weiterhin Transaktionen verarbeitet. Wenn Sie dem Cluster einen Aurora Serverless v2-Reader hinzufügen und ihm eine Hochstufungsstufe zwischen 2 und 15 zuweisen, können Sie sich direkt mit diesem Reader verbinden, um die Berichte zu erstellen. Abhängig davon, wie arbeitsspeicherintensiv und CPU-intensiv die Berichtsabfragen sind, kann dieser Reader hochskalieren, um die Workload zu bewältigen. Anschließend kann er wieder herunterskalieren, wenn die Berichte erstellt sind.

Aurora Serverless v2 und Speicher

Der Speicher für jeden Aurora-DB-Cluster besteht aus sechs Kopien aller Ihrer Daten, die auf drei AZs verteilt sind. Diese integrierte Datenreplikation gilt unabhängig davon, ob Ihr DB-Cluster zusätzlich zum Writer auch Reader enthält. Auf diese Weise sind Ihre Daten sicher, auch bei Problemen, die sich auf die Rechenkapazität des Clusters auswirken.

Aurora Serverless v2-Speicher hat die gleichen Zuverlässigkeits- und Haltbarkeitseigenschaften wie unter Amazon-Aurora-Speicher beschrieben. Das liegt daran, dass der Speicher für Aurora-DB-Cluster identisch funktioniert, unabhängig davon, ob die Rechenkapazität Aurora Serverless v2 verwendet oder bereitgestellt wird.

Konfigurationsparameter für Aurora-Cluster

Sie können Cluster- und Datenbankkonfigurationsparameter für Cluster mit Aurora Serverless v2-Kapazität genauso anpassen wie für bereitgestellte DB-Clustern. Einige kapazitätsbezogene Parameter werden für Aurora Serverless v2 jedoch unterschiedlich behandelt. In einem Cluster mit gemischter Konfiguration gelten die Parameterwerte, die Sie für diese kapazitätsbezogenen Parameter angeben, weiterhin für bereitgestellte Writer und Reader.

Fast alle Parameter funktionieren für Aurora Serverless v2-Writer und Reader auf die gleiche Weise wie für bereitgestellte Writer und Reader. Ausnahmen sind einige Parameter, die Aurora während der Skalierung automatisch anpasst, und einige Parameter, die von Aurora als feste Werte beibehalten werden, die von der maximalen Kapazitätseinstellung abhängen.

Beispielsweise erhöht sich die für den Puffer-Cache reservierte Speichermenge, wenn ein Writer oder Reader hochskaliert, und nimmt beim Herunterskalieren ab. Auf diese Weise kann Arbeitsspeicher freigegeben werden, wenn Ihre Datenbank nicht ausgelastet ist. Umgekehrt legt Aurora die maximale Anzahl von Verbindungen automatisch auf einen Wert fest, der basierend auf der maximalen Kapazitätseinstellung angemessen ist. Auf diese Weise werden aktive Verbindungen nicht unterbrochen, wenn die Last sinkt und Aurora Serverless v2 herunterskaliert. Weitere Informationen zum Umgang von Aurora Serverless v2 mit bestimmten Parametern finden Sie unter Arbeiten mit Parametergruppen für Aurora Serverless v2.

Anforderungen und Einschränkungen für Aurora Serverless v2

Beachten Sie beim Erstellen eines Clusters, mit dem Sie Aurora Serverless v2-DB-Instances verwenden möchten, die folgenden Anforderungen und Einschränkungen.

Themen

- Verfügbarkeit von Regionen und Versionen
- Für Cluster, die Aurora Serverless v2 verwenden, muss ein Kapazitätsbereich angegeben sein
- Einige bereitgestellte Funktionen werden in Aurora Serverless v2 nicht unterstützt
- Einige Aurora Serverless v2-Aspekte unterscheiden sich von Aurora Serverless v1

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Aurora-Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur

Verfügbarkeit von Versionen und Regionen im Zusammenhang mit Aurora und Aurora Serverless v2 finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2.

Im folgenden Beispiel finden Sie die AWS CLI-Befehle zur Bestätigung der genauen DB-Engine-Werte, die Sie mit Aurora Serverless v2 für eine bestimmte AWS-Region verwenden können. Der `--db-instance-class`-Parameter für Aurora Serverless v2 ist immer `db.serverless`. Der `--engine`-Parameter kann `aurora-mysql` oder `aurora-postgresql` sein. Ersetzen Sie die entsprechenden Werte `--region` und `--engine` zur Bestätigung der `--engine-version`-Werte, die Sie verwenden können. Wenn der Befehl keine Ausgabe erzeugt, ist Aurora Serverless v2 für diese Kombination aus AWS-Region und DB-Engine nicht verfügbar.

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-
class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text
```

Für Cluster, die Aurora Serverless v2 verwenden, muss ein Kapazitätsbereich angegeben sein

Ein Aurora-Cluster muss über ein `ServerlessV2ScalingConfiguration`-Attribut verfügen, bevor Sie DB-Instances hinzufügen können, die die `db.serverless`-DB-Instance-Klasse verwenden. Dieses Attribut gibt den Kapazitätsbereich an. Die Aurora Serverless v2-Kapazität reicht von mindestens 0 Aurora-Kapazitätseinheiten (ACUs) bis 256 ACUs in inkrementellen Schritten von 0,5 ACUs. Der zulässige Mindestwert hängt von der Aurora-Version ab. Jede ACU bietet das Äquivalent von ungefähr 2 Gibibyte (GiB) RAM und zugehörige CPU- und Netzwerkressourcen. Details darüber, wie Aurora Serverless v2 die Einstellungen des Kapazitätsbereichs verwendet, finden Sie unter Funktionsweise von Aurora Serverless v2.

Die zulässigen Kapazitätsbereiche für verschiedene DB-Engine- und Plattformversionen finden Sie unter Kapazität von Aurora Serverless v2. Der verfügbare Skalierungsbereich für einen bestimmten Cluster wird sowohl von der Engine-Version als auch von der Hardware (Plattformversion) bestimmt.

Sie können die minimalen und maximalen ACU-Werte in der AWS-Managementkonsole angeben, wenn Sie einen Cluster und eine damit verbundene Aurora Serverless v2-DB-Instance erstellen. Sie

können auch die `--serverless-v2-scaling-configuration`-Option in der AWS CLI angeben. Oder Sie können den `ServerlessV2ScalingConfiguration`-Parameter mit der Amazon-RDS-API angeben. Sie können dieses Attribut angeben, wenn Sie einen Cluster erstellen oder einen vorhandenen Cluster ändern. Informationen zu den Verfahren zum Festlegen des Kapazitätsbereichs finden Sie unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster. Eine ausführliche Diskussion darüber, wie minimale und maximale Kapazitätswerte ausgewählt werden und wie sich diese Einstellungen auf einige Datenbankparameter auswirken, finden Sie unter Auswählen des Aurora Serverless v2-Kapazitätsbereichs für einen Aurora-Cluster.

Einige bereitgestellte Funktionen werden in Aurora Serverless v2 nicht unterstützt

Die folgenden Funktionen der von Aurora bereitgestellten DB-Instances sind derzeit für Amazon Aurora Serverless v2 nicht verfügbar:

- Datenbankaktivitäts-Streams (DAS)
- Cluster-Cacheverwaltung für Aurora PostgreSQL. Der `apg_ccm_enabled`-Konfigurationsparameter gilt nicht für Aurora Serverless v2-DB-Instances.

Einige Aurora-Funktionen arbeiten mit Aurora Serverless v2, können jedoch Probleme verursachen, wenn Ihr Kapazitätsbereich niedriger ist als für die Speicheranforderungen dieser Funktionen mit Ihrer spezifischen Workload erforderlich. In diesem Fall funktioniert Ihre Datenbank möglicherweise nicht so gut wie gewohnt oder es kann zu Fehlern wegen zu wenig Speicherplatz kommen. Empfehlungen zum Einstellen des entsprechenden Kapazitätsbereichs finden Sie unter Auswählen des Aurora Serverless v2-Kapazitätsbereichs für einen Aurora-Cluster. Informationen zur Fehlerbehebung, wenn für Ihre Datenbank aufgrund eines falsch konfigurierten Kapazitätsbereichs Fehler wegen zu wenig Speicherplatz auftreten, finden Sie unter Vermeiden von Fehlern aufgrund von zu wenig Speicher.

Aurora Auto Scaling wird nicht unterstützt. Diese Art der Skalierung fügt basierend auf der CPU-Auslastung neue Reader hinzu, um zusätzliche leseintensive Workloads zu bewältigen. Die Skalierung basierend auf der CPU-Auslastung ist jedoch für Aurora Serverless v2 nicht sinnvoll. Alternativ können Sie Aurora Serverless v2-Reader-DB-Instances im Vorfeld erstellen und sie auf geringe Kapazität herunterskaliert lassen. Dies ist eine schnellere und weniger störende Methode, um die Lesekapazität eines Clusters zu skalieren, als neue DB-Instances dynamisch hinzuzufügen.

Einige Aurora Serverless v2-Aspekte unterscheiden sich von Aurora Serverless v1

Wenn Sie Aurora Serverless v1-Benutzer sind und Aurora Serverless v2 zum ersten Mal verwenden, sehen Sie sich die Unterschiede zwischen Aurora Serverless v2- und Aurora Serverless v1-Anforderungen an, um zu verstehen, inwieweit sich die Anforderungen zwischen Aurora Serverless v1 und Aurora Serverless v2 unterscheiden.

Erstellen eines DB-Clusters, der Aurora Serverless v2 verwendet

Wenn Sie einen Aurora-Cluster erstellen möchten, dem Sie Aurora Serverless v2-DB-Instances hinzufügen können, folgen Sie dem gleichen Verfahren wie in Erstellen eines Amazon Aurora-DB Clusters. Mit Aurora Serverless v2 sind Ihre Cluster mit bereitgestellten Clustern austauschbar. Es sind Cluster möglich, in denen einige DB-Instances Aurora Serverless v2 verwenden und andere DB-Instances bereitgestellt werden.

Themen

- Einstellungen für Aurora Serverless v2-DB-Cluster
- Erstellen eines Aurora Serverless v2-DB Clusters
- Erstellen einer Aurora Serverless v2-Writer-DB-Instance

Einstellungen für Aurora Serverless v2-DB-Cluster

Stellen Sie sicher, dass die Anfangseinstellungen des Clusters die unter Anforderungen und Einschränkungen für Aurora Serverless v2 aufgeführten Anforderungen erfüllen. Geben Sie die folgenden Einstellungen an, um sicherzustellen, dass Sie dem Cluster Aurora Serverless v2-DB-Instances hinzufügen können:

AWS-Region

Erstellen Sie den Cluster in einer AWS-Region, in der Aurora Serverless v2-DB-Instances verfügbar sind. Einzelheiten zu verfügbaren Regionen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2.

DB-Engine-Version

Wählen Sie eine Engine-Version aus, die mit Aurora Serverless v2 kompatibel ist. Weitere Informationen über die Aurora Serverless v2-Versionsanforderungen finden Sie unter Anforderungen und Einschränkungen für Aurora Serverless v2.

DB-Instance-Klasse (DB instance class)

Wenn Sie einen Cluster mit der AWS-Managementkonsole erstellen, wählen Sie gleichzeitig die DB-Instance-Klasse für die Writer-DB-Instance aus. Wählen Sie die DB-Instance-Klasse Serverless aus. Wenn Sie diese DB-Instance-Klasse auswählen, geben Sie auch den Kapazitätsbereich für die Writer-DB-Instance an. Derselbe Kapazitätsbereich gilt für alle anderen Aurora Serverless v2-DB-Instances, die Sie diesem Cluster hinzufügen.

Wenn die Option Serverless für die DB-Instance-Klasse nicht angezeigt wird, stellen Sie sicher, dass Sie eine DB-Engine-Version ausgewählt haben, die für Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2 unterstützt wird.

Wenn Sie die AWS CLI oder die Amazon-RDS-API verwenden, lautet der Parameter, den Sie für die DB-Instance-Klasse angeben, `db.serverless`.

Capacity range (Kapazitätsbereich)

Geben Sie die Werte der minimalen und maximalen Aurora-Kapazitätseinheit (ACU) ein, die für alle DB-Instances im Cluster gelten. Diese Option ist sowohl auf der Seite Create cluster (Cluster erstellen) als auch auf der Seite Add reader (Reader hinzufügen) der Konsole verfügbar, wenn Sie Serverless für die DB-Instance-Klasse auswählen.

Die zulässigen Kapazitätsbereiche für verschiedene DB-Engine-Versionen finden Sie unter Kapazität von Aurora Serverless v2.

Falls die Felder für minimale und maximale ACUs nicht angezeigt werden, überprüfen Sie, ob Sie die DB-Instance-Klasse Serverless für die Writer-DB-Instance ausgewählt haben.

Wenn Sie den Cluster zunächst mit einer bereitgestellten DB-Instance erstellen, geben Sie nicht die minimalen und maximalen ACUs an. In diesem Fall können Sie den Cluster später ändern und diese Einstellung hinzufügen. Sie können dem Cluster auch eine Aurora Serverless v2-Reader-DB-Instance hinzufügen. Im Rahmen dieses Prozesses geben Sie den Kapazitätsbereich an.

Bis Sie den Kapazitätsbereich für Ihren Cluster angegeben haben, können Sie dem Cluster keine Aurora Serverless v2-DB-Instances über die AWS CLI oder die RDS API hinzufügen. Wenn

Sie versuchen, eine Aurora Serverless v2-DB-Instance hinzuzufügen, wird eine Fehlermeldung ausgegeben. In den Verfahren der AWS CLI oder der RDS API wird der Kapazitätsbereich durch das Attribut `ServerlessV2ScalingConfiguration` dargestellt.

Bei Clustern, die mehr als eine Reader-DB-Instance enthalten, spielt die Failover-Priorität jeder Aurora Serverless v2-Reader-DB-Instance eine wichtige Rolle dabei, wie diese DB-Instance hoch- oder herunterskaliert. Sie können die Priorität nicht angeben, wenn Sie den Cluster erstellen. Denken Sie an diese Eigenschaft, wenn Sie Ihrem Cluster eine zweite oder weitere Reader-DB-Instance hinzufügen. Weitere Informationen finden Sie unter Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader.

Erstellen eines Aurora Serverless v2-DB Clusters

Sie können einen Aurora Serverless v2-DB-Cluster über die AWS-Managementkonsole, die AWS CLI oder die RDS-API erstellen.

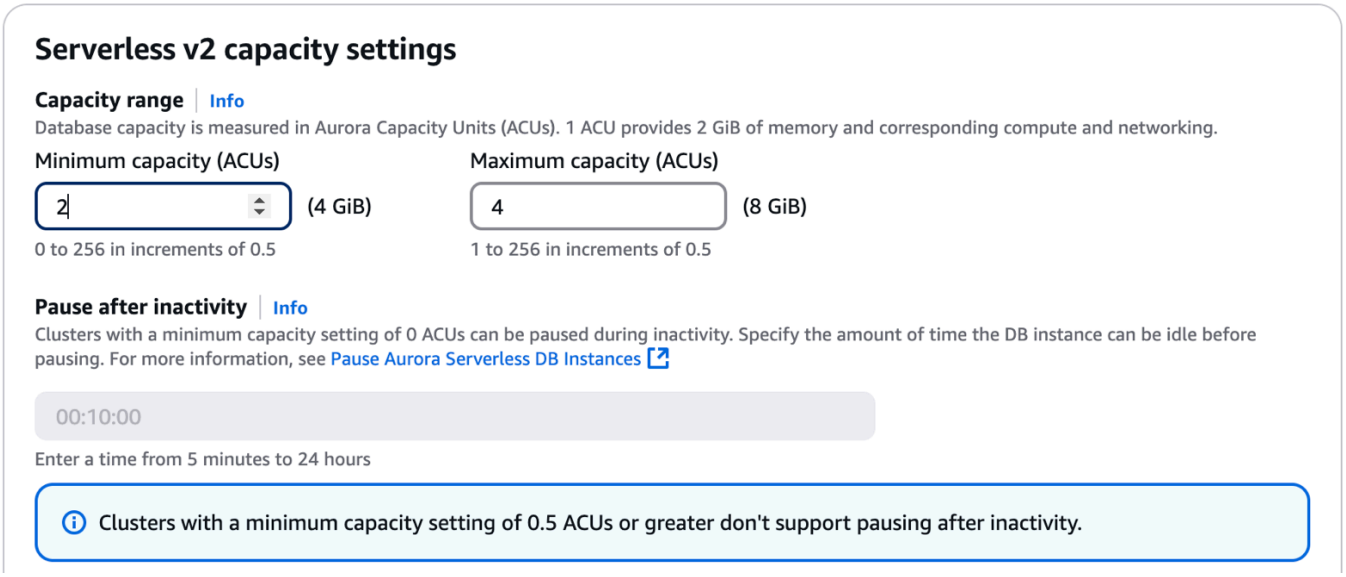
Konsole

Erstellen Sie einen Cluster mit einem Aurora Serverless v2-Writer wie folgt

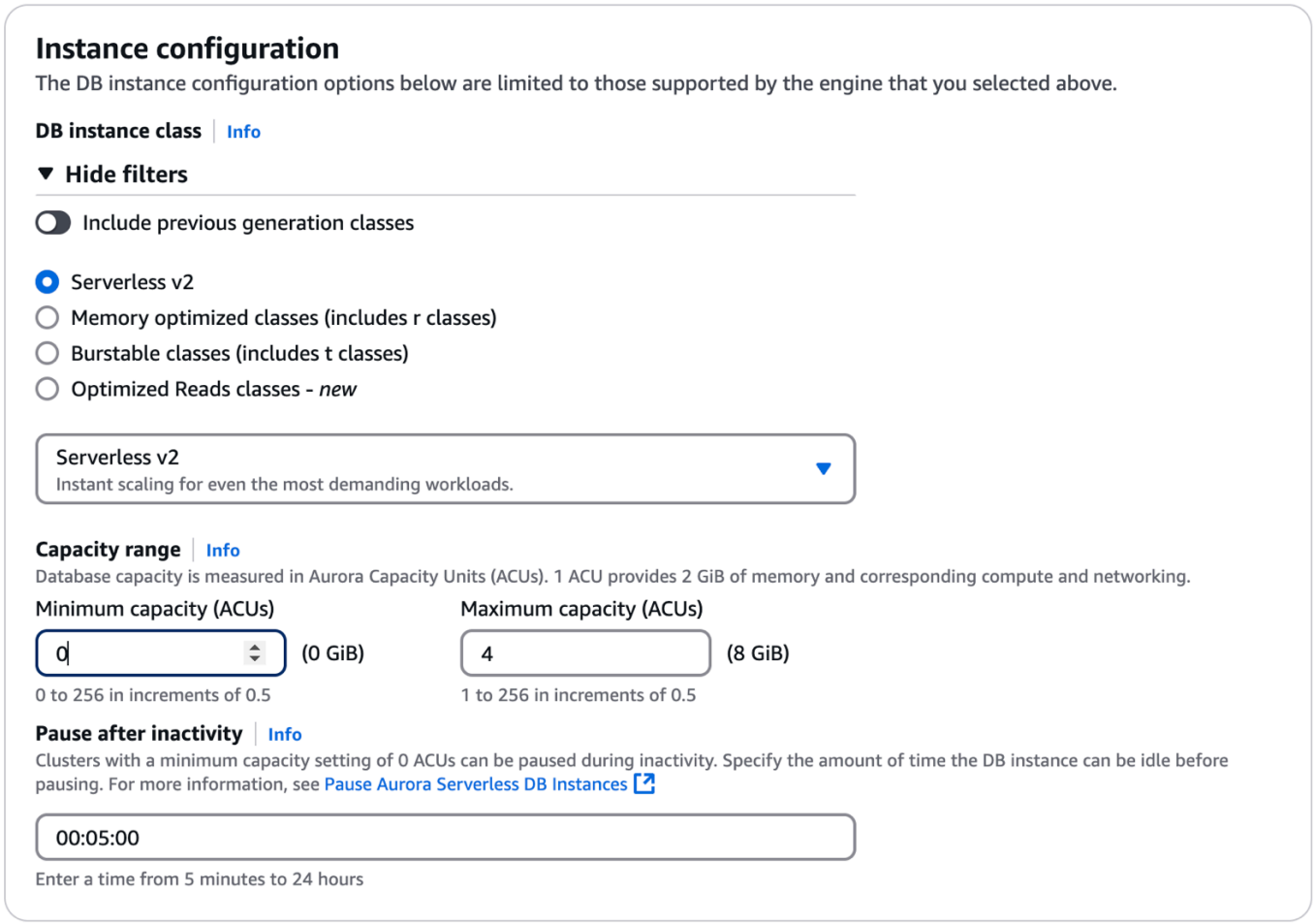
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie Create database (Datenbank erstellen) aus. Wählen Sie auf der angezeigten Seite die folgenden Optionen aus:
   - Wählen Sie als Engine-Typ Aurora (MySQL-kompatibel) oder Aurora (PostgreSQL-kompatibel).
   - Wählen Sie unter Version eine der unterstützten Versionen für Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2 aus.
4. Wählen Sie für DB-Instance-Klasse die Option Serverless v2 aus.
5. Unter Kapazitätsbereich können Sie den Standardbereich übernehmen. Sie können auch andere Mindest- und Höchstwerte für die Kapazitätseinheiten festlegen. Sie können zwischen mindestens 0 ACU bis maximal 256 ACU in Schritten von 0,5 ACUs wählen.

   Weitere Informationen zu Aurora Serverless v2-Kapazitätseinheiten finden Sie unter Kapazität von Aurora Serverless v2 und Performance und Skalierung für Aurora Serverless v2.

Je nach ausgewählter Engine und Version kann die Obergrenze 128 ACUs und/oder die Untergrenze 0,5 ACUs sein. Einzelheiten zum Limit für jede Kombination aus Aurora-Engine und -Version finden Sie unter Kapazität von Aurora Serverless v2.



Wenn Sie eine Mindestkapazität von 0 ACUs auswählen, wird die Aurora Serverless v2-Funktion zum automatischen Anhalten und Fortsetzen aktiviert. In diesem Fall können Sie zusätzlich festlegen, wie lange die Aurora Serverless v2-DB-Instances ohne Datenbankverbindungen warten, bevor sie automatisch angehalten werden. Informationen zur Funktion zum automatischen Anhalten und Fortsetzen finden Sie unter Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2.



6. Wählen Sie andere DB-Clustereinstellungen aus, wie unter Einstellungen für Aurora-DB-Cluster beschrieben.
7. Wählen Sie Datenbank erstellen aus, um Ihren Aurora-DB-Cluster mit einer Aurora Serverless v2-DB-Instance als Writer-Instance (auch als primäre DB-Instance bezeichnet) zu erstellen.

CLI

Wenn Sie einen DB-Cluster mit der AWS CLI erstellen möchten, der mit Aurora Serverless v2-DB-Instances kompatibel ist, folgen Sie dem CLI-Verfahren unter Erstellen eines Amazon Aurora-DB Clusters. Fügen Sie die folgenden Parameter in Ihren `create-db-cluster`-Befehl ein:

- --region *AWS_Region_where_Aurora Serverless v2_instances_are_available*
- --engine-version *serverless_v2_compatible_engine_version*
- --serverless-v2-scaling-configuration MinCapacity=*minimum_capacity*,MaxCapacity=*maximum_capacity*

Das folgende Beispiel zeigt, wie ein Aurora Serverless v2-DB-Cluster erstellt wird.

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.04.1 \
    --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
    --master-username myuser \
    --manage-master-user-password
```

> **Note**
>
> Wenn Sie einen DB-Cluster von Aurora Serverless v2 mit AWS CLI erstellen, erscheint der Engine-Modus in der Ausgabe als `provisioned` anstatt `serverless`. Der Engine-Modus `serverless` bezieht sich auf Aurora Serverless v1.

In diesem Beispiel wird die Option `--manage-master-user-password` zum Generieren des Administratorpassworts und zum Verwalten dieses Passworts in Secrets Manager angegeben. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager. Alternativ können Sie die Option `--master-password` verwenden, um das Passwort selbst festzulegen und zu verwalten.

Weitere Informationen über die Aurora Serverless v2-Versionsanforderungen finden Sie unter Anforderungen und Einschränkungen für Aurora Serverless v2. Informationen über die zulässigen Werte für den Kapazitätsbereich und was diese Werte darstellen, finden Sie unter Kapazität von Aurora Serverless v2 und Performance und Skalierung für Aurora Serverless v2.

Wenn Sie überprüfen möchten, ob ein vorhandener Cluster die angegebenen Kapazitätseinstellungen hat, sehen Sie sich die Ausgabe des `describe-db-clusters`-Befehls für das `ServerlessV2ScalingConfiguration`-Attribut an. Dieses Attribut sieht ähnlich wie folgt aus.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> Wenn Sie beim Erstellen des Clusters die minimalen und maximalen ACU-Werte nicht angeben, können Sie diese Einstellungen später mit dem `modify-db-cluster`-Befehl hinzufügen. Bis dahin können Sie dem Cluster keine Aurora Serverless v2-DB-Instances hinzufügen. Wenn Sie versuchen, eine `db.serverless`-DB-Instance hinzuzufügen, wird eine Fehlermeldung ausgegeben.

API

Wenn Sie einen DB-Cluster mit der RDS API erstellen möchten, der mit Aurora Serverless v2-DB-Instances kompatibel ist, folgen Sie dem API-Verfahren unter Erstellen eines Amazon Aurora-DB Clusters. Wählen Sie die folgenden Einstellungen aus. Stellen Sie sicher, dass die `CreateDBCluster`-Operation die folgenden Parameter enthält:

```
EngineVersion serverless_v2_compatible_engine_version
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and
 MaxCapacity=maximum_capacity
```

Weitere Informationen über die Aurora Serverless v2-Versionsanforderungen finden Sie unter Anforderungen und Einschränkungen für Aurora Serverless v2. Informationen über die zulässigen Werte für den Kapazitätsbereich und was diese Werte darstellen, finden Sie unter Kapazität von Aurora Serverless v2 und Performance und Skalierung für Aurora Serverless v2.

Wenn Sie überprüfen möchten, ob ein vorhandener Cluster die angegebenen Kapazitätseinstellungen hat, sehen Sie sich die Ausgabe der `DescribeDBClusters`-Operation für das `ServerlessV2ScalingConfiguration`-Attribut an. Dieses Attribut sieht ähnlich wie folgt aus.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> Wenn Sie beim Erstellen des Clusters die minimalen und maximalen ACU-Werte nicht angeben, können Sie diese Einstellungen später mit der `ModifyDBCluster`-Operation hinzufügen. Bis dahin können Sie dem Cluster keine Aurora Serverless v2-DB-Instances

hinzufügen. Wenn Sie versuchen, eine `db.serverless`-DB-Instance hinzuzufügen, wird eine Fehlermeldung ausgegeben.

Erstellen einer Aurora Serverless v2-Writer-DB-Instance

Sie geben zwar den Aurora Serverless v2-Kapazitätsbereich an, wenn Sie einen Aurora-Cluster erstellen, Sie können jedoch für jede DB-Instance im Cluster wählen, ob Sie Aurora Serverless v2 oder eine Bereitstellung verwenden möchten. Um sofort mit der Verwendung von Aurora Serverless v2 in Ihrem DB-Cluster zu beginnen, fügen Sie eine Writer-DB-Instance hinzu, die die `db.serverless`-Instance-Klasse verwendet. In der Konsole treffen Sie diese Auswahl in der Regel als Teil des Workflows zur Erstellung des DB-Clusters. Daher gilt dieses Verfahren hauptsächlich, wenn Sie das Setup über die CLI durchführen.

Konsole

Wenn Sie einen DB-Cluster mit der AWS-Managementkonsole erstellen, geben Sie gleichzeitig die Eigenschaften der Writer-DB-Instance an. Damit die Writer-DB-Instance Aurora Serverless v2 verwendet, wählen Sie die DB-Instance-Klasse Serverless aus.

Anschließend legen Sie den Kapazitätsbereich für den Cluster fest, indem Sie die Werte für die minimale und maximale Aurora-Kapazitätseinheit (ACU) angeben. Diese minimalen und maximalen Werte gelten für jede Aurora Serverless v2-DB-Instance im Cluster. Dieses Verfahren und Informationen zur Bedeutung der Einstellungen für die minimale und maximale Kapazität finden Sie unter Erstellen eines Aurora Serverless v2-DB Clusters.

Wenn Sie bei der Ersterstellung des Clusters keine Aurora Serverless v2-DB-Instance erstellen, können Sie später eine oder mehrere Aurora Serverless v2-DB-Instances hinzufügen. Befolgen Sie hierzu die Verfahren unter Hinzufügen eines Aurora Serverless v2-Readers und Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2. Sie geben den Kapazitätsbereich zu dem Zeitpunkt an, an dem Sie dem Cluster die erste Aurora Serverless v2-DB-Instance hinzufügen. Sie können den Kapazitätsbereich später ändern, indem Sie die Schritte unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster befolgen.

CLI

Wenn Sie einen DB-Cluster von Aurora Serverless v2 mit der AWS CLI erstellen, fügen Sie die Writer-DB-Instance mit dem Befehl create-db-instance explizit hinzu. Schließen Sie den folgenden Parameter ein:

- `--db-instance-class db.serverless`

Das folgende Beispiel zeigt, wie eine Writer-DB-Instance von Aurora Serverless v2 erstellt wird.

```
aws rds create-db-instance \
    --db-cluster-identifier my-serverless-v2-cluster \
    --db-instance-identifier my-serverless-v2-instance \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

Verwalten von Aurora Serverless v2-DB-Clustern

Mit Aurora Serverless v2 sind Ihre Cluster mit bereitgestellten Clustern austauschbar. Die Aurora Serverless v2-Eigenschaften gelten für eine oder mehrere DB-Instances innerhalb eines DB-Clusters. Daher sind die Verfahren zum Erstellen von Clustern, zum Ändern von Clustern, zum Erstellen und Wiederherstellen von Snapshots usw. im Grunde die gleichen wie bei anderen Arten von Aurora-Clustern. Allgemeine Verfahren zum Verwalten von Aurora-Clustern und DB-Instances finden Sie unter Verwalten eines Amazon Aurora-DB-Clusters.

In den folgenden Themen erfahren Sie mehr über die Überlegungen im Hinblick auf die Verwaltung von Clustern, die Aurora Serverless v2-DB-Instances enthalten.

Themen

- Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster
- Überprüfen des Kapazitätsbereichs für Aurora Serverless v2
- Hinzufügen eines Aurora Serverless v2-Readers
- Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2
- Konvertieren eines Aurora Serverless v2-Writers oder -Readers in bereitgestellt
- Anhalten von Aurora Serverless v2-DB-Instances
- Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader
- Verwenden von TLS/SSL mit Aurora Serverless v2
- Anzeigen von Aurora Serverless v2-Writern und -Readern
- Protokollierung für Aurora Serverless v2

Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster

Wenn Sie die Konfigurationsparameter oder andere Einstellungen für Cluster mit Aurora Serverless v2-DB-Instances oder die DB-Instances selbst ändern möchten, folgen Sie denselben allgemeinen Verfahren wie für bereitgestellte Cluster. Details hierzu finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Die wichtigste Einstellung, die für Aurora Serverless v2 einzigartig ist, ist der Kapazitätsbereich. Nachdem Sie die Werte für die minimale und maximale Aurora-Kapazitätseinheit (ACU) für einen Aurora-Cluster festgelegt haben, müssen Sie die Kapazität der Aurora Serverless v2-DB-Instances im Cluster aktiv anpassen. Aurora übernimmt diesen Schritt nicht. Diese Einstellung wird auf Cluster-Ebene verwaltet. Für jede Aurora Serverless v2-DB-Instance im Cluster gelten die gleichen minimalen und maximalen ACU-Werte.

Sie können die folgenden spezifischen Werte festlegen:

- Minimum ACUs (Minimale ACUs) – Die Aurora Serverless v2-DB-Instance kann die Kapazität auf diese Anzahl von ACUs reduzieren.
- Maximum ACUs (Maximale ACUs) – Die Aurora Serverless v2-DB-Instance kann die Kapazität auf diese Anzahl von ACUs erhöhen.

Der verfügbare Kapazitätsbereich für Aurora Serverless v2 ist sowohl von der DB-Engine-Version als auch von der Plattformversion abhängig. Neuere DB-Engine-Versionen ermöglichen eine maximale Kapazität von 256 ACUs, eine Mindestkapazität von 0 ACUs oder beides. Die Kapazitätsbereiche für verschiedene DB-Engine-Versionen finden Sie unter Kapazität von Aurora Serverless v2.

Informationen zur Funktion zum automatischen Anhalten und Fortsetzen, die durch Festlegen der Mindestkapazität auf 0 ACUs aktiviert wird, finden Sie unter Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2.

Weitere Informationen dazu, wie Sie sicherstellen können, dass Ihre Aurora Serverless v2-DB-Cluster auf bis zu 256 ACUs skaliert werden können, finden Sie unter Aktualisieren des Aurora Serverless v2-DB-Clusters, um eine Skalierung auf 256 ACUs zu ermöglichen.

> **Note**
>
> Wenn Sie den Kapazitätsbereich für einen DB-Cluster von Aurora Serverless v2 ändern, erfolgt die Änderung sofort, unabhängig davon, ob Sie sie sofort oder während des nächsten geplanten Wartungsfensters anwenden möchten.

In Aurora Serverless v2 können Sie die aktuelle Kapazität nicht direkt festlegen, ohne den Kapazitätsbereich zu ändern, da die Kapazität dynamisch an die Workload innerhalb des angegebenen Bereichs angepasst wird. Sie können jedoch die Kapazität auf eine der folgenden Arten indirekt beeinflussen:

- Mindestkapazität anpassen – Senken Sie die Mindestkapazität vorübergehend, um die Basiskapazität bei geringen Workloads zu reduzieren.
- Skalierung vorübergehend unterbrechen – Um die Kapazität auf einen bestimmten Wert zu fixieren, legen Sie die Mindest- und Höchstkapazität auf denselben Wert fest.
- Bei hoher Auslastung proaktiv skalieren – Wenn Sie mit hohen Workloads rechnen, erhöhen Sie vorab die Mindestkapazität, um in Zeiten hoher Aktivität einen höheren Ausgangswert aufrechtzuerhalten.

Weitere Informationen zu den Auswirkungen des Kapazitätsbereichs und zur Überwachung und Feinabstimmung finden Sie unter Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2 und Performance und Skalierung für Aurora Serverless v2. Ihr Ziel ist es, sicherzustellen, dass die maximale Kapazität für den Cluster hoch genug ist, um Spitzen bei der Workload zu bewältigen, und die miminale Kapazität niedrig genug ist, um die Kosten zu minimieren, wenn der Cluster nicht aktiv ist.

Angenommen, Sie bestimmen basierend auf Ihrer Überwachung, dass der ACU-Bereich für den Cluster höher, niedriger, breiter oder schmaler sein sollte. Sie können die Kapazität eines Aurora-Clusters mit der AWS-Managementkonsole, der AWS CLI oder der Amazon RDS API auf einen bestimmten Bereich von ACUs festlegen. Dieser Kapazitätsbereich gilt für jede Aurora Serverless v2-DB-Instance im Cluster.

Angenommen, Ihr Cluster hat einen Kapazitätsbereich von 1 bis 16 ACUs und enthält zwei Aurora Serverless v2-DB-Instances. Dann verbraucht der Cluster insgesamt zwischen 2 ACUs (bei Inaktivität) und 32 ACUs (bei Vollauslastung). Wenn Sie den Kapazitätsbereich von 8 in 40,5 ACUs

ändern, verbraucht der gesamte Cluster jetzt 16 ACUs bei Inaktivität und bei voller Auslastung bis zu 81 ACUs.

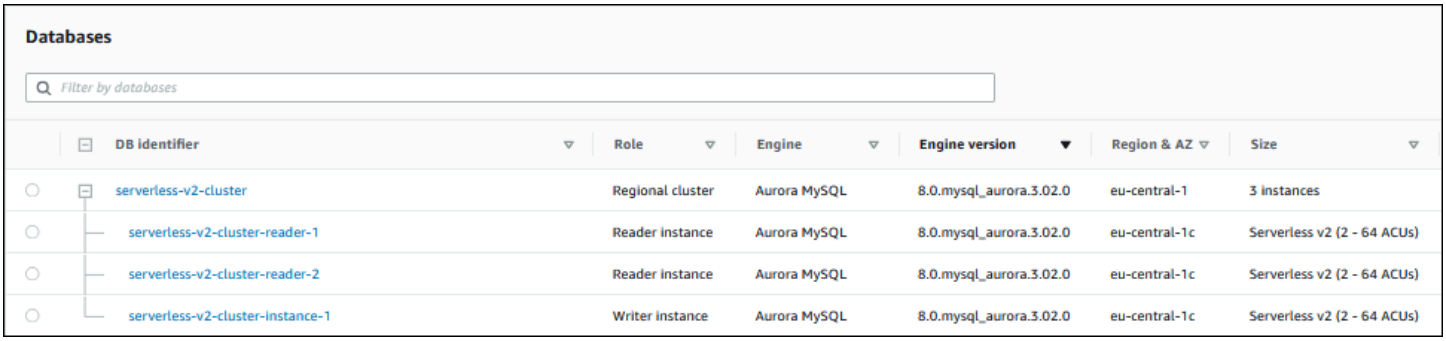
Aurora legt automatisch bestimmte Parameter für Aurora Serverless v2-DB-Instances auf Werte fest, die vom maximalen ACU-Wert im Kapazitätsbereich abhängen. Eine Liste dieser Parameter finden Sie unter Maximale Anzahl der Verbindungen für Aurora Serverless v2. Bei statischen Parametern, die von dieser Berechnungsart abhängen, wird der Wert beim Neustart der DB-Instance erneut ausgewertet. So können Sie den Wert solcher Parameter aktualisieren, indem Sie die DB-Instance neu starten, nachdem Sie den Kapazitätsbereich geändert haben. Wenn Sie wissen möchten, ob Sie Ihre DB-Instance neu starten müssen, um Parameteränderungen zu übernehmen, überprüfen Sie das `ParameterApplyStatus`-Attribut der DB-Instance. Der Wert `pending-reboot` gibt an, dass ein Neustart Änderungen auf einige Parameterwerte anwendet.

Konsole

Sie können den Kapazitätsbereich eines Clusters, der Aurora Serverless v2-DB-Instances enthält, über die AWS-Managementkonsole festlegen.

Wenn Sie die Konsole verwenden, legen Sie den Kapazitätsbereich für den Cluster zu dem Zeitpunkt fest, zu dem Sie dem Cluster die erste Aurora Serverless v2-DB-Instance hinzufügen. Diesen Schritt können Sie ausführen, wenn Sie beim Erstellen des Clusters die DB-Instance-Klasse Serverless v2 für die Writer-DB-Instance auswählen. Sie können diesen Schritt auch ausführen, wenn Sie beim Hinzufügen einer Aurora Serverless v2-Reader-DB-Instance zum Cluster die DB-Instance-Klasse Serverless auswählen. Möglich ist dieser Schritt auch, wenn Sie eine vorhandene bereitgestellte DB-Instance im Cluster in die DB-Instance-Klasse Serverless konvertieren. Die ausführliche Version dieser Verfahren finden Sie unter Erstellen einer Aurora Serverless v2-Writer-DB-Instance, Hinzufügen eines Aurora Serverless v2-Readers und Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2.

Der jeweilige Kapazitätsbereich, den Sie auf Cluster-Ebene festlegen, gilt für alle Aurora Serverless v2-DB-Instances in Ihrem Cluster. Die folgende Abbildung zeigt einen Cluster mit mehreren Aurora Serverless v2-Reader-DB-Instances. Jede Instance hat einen identischen Kapazitätsbereich von 2 bis 64 ACU.

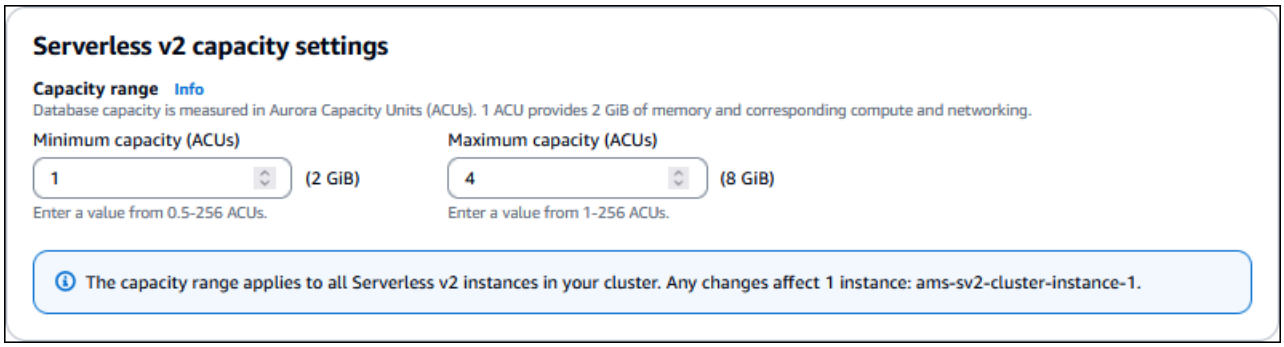


So ändern Sie den Kapazitätsbereich eines Aurora Serverless v2-Clusters

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Cluster, der Ihre Aurora Serverless v2-DB-Instances enthält, aus der Liste aus. Der Cluster muss bereits mindestens eine Aurora Serverless v2-DB-Instance enthalten. Andernfalls zeigt Aurora den Abschnitt Capacity range (Kapazitätsbereich) nicht an.
4. Wählen Sie für Actions (Aktionen) die Option Modify (Ändern) aus.
5. Wählen Sie im Abschnitt Capacity range (Kapazitätsbereich) Folgendes aus:

   a. Geben Sie einen Wert für Minimum ACUs (Minimale ACUs) ein. Die Konsole zeigt den zulässigen Wertebereich an. Sie können eine Mindestkapazität zwischen 0 und 256 ACUs wählen. Sie können eine maximale Kapazität zwischen 1 und 256 ACUs wählen. Sie können die Kapazitätswerte in Schritten von 0,5 ACUs anpassen. Der verfügbare Kapazitätsbereich ist sowohl von der DB-Engine-Version als auch von der Plattformversion abhängig.

   b. Geben Sie einen Wert für Maximum ACUs (Maximale ACUs) ein. Dieser Wert muss gleich oder größer als der Wert für Minimum ACUs (Minimale ACUs) sein. Die Konsole zeigt den zulässigen Wertebereich an. In der folgenden Abbildung wird diese Auswahl veranschaulicht.



6. Klicken Sie auf Weiter. Die Seite Summary of modifications (Zusammenfassung der Änderungen) wird angezeigt.
7. Wählen Sie Apply immediately (Sofort anwenden) aus.

   Die Kapazitätsänderung erfolgt sofort, unabhängig davon, ob Sie sie sofort oder während des nächsten geplanten Wartungsfensters anwenden möchten.
8. Wählen Sie Modify cluster (Cluster ändern), um die Zusammenfassung der Änderungen zu akzeptieren. Sie können auch Back (Zurück) wählen, um Ihre Änderungen zu bearbeiten, oder Cancel (Abbrechen), um Ihre Änderungen zu verwerfen.

AWS CLI

Wenn Sie die Kapazität eines Clusters festlegen möchten, für den Sie Aurora Serverless v2-DB-Instances mit der AWS CLIverwenden möchten, führen Sie den AWS CLI-Befehl modify-db-cluster aus. Legen Sie die Option `--serverless-v2-scaling-configuration` fest. Der Cluster enthält möglicherweise bereits eine oder mehrere Aurora Serverless v2-DB-Instances oder Sie können die DB-Instances später hinzufügen. Gültige Werte für die Felder `MinCapacity` und `MaxCapacity` sind unter anderem folgende:

- `0`, `0.5`, `1`, `1.5`, 2 usw. in Schritten von 0,5 bis maximal 256. Der ACU-Mindestwert von 0 ist nur für die Aurora-Versionen verfügbar, die die Auto-Pause-Funktion unterstützen.

Die maximal verfügbare Kapazität ist sowohl von der DB-Engine-Version als auch von der Plattformversion abhängig.

In diesem Beispiel legen Sie den ACU-Bereich eines Aurora-DB-Clusters namens `sample-cluster` auf ein Minimum von `48.5` und maximal 64 fest.

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
  --serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

Die Kapazitätsänderung erfolgt sofort, unabhängig davon, ob Sie sie sofort oder während des nächsten geplanten Wartungsfensters anwenden möchten.

Danach können Sie dem Cluster Aurora Serverless v2-DB-Instances hinzufügen und jede neue DB-Instance kann zwischen 48,5 und 64 ACUs skaliert werden. Der neue Kapazitätsbereich gilt auch für alle Aurora Serverless v2-DB-Instances, die sich bereits im Cluster befanden. Die DB-Instances skalieren bei Bedarf hoch oder herunter, um in den neuen Kapazitätsbereich zu gelangen.

Weitere Beispiele für die Einstellung des Kapazitätsbereichs über die CLI finden Sie unter Auswählen des Aurora Serverless v2-Kapazitätsbereichs für einen Aurora-Cluster.

Um die Skalierungskonfiguration eines Aurora Serverless-DB-Clusters über die AWS CLI zu ändern, führen Sie den AWS CLI-Befehl modify-db-cluster aus. Geben Sie die Option `--serverless-v2-scaling-configuration` an, um die minimale und die maximale Kapazität zu konfigurieren. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: `0`, `0.5`, `1`, `1.5`, 2 usw. in Schritten von 0,5 ACUs bis maximal 256.
- Aurora PostgreSQL: `0`, `0.5`, `1`, `1.5`, 2 usw. in Schritten von 0,5 ACUs bis maximal 256.
- Der ACU-Mindestwert von 0 ist nur für die Aurora-Versionen verfügbar, die die Auto-Pause-Funktion unterstützen.

Im folgenden Beispiel ändern Sie die Skalierungskonfiguration einer Aurora Serverless v2-DB-Instance mit dem Namen `sample-instance`, die Teil eines Clusters namens `sample-cluster` ist.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

Für Windows:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

RDS-API

Sie können die Kapazität einer Aurora-DB-Instance über die API-Operation ModifyDBCluster festlegen. Geben Sie den Parameter `ServerlessV2ScalingConfiguration` an. Gültige Werte für die Felder `MinCapacity` und `MaxCapacity` sind unter anderem folgende:

- `0`, `0.5`, `1`, `1.5`, 2 usw. in Schritten von 0,5 bis maximal 256. Der ACU-Mindestwert von 0 ist nur für die Aurora-Versionen verfügbar, die die Auto-Pause-Funktion unterstützen.

Sie können die Skalierungskonfiguration eines Clusters, der Aurora Serverless v2-DB-Instances enthält, über die API-Operation ModifyDBCluster ändern. Geben Sie den Parameter

`ServerlessV2ScalingConfiguration` an, um die minimale und die maximale Kapazität zu konfigurieren. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: `0`, `0.5`, `1`, `1.5`, 2 usw. in Schritten von 0,5 ACUs bis maximal 256.
- Aurora PostgreSQL: `0`, `0.5`, `1`, `1.5`, 2 usw. in Schritten von 0,5 ACUs bis maximal 256.
- Der ACU-Mindestwert von 0 ist nur für die Aurora-Versionen verfügbar, die die Auto-Pause-Funktion unterstützen.

Die maximal verfügbare Kapazität ist sowohl von der DB-Engine-Version als auch von der Plattformversion abhängig.

Die Kapazitätsänderung erfolgt sofort, unabhängig davon, ob Sie sie sofort oder während des nächsten geplanten Wartungsfensters anwenden möchten.

Aktualisieren des Aurora Serverless v2-DB-Clusters, um eine Skalierung auf 256 ACUs zu ermöglichen

In einigen Fällen erhalten Sie eine Fehlermeldung, wenn Sie versuchen, Ihren Aurora Serverless v2-DB-Cluster auf Kapazitäten von mehr als 128 ACUs zu skalieren. In der Fehlermeldung erfahren Sie, welche Instances mit dem neuen Skalierungsbereich nicht kompatibel sind. Dies unterstreicht die wichtige Beziehung zwischen Ihrem Kapazitätsbereich, der DB-Engine-Version und der Plattformversion.

Es gibt zwei mögliche Ursachen dafür, dass eine Skalierung auf mehr als 128 ACUs nicht möglich ist:

- ältere DB-Engine-Version: Aktualisieren Sie die DB-Engine-Version auf eine Version, die 256 ACUs unterstützt. Weitere Informationen finden Sie unter Kapazität von Aurora Serverless v2.
- ältere Plattformversion: Aktualisieren Sie die Plattform für Ihren Aurora Serverless v2-DB-Cluster. Sie können dafür eine der folgenden Möglichkeiten auswählen:
  - Beenden Sie den DB-Cluster und starten Sie ihn neu. Wenn der Cluster neu gestartet wird, wird er mit der neuesten Plattformversion ausgeführt, mit der möglicherweise ein höheres ACU-Maximum erreicht werden kann.

    Das Beenden und Starten eines DB-Clusters verursacht jedoch eine gewisse Ausfallzeit, in der Regel mehrere Minuten. Weitere Informationen finden Sie unter Stoppen und Starten eines Amazon Aurora-DB-Clusters.
  - Verwenden Sie eine Blau/Grün-Bereitstellung. Weitere Informationen finden Sie unter Überblick über Aurora Blue/Green Aurora-Bereitstellungen.

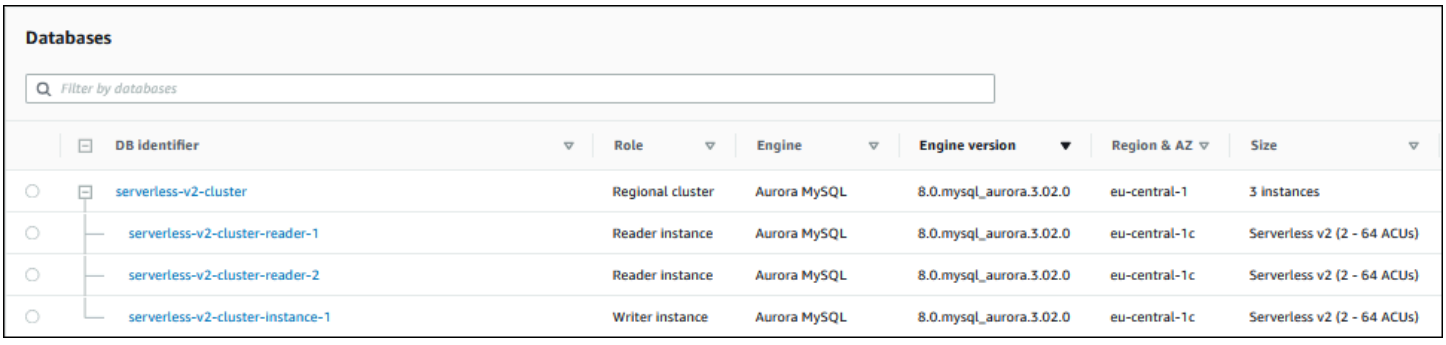
1. Erstellen Sie eine Blau/Grün-Bereitstellung. Weitere Informationen finden Sie unter Eine blue/green Bereitstellung in — Amazon Aurora erstellen.
2. Vergewissern Sie sich, dass Sie die maximale Kapazität für die Staging-Umgebung (grün) auf 256 ACUs festlegen können.
3. Wechseln Sie zur grünen Umgebung. Weitere Informationen finden Sie unter Wechseln einer blue/green Bereitstellung in Amazon Aurora.
4. Löschen Sie den ursprünglichen DB-Cluster.

> **Note**
>
> Wenn Sie Ihre Datenbankanmeldedaten in AWS Secrets Manager verwalten, können Sie keine Blau/Grün-Bereitstellungen verwenden.
> Aurora Global Database unterstützt keine Blau/Grün-Bereitstellungen.

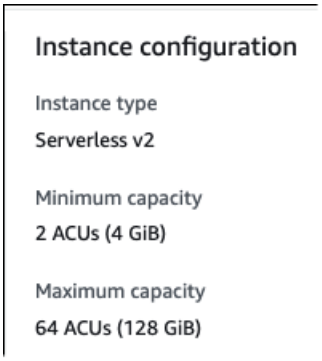
Überprüfen des Kapazitätsbereichs für Aurora Serverless v2

Das Verfahren zur Überprüfung des Kapazitätsbereichs für Ihren Aurora Serverless v2-Cluster erfordert, dass Sie zuerst einen Kapazitätsbereich festlegen. Wenn Sie dies noch nicht getan haben, gehen Sie wie unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster beschrieben vor.

Der jeweilige Kapazitätsbereich, den Sie auf Cluster-Ebene festlegen, gilt für alle Aurora Serverless v2-DB-Instances in Ihrem Cluster. Die folgende Abbildung zeigt einen Cluster mit mehreren Aurora Serverless v2-DB-Instances. Jede Instance hat einen identischen Kapazitätsbereich.



Sie können auch die Detailseite für alle Aurora Serverless v2-DB-Instances im Cluster anzeigen. Der Kapazitätsbereich der DB-Instances wird auf dem Tab Configuration (Konfiguration) angezeigt.



Sie können den aktuellen Kapazitätsbereich für den Cluster auch auf der Seite Modify (Ändern) für den Cluster sehen. An diesem Punkt können Sie den Kapazitätsbereich ändern. Alle Möglichkeiten, wie Sie den Kapazitätsbereich einstellen oder ändern können, finden Sie unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster.

Überprüfen des aktuellen Kapazitätsbereichs für einen Aurora-Cluster

Sie können den Kapazitätsbereich überprüfen, der für Aurora Serverless v2-DB-Instances in einem Cluster konfiguriert ist, indem Sie sich das `ServerlessV2ScalingConfiguration`-Attribut für den Cluster ansehen. Das folgende AWS CLI-Beispiel zeigt einen Cluster mit einer minimalen Kapazität von 0,5 Aurora-Kapazitätseinheiten (ACU) und einer maximalen Kapazität von 16 ACUs.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
  --query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
    [
        {
            "MinCapacity": 0.5,
            "MaxCapacity": 16.0
        }
    ]
]
```

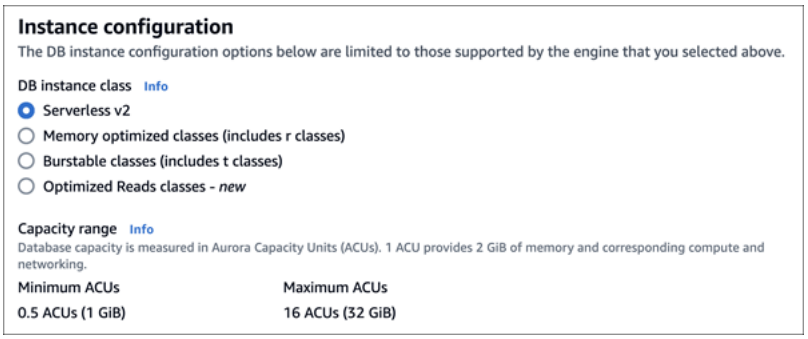
Hinzufügen eines Aurora Serverless v2-Readers

Wenn Sie Ihrem Cluster eine Aurora Serverless v2-Reader-DB-Instance hinzufügen möchten, folgen Sie dem gleichen allgemeinen Verfahren wie in Hinzufügen von Aurora-Replicas zu einem DB-Cluster. Wählen Sie die Instance-Klasse Serverless v2 für die neue DB-Instance aus.

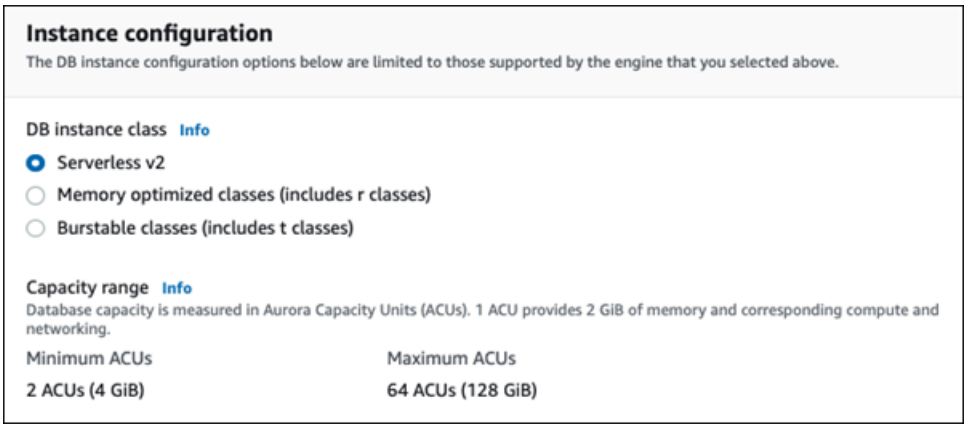
Wenn die Reader-DB-Instance die erste Aurora Serverless v2-DB-Instance im Cluster ist, wählen Sie auch den Kapazitätsbereich aus. Diese Einstellung gilt für diese Reader-DB-Instance und für

alle weiteren Aurora Serverless v2-DB-Instances, die Sie dem Cluster hinzufügen. Jede Aurora Serverless v2-DB-Instance kann zwischen dem minimalen und dem maximalen ACU-Wert skaliert werden.

Wenn im Cluster bereits andere Aurora Serverless v2-Instances vorhanden sind, zeigt das Dialogfeld Reader hinzufügen den aktuellen Kapazitätsbereich für den Cluster an. In diesem Fall können Sie die Kapazität nicht ändern. Die folgende Abbildung zeigt den Bericht über die Kapazität des aktuellen Clusters.



Wenn Sie dem Cluster bereits Aurora Serverless v2-DB-Instances hinzugefügt haben, wird Ihnen beim Hinzufügen weiterer Aurora Serverless v2-Reader-DB-Instances der aktuelle Kapazitätsbereich angezeigt. Die folgende Abbildung zeigt diese schreibgeschützten Steuerelemente.



Wenn Sie den Kapazitätsbereich für den Cluster ändern möchten, gehen Sie vor, wie in Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster bechrieben.

Bei Clustern, die mehr als eine Reader-DB-Instance enthalten, spielt die Failover-Priorität jeder Aurora Serverless v2-Reader-DB-Instance eine wichtige Rolle dabei, wie diese DB-Instance hoch- oder herunterskaliert. Sie können die Priorität nicht angeben, wenn Sie den Cluster erstellen. Denken Sie an diese Eigenschaft, wenn Sie Ihrem Cluster einen zweiten Reader oder eine weitere DB-Instance hinzufügen. Weitere Informationen finden Sie unter Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader.

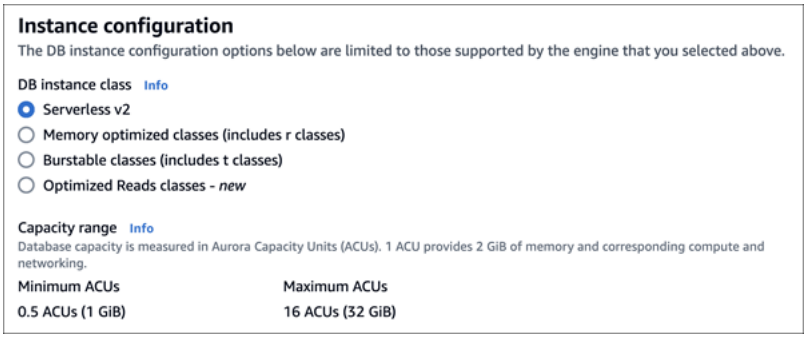
Weitere Möglichkeiten zum Anzeigen des aktuellen Kapazitätsbereichs für einen Cluster finden Sie unter Überprüfen des Kapazitätsbereichs für Aurora Serverless v2.

Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2

Sie können eine bereitgestellte DB-Instance konvertieren, um Aurora Serverless v2 zu verwenden. Befolgen Sie dazu die Anleitung unter Ändern einer DB-Instance in einem DB-Cluster. Der Cluster muss die Anforderungen in Anforderungen und Einschränkungen für Aurora Serverless v2 erfüllen. Aurora Serverless v2-DB-Instances erfordern beispielsweise, dass der Cluster bestimmte Mindest-Engine-Versionen ausführt.

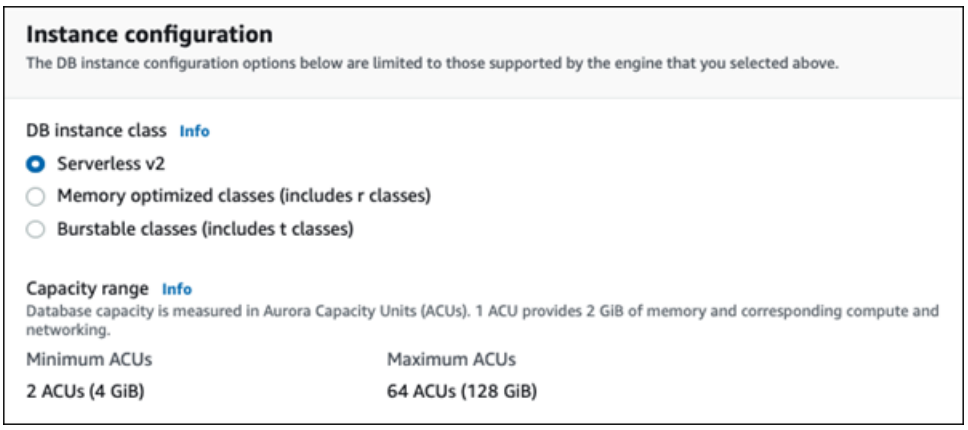
Angenommen, Sie konvertieren einen laufenden bereitgestellten Cluster, um Aurora Serverless v2 zu nutzen. In diesem Fall können Sie Ausfallzeiten minimieren, indem Sie im ersten Schritt des Umstellungsprozesses eine DB-Instance in Aurora Serverless v2 konvertieren. Das vollständige Verfahren finden Sie unter Umstellung von einem bereitgestellten Cluster zu Aurora Serverless v2.

Wenn die DB-Instance, die Sie konvertieren, die erste Aurora Serverless v2-DB-Instance im Cluster ist, wählen Sie den Kapazitätsbereich für den Cluster im Rahmen der Operation Modify (Ändern) aus. Dieser Kapazitätsbereich gilt für jede Aurora Serverless v2-DB-Instance, die Sie dem Cluster hinzufügen. Die folgende Abbildung zeigt die Seite, auf der Sie die minimalen und maximalen Aurora-Kapazitätseinheiten (ACUs) angeben.



Weitere Informationen zur Bedeutung des Kapazitätsbereichs finden Sie unter Kapazität von Aurora Serverless v2.

Wenn der Cluster bereits eine oder mehrere Aurora Serverless v2-DB-Instances enthält, sehen Sie den vorhandenen Kapazitätsbereich, wenn Sie die Operation Modify (Ändern) verwenden. Die folgende Abbildung zeigt ein Beispiel für dieses Informationsfeld.



Wenn Sie den Kapazitätsbereich für den Cluster ändern möchten, nachdem Sie weitere Aurora Serverless v2-DB-Instances hinzugefügt haben, gehen Sie vor, wie in Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster bechrieben.

Konvertieren eines Aurora Serverless v2-Writers oder -Readers in bereitgestellt

Sie können eine Aurora Serverless v2-DB-Instance in eine bereitgestellte DB-Instance konvertieren. Befolgen Sie dazu die Anleitung unter Ändern einer DB-Instance in einem DB-Cluster. Wählen Sie eine andere DB-Instance-Klasse als Serverless aus.

Sie können eine Aurora Serverless v2-DB-Instance in eine bereitgestellte Instance konvertieren, wenn sie eine größere Kapazität benötigt, als mit den maximalen Aurora-Kapazitätseinheiten (ACUs) einer Aurora Serverless v2-DB-Instance verfügbar ist. Zum Beispiel haben die größten DB-Instance-Klassen db.r5 und db.r6g eine größere Speicherkapazität als die Kapazität, auf die eine Aurora Serverless v2-DB-Instance skalieren kann.

> **Tip**
>
> Einige der älteren DB-Instance-Klassen wie db.r3 und db.t2 sind für die Aurora-Versionen, die Sie mit Aurora Serverless v2 verwenden, nicht verfügbar. Informationen dazu, wie Sie feststellen können, welche DB-Instance-Klassen beim Konvertieren einer Aurora Serverless v2-DB-Instance in eine bereitgestellte Instance verwendet werden können, finden Sie unter Unterstützte DB-Engines für DB-Instance-Klassen.

Wenn Sie die Writer-DB-Instance Ihres Clusters von Aurora Serverless v2 in eine bereitgestellte Instance konvertieren, können Sie die Vorgehensweise unter Umstellung von einem bereitgestellten

Cluster zu Aurora Serverless v2 verwenden. Stellen Sie eine der Reader-DB-Instances im Cluster von Aurora Serverless v2 auf eine bereitgestellte Instance um. Führen Sie ein Failover durch, damit diese bereitgestellte DB-Instance zur Writer-Instance wird.

Jeder Kapazitätsbereich, den Sie zuvor für den Cluster angegeben haben, bleibt bestehen, auch wenn alle Aurora Serverless v2-DB-Instances aus dem Cluster entfernt werden. Wenn Sie den Kapazitätsbereich ändern möchten, können Sie den Cluster ändern, wie in Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster erläutert.

Anhalten von Aurora Serverless v2-DB-Instances

Sie können Aurora-Cluster so konfigurieren, dass ihre Aurora Serverless v2-DB-Instances automatisch angehalten und fortgesetzt werden. Weitere Informationen finden Sie unter Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2.

Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader

Bei Clustern mit mehreren Aurora Serverless v2-DB-Instances oder mit einer Mischung aus bereitgestellten und Aurora Serverless v2-DB-Instances achten Sie auf die Einstellung der Hochstufungsstufe für jede einzelne Aurora Serverless v2-DB-Instance. Diese Einstellung steuert mehr Verhaltensweisen für Aurora Serverless v2-DB-Instances als für bereitgestellte DB-Instances.

In der AWS-Managementkonsole geben Sie diese Einstellung mit der Option Failover priority (Failover-Priorität) unter Additional configuration (Zusätzliche Konfiguration) auf den Seiten Create database (Datenbank erstellen), Modify Instance (Instance ändern) und Add reader (Reader hinzufügen) an. Sie sehen diese Eigenschaft für vorhandene DB-Instances in der optionalen Spalte Priority tier(Prioritätsstufe) auf der Seite Databases (Datenbanken). Diese Eigenschaft können Sie auch der Detailseite für einen DB-Cluster oder eine DB-Instance entnehmen.

Bei bereitgestellten DB-Instances bestimmt die Auswahl der Stufe 0 bis 15 nur die Reihenfolge, in der Aurora entscheidet, welche Reader-DB-Instance während eines Failover-Vorgangs zum Writer hochgestuft werden soll. Für Aurora Serverless v2-Reader-DB-Instances bestimmt die Stufennummer auch, ob die DB-Instance auf die Kapazität der Writer-DB-Instance hochskaliert oder unabhängig und basierend auf ihrer eigenen Workload skaliert wird. Aurora Serverless v2-Reader-DB-Instances in Stufe 0 oder 1 werden auf einer minimalen Kapazität gehalten, die mindestens so hoch ist wie die der Writer-DB-Instance. Auf diese Weise sind sie im Falle eines Failovers zur Übernahme von der Writer-DB-Instance bereit. Wenn die Writer-DB-Instance eine bereitgestellte DB-Instance ist, schätzt Aurora die entsprechende Aurora Serverless v2-Kapazität. Sie verwendet diese Schätzung als Mindestkapazität für die Aurora Serverless v2-Reader-DB-Instance.

Aurora Serverless v2-Reader-DB-Instances der Stufen 2 bis 15 haben nicht die gleiche Einschränkung ihrer minimalen Kapazität. Wenn sie inaktiv sind, können sie auf den minimalen Aurora-Kapazitätseinheitwert (ACU) herunterskaliert werden, der im Kapazitätsbereich des Clusters angegeben ist.

Das folgende AWS CLI-Beispiel für Linux zeigt, wie Sie die Hochstufungsstufen aller DB-Instances in Ihrem Cluster untersuchen. Das letzte Feld enthält den Wert `True` für die Writer-DB-Instance und `False` für alle Reader-DB-Instances.

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
  --query 'DBClusters[*].DBClusterMembers[*].
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
  --output text

1   instance-192  True
1   tier-01-4840  False
10  tier-10-7425  False
15  tier-15-6694  False
```

Das folgende AWS CLI-Beispiel für Linux zeigt, wie Sie die Hochstufungsstufe einer bestimmten DB-Instance in Ihrem Cluster ändern. Mit den Befehlen wird zuerst die DB-Instance geändert und eine neue Hochstufungsstufe festgelegt. Dann wird gewartet, bis die DB-Instance wieder verfügbar ist, und die neue Hochstufungsstufe für die DB-Instance bestätigt.

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
  --query '*[].[PromotionTier]' --output text
0
```

Weitere Hilfestellung zum Angeben von Hochstufungsstufen für verschiedene Anwendungsfälle finden Sie unter Aurora Serverless v2-Skalierung.

Verwenden von TLS/SSL mit Aurora Serverless v2

Aurora Serverless v2 kann das TLS-/SSL-Protokoll (Transport Layer Security/Secure Sockets Layer) verwenden, um die Kommunikation zwischen Clients und den Aurora Serverless v2-DB-Instances zu verschlüsseln. Die TLS-/SSL-Versionen 1.0, 1.1 und 1.2 werden unterstützt. Allgemeine Informationen zur Verwendung von TLS/SSL mit Aurora finden Sie unter TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

Weitere Informationen zum Verbinden mit der Aurora MySQL-Datenbank über den MySQL-Client finden Sie unter Verbinden mit einer DB-Instance, auf der die MySQL-Datenbank-Engine ausgeführt wird.

Aurora Serverless v2 unterstützt alle für den MySQL-Client (`mysql`) und PostgreSQL-Client (`psql`) verfügbaren TLS-/SSL-Modi, einschließlich der in der folgenden Tabelle aufgeführten.

Beschreibung des TLS-/SSL-Modus	mysql-	psql
Verbinden ohne Verwendung von TLS/SSL	DISABLED	disable
Verbindung zuerst mit TLS/SSL probieren, bei Bedarf aber auf Nicht-SSL zurückgreifen	PREFERRED	bevorzugen (Standard)
Verwendung von TLS/SSL erzwingen	REQUIRED	require
SSL erzwingen und die Zertifizierungsstelle (CA) überprüfen.	VERIFY_CA	verify-ca
TLS/SSL erzwingen, CA überprüfen und CA-Hostnamen überprüfen	VERIFY_IDENTITY	verify-full

Aurora Serverless v2 nutzt Platzhalter-Zertifikate. Wenn Sie bei Verwendung von TLS/SSL die Option "CA überprüfen" oder "CA und CA-Hostnamen überprüfen" angeben, laden Sie zuerst den Amazon Root CA 1 Trust Store von Amazon Trust Services herunter. Danach können Sie diese PEM-formatierte Datei in Ihrem Client-Befehl identifizieren. Gehen Sie folgendermaßen vor, um diesen Vorgang mit dem PostgreSQL-Client auszuführen.

Für Linux, macOS oder Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Weitere Informationen zum Arbeiten mit der Aurora PostgreSQL-Datenbank unter Verwendung des Postgres-Clients finden Sie unter Herstellen einer Verbindung zu einer DB-Instance, in der die PostgreSQL-Datenbank-Engine ausgeführt wird.

Weitere Informationen zum Verbinden mit Aurora-DB-Clustern im Allgemeinen finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

Unterstützte Verschlüsselungssammlungen für Verbindungen mit Aurora Serverless v2-DB-Clustern

Durch die Verwendung von konfigurierbaren Chiffrier-Suiten können Sie mehr Kontrolle über die Sicherheit Ihrer Datenbankverbindungen haben. Sie können eine Liste von Verschlüsselungssammlungen angeben, die Sie zum Sichern von TLS/SSL-Verbindungen zu Ihrer Datenbank des Clients zulassen möchten. Mit konfigurierbaren Chiffrier-Suiten können Sie die Verbindungsverschlüsselung steuern, die Ihr Datenbankserver akzeptiert. Dadurch wird die Verwendung von Verschlüsselungsverfahren verhindert, die nicht sicher sind oder nicht mehr verwendet werden.

Aurora Serverless v2-DB-Cluster, die auf Aurora MySQL basieren, unterstützen dieselben Verschlüsselungssammlungen wie von Aurora MySQL bereitgestellte DB-Cluster. Weitere Informationen über diese Verschlüsselungssammlungen finden Sie unter Konfigurieren von Cipher-Suites für Verbindungen mit Aurora-MySQL-DB-Clustern.

Aurora Serverless v2-DB-Cluster, die auf Aurora MySQL basieren, unterstützen dieselben Verschlüsselungssammlungen wie von Aurora PostgreSQL bereitgestellte DB-Cluster. Weitere Informationen über diese Verschlüsselungssammlungen finden Sie unter Konfigurieren von Chiffrier-Suiten für Verbindungen mit Aurora-PostgreSQL-DB-Clustern.

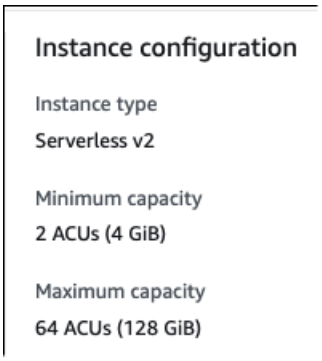
Anzeigen von Aurora Serverless v2-Writern und -Readern

Sie können sich die Details von Aurora Serverless v2-DB-Instances auf die gleiche Weise anzeigen lassen wie für bereitgestellte DB-Instances. Befolgen Sie hierzu das allgemeine Verfahren unter Anzeigen eines DB-Clusters von Amazon Aurora. Ein Cluster könnte alle Aurora Serverless v2-DB-Instances, alle bereitgestellten DB-Instances oder jeweils einige davon enthalten.

Nachdem Sie eine oder mehrere Aurora Serverless v2-DB-Instances erstellt haben, können Sie anzeigen, welche DB-Instances den Typ Serverless (Serverlos) und welche den Typ Instance haben. Außerdem können Sie die minimalen und maximalen Aurora-Kapazitätseinheiten (ACUs) anzeigen, die die Aurora Serverless v2-DB-Instance verwenden kann. Jede ACU ist eine Kombination aus Rechenkapazität (CPU) und Arbeitsspeicherkapazität (RAM). Dieser Kapazitätsbereich gilt für jede Aurora Serverless v2-DB-Instance im Cluster. Informationen zum Verfahren zur Überprüfung des Kapazitätsbereichs eines Clusters oder einer Aurora Serverless v2-DB-Instance im Cluster finden Sie unter Überprüfen des Kapazitätsbereichs für Aurora Serverless v2.

In der AWS-Managementkonsole sind Aurora Serverless v2-DB-Instances in der Spalte Size (Größe) der Seite Database (Datenbanken) gekennzeichnet. Bei bereitgestellten DB-Instances wird der Name einer DB-Instance-Klasse wie r6g.xlarge angezeigt. Die Aurora Serverless-DB-Instances zeigen Serverless (Serverlos) für die DB-Instance-Klasse sowie die minimale und maximale Kapazität der DB-Instance an. Es wird beispielsweise Serverless v2 (4–64 ACUs) oder Serverless v2 (1–40 ACUs) angezeigt.

Die gleichen Informationen finden Sie auf dem Tab Configuration (Konfiguration) für jede Aurora Serverless v2-DB-Instance in der Konsole. Sie sehen beispielsweise einen Abschnitt Instance type (Instance-Typ) wie den folgenden. Hier ist der Wert Instance type (Instance-Typ) Serverless v2, der Wert Minimum capacity (Minimale Kapazität) beträgt 2 ACUs (4 GiB) und der Wert Maximum capacity (Maximale Kapazität) ist auf 64 ACUs (128 GiB) festgelegt.



Sie können die Kapazität jeder einzelnen Aurora Serverless v2-DB-Instance im Zeitverlauf überwachen. Auf diese Weise können Sie die minimalen, maximalen und durchschnittlichen ACUs überprüfen, die von jeder DB-Instance verbraucht werden. Sie können auch überprüfen, wie nahe die DB-Instance an ihre minimale oder maximale Kapazität gekommen ist. Wenn Sie solche Details in der AWS-Managementkonsole anzeigen möchten, untersuchen Sie die Grafiken der Amazon-CloudWatch-Metriken auf dem Tab Monitoring (Überwachung) für die DB-Instance. Weitere Informationen über die angezeigten Metriken und ihre Interpretation finden Sie unter Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2.

Protokollierung für Aurora Serverless v2

Wenn Sie die Datenbankprotokollierung aktivieren möchten, geben Sie die Protokolle an, die mithilfe von Konfigurationsparametern in Ihrer benutzerdefinierten Parametergruppe aktiviert werden sollen.

Für Aurora MySQL können Sie die folgenden Protokolle aktivieren.

Aurora MySQL	Beschreibung
`general_log`	Erstellt das allgemeine Protokoll. Stellen Sie zum Einschalten auf 1. Die Standardeinstellung ist aus (0).
`log_queries_not_using_indexes`	Protokolliert alle Abfragen im Slow-Query-Protokoll, das keinen Index verwendet. Die Standardeinstellung ist aus (0). Stellen Sie auf 1 ein, um dieses Protokoll zu aktivieren.
`long_query_time`	Verhindert, dass Fast-Running-Queries im langsamen Slow-Query-Protokoll protokoll iert werden. Kann auf einen Gleitkommawert zwischen 0 und 31536000 gesetzt werden. Die Standardeinstellung ist 0 (nicht aktiv).
`server_audit_events`	Die Liste der Ereignisse, die in den Protokollen erfasst werden sollen. Unterstützte Werte sind `CONNECT`, `QUERY`, `QUERY_DCL` , `QUERY_DDL` , `QUERY_DML` und `TABLE`.
`server_audit_logging`	Setzen Sie den Parameter auf 1, um die Serverprüfungsprotokollierung zu aktivieren. Wenn Sie diesen Parameter aktivieren, können Sie angeben, welche Prüfungsereignisse an CloudWatch gesendet werden sollen, indem Sie sie im `server_audit_events` -Parameter auflisten.

Aurora MySQL	Beschreibung
`slow_query_log`	Erstellt ein Slow-Query-Protokoll. Auf 1 setzen, um das Slow-Query-Protokoll zu aktivieren. Die Standardeinstellung ist aus (0).

Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.

Für Aurora PostgreSQL können Sie die folgenden Protokolle auf Ihren Aurora Serverless v2-DB-Instances aktivieren.

Aurora PostgreSQL	Beschreibung
`log_connections`	Protokolliert jede erfolgreiche Verbindung.
`log_disconnections`	Protokolliert das Ende einer Sitzung einschließlich der Dauer.
`log_lock_waits`	Der Standardwert ist 0 (aus). Stellen Sie auf 1 ein, um die Wartezeiten für die Sperrung zu protokollieren.
`log_min_duration_statement`	Die Mindestdauer (in Millisekunden), die eine Anweisung vor der Protokollierung ausgeführt wird.
`log_min_messages`	Legt die Nachrichtenebenen fest, die protokolliert werden. Unterstützte Werte sind , , , , , , , , , und `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`. Zum Protokollieren von Leistungsdaten im `postgres`-Protokoll, setzen Sie den Wert auf `debug1` .

Aurora PostgreSQL	Beschreibung
`log_temp_files`	Protokolliert die Verwendung von temporären Dateien, die über den angegebenen Kilobyte (kB) liegen.
`log_statement`	Steuert die spezifischen SQL-Anweisungen, die protokolliert werden. Unterstützte Werte sind `none`, `ddl`, `mod` und `all`. Der Standardwert ist `none`.

Themen

- Protokollieren mit Amazon CloudWatch
- Anzeigen von Aurora Serverless v2-Protokollen in Amazon CloudWatch
- Überwachen der Kapazität mit Amazon CloudWatch
- Überwachen von Aurora Serverless v2-Aktivitäten zum Anhalten und Fortsetzen

Protokollieren mit Amazon CloudWatch

Nachdem Sie das Verfahren unter Protokollierung für Aurora Serverless v2 verwendet haben, um auszuwählen, welche Datenbankprotokolle aktiviert werden sollen, können Sie festlegen, welche Protokolle Sie auf Amazon CloudWatch hochladen („veröffentlichen“) möchten.

Mit Amazon CloudWatch können Sie Protokolldaten analysieren, Alarme erstellen und Metriken anzeigen lassen. Standardmäßig werden Fehlerprotokolle für Aurora Serverless v2 aktiviert und automatisch auf CloudWatch hochgeladen. Sie können auch andere Protokolle von Aurora Serverless v2-DB-Instances auf CloudWatch hochladen.

In diesem Fall wählen Sie aus, welches dieser Protokolle auf CloudWatch hochgeladen werden soll, indem Sie die Einstellungen Log exports (Protokollexporte) in der AWS-Managementkonsole oder die Option `--enable-cloudwatch-logs-exports` in der AWS CLI verwenden.

Sie können auswählen, welches Ihrer Aurora Serverless v2-Protokolle auf CloudWatch hochgeladen werden soll. Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.

Wie bei jedem Typ von Aurora-DB-Cluster können Sie die standardmäßige DB-Cluster-Parametergruppe nicht ändern. Erstellen Sie stattdessen Ihre eigene DB-Cluster-Parametergruppe basierend auf einem Standardparameter für Ihren DB-Cluster und Engine-Typ. Sie sollten Ihre benutzerdefinierte DB-Cluster-Parametergruppe erstellen, bevor Sie Ihren Aurora Serverless v2-DB-Cluster erstellen, damit Sie diese auswählen können, wann Sie eine Datenbank in der Konsole erstellen.

> **Note**
>
> Bei Aurora Serverless v2 können Sie sowohl DB-Cluster- als auch DB-Parametergruppen erstellen. Im Gegensatz dazu können Sie bei Aurora Serverless v1 nur DB-Cluster-Parametergruppen erstellen.

Anzeigen von Aurora Serverless v2-Protokollen in Amazon CloudWatch

Nachdem Sie das Verfahren unter Protokollieren mit Amazon CloudWatch verwendet haben, um auszuwählen, welche Datenbankprotokolle aktiviert werden sollen, können Sie den Inhalt der Protokolle anzeigen.

Weitere Informationen zur Verwendung von CloudWatch mit Aurora-MySQL- und Aurora-PostgreSQL-Protokollen finden Sie unter Überwachung von Protokollereignissen in Amazon CloudWatch und Veröffentlichen von Aurora-PostgreSQL-Protokollen in Amazon CloudWatch Logs.

So zeigen Sie Protokolle für Ihren Aurora Serverless v2-DB-Cluster an:

1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie Ihre AWS-Region.
3. Wählen Sie Protokollgruppen.
4. Wählen Sie Ihr DB-Cluster-Protokoll für Aurora Serverless v2 in der Liste aus. Bei Protokollen ist das Benennungsmuster wie folgt.

   ```
   /aws/rds/cluster/cluster-name/log_type
   ```

> **Note**
>
> Für Aurora-MySQL-kompatible DB-Cluster von Aurora Serverless v2 enthält das Fehlerprotokoll auch dann Skalierungsereignisse für Pufferpools, wenn keine Fehler vorliegen.

Überwachen der Kapazität mit Amazon CloudWatch

Mit Aurora Serverless v2 können Sie CloudWatch verwenden, um die Kapazität und Auslastung aller Aurora Serverless v2-DB-Instances in Ihrem Cluster zu überwachen. Sie können Metriken auf Instance-Ebene anzeigen, um die Kapazität jeder einzelnen Aurora Serverless v2-DB-Instance beim Hoch- und Herunterskalieren zu überprüfen. Sie können die kapazitätsbezogenen Metriken auch mit anderen Metriken vergleichen, um zu sehen, wie sich Änderungen an Workloads auf den Ressourcenverbrauch auswirken. Sie können beispielsweise `ServerlessDatabaseCapacity` mit `DatabaseUsedMemory`, `DatabaseConnections` und `DMLThroughput` vergleichen, um einzuschätzen, wie Ihr DB-Cluster während des Betriebs reagiert. Weitere Informationen zu den kapazitätsbezogenen Metriken, die für Aurora Serverless v2 gelten, finden Sie unter Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2.

So überwachen Sie die Kapazität Ihres Aurora Serverless v2-DB-Clusters:

1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie Metrics (Metriken) aus. Alle verfügbaren Metriken werden in der Konsole als Karten angezeigt, gruppiert nach Servicename.
3. Wählen Sie RDS.
4. (Optional) Verwenden Sie das Feld Suchen (Search), um die Metriken zu finden, die für Aurora Serverless v2 besonders wichtig sind: `ServerlessDatabaseCapacity`, `ACUUtilization`, `CPUUtilization` und `FreeableMemory`.

Wir empfehlen Ihnen, ein CloudWatch-Dashboard einzurichten, um die Kapazität Ihres Aurora Serverless v2-DB-Clusters mit dieser kapazitätsbezogenen Metrik zu überwachen. Informationen dazu finden Sie unter Erstellen von Dashboards mit CloudWatch.

Weitere Informationen zur Verwendung von Amazon CloudWatch mit Amazon Aurora finden Sie unter Veröffentlichen von Amazon Aurora MySQL-Protokollen in Amazon CloudWatch Logs.

Überwachen von Aurora Serverless v2-Aktivitäten zum Anhalten und Fortsetzen

Aurora schreibt eine separate Protokolldatei für Aurora Serverless v2-DB-Instances mit aktivierter Auto-Pause-Funktion. Aurora schreibt für jedes 10-Minuten-Intervall, in dem die Instance nicht angehalten wurde, in das Protokoll. Aurora speichert bis zu sieben dieser Protokolle, die täglich rotiert werden. Die aktuelle Protokolldatei hat den Namen `instance.log` und ältere Protokolle werden nach dem Muster `instance.`*`YYYY-MM-DD`*`.`*`N`*`.log` benannt.

Dieses Protokoll ist für Aurora Serverless v2-DB-Instances standardmäßig aktiviert, bei denen die Auto-Pause-Funktion aktiviert ist. Sie können den Inhalt dieses Protokolls in der AWS-Managementkonsole oder mithilfe der AWS CLI oder RDSP-API anzeigen. Derzeit ist es nicht möglich, dieses Protokoll in CloudWatch hochzuladen.

Die unter DB-Instance-Ereignisse aufgeführten Ereignisse bieten einen allgemeinen Überblick über die Aktivitäten zum Anhalten und Fortsetzen, u. a. folgende:

- wann mit dem Anhalten der Instance begonnen wird und wann das Anhalten beendet wird.
- wann mit dem Fortsetzen der Instance begonnen wird und wann das Fortsetzen beendet wird.
- Fälle, in denen die Instance das Anhalten versucht hat, dies aber verhindert wurde.

`instance.log` bietet detailliertere Informationen zu den Gründen, warum eine Aurora Serverless v2-Instance angehalten oder nicht angehalten werden kann.

Das Protokoll könnte darauf hinweisen, dass eine Instance aus verschiedenen Gründen fortgesetzt wurde:

- Benutzeraktivität: Eine Datenbankverbindungsanfrage. Diese kann von einer interaktiven Clientsitzung, einem Aufruf der RDS-Daten-API oder einer Anfrage zum Herunterladen von Protokollen von der Instance stammen.
- Hintergrundaktivität: Eine weit gefasste Kategorie, die alle Gründe umfasst, aus denen Aurora eine Instance fortsetzt. Wenn beispielsweise eine Verbindungsanfrage zu einer Reader-Instance dazu führt, dass die Writer-Instance fortgesetzt wird, zeigt das Protokoll für den Reader die Benutzeraktivität, während das Protokoll für den Writer diese Fortsetzungsanforderung als Hintergrundaktivität klassifiziert. Andere Gründe außer Benutzerverbindungsanforderungen, aus denen Aurora eine Instance fortsetzen kann, finden Sie unter Fortsetzen einer automatisch angehaltenen Aurora Serverless v2-Instance.

Wenn Aurora keine Bedingungen feststellt, die verhindern würden, dass die Instance nach Ablauf des Auto-Pause-Intervalls angehalten wird, wird regelmäßig eine Informationsmeldung in das Protokoll geschrieben. Bei Clustern mit nur einer DB-Instance enthält das Protokoll die folgende Meldung:

```
[INFO] No auto-pause blockers registered since time
```

Bei Clustern mit mehreren DB-Instances unterscheidet sich die Meldung geringfügig. Das liegt daran, dass ein Writer aufgrund von Aktivitäten auf einer der Reader-Instances möglicherweise nicht angehalten werden kann. Wenn die Aktivität auf dem Reader beendet wird, bevor das Auto-Pause-Intervall auf dem Writer abläuft, kann der Writer zur erwarteten Zeit angehalten werden.

```
[INFO] No auto-pause blockers registered since time.
Database might be required to maintain compute capacity for high availability.
```

Wenn eine Operation zum Anhalten gestartet wird, aber eine neue Datenbankverbindungsanforderung eingeht, bevor die Instance das Anhalten beendet hat, enthält das Protokoll die folgende Meldung:

```
[INFO] Unable to pause database due to a new database activity
```

Wenn Aurora Bedingungen feststellt, die definitiv verhindern, dass die Instance angehalten werden kann, enthält das Protokoll die folgende Meldung mit einer Auflistung aller dieser Bedingungen:

```
[INFO] Auto-pause blockers registered since time: list_of_conditions
```

Auf diese Weise verhindert Aurora nicht, dass Sie Replikation, Null-ETL-Integration, Aurora Global Database usw. in Kombination mit der Auto-Pause-Funktion aktivieren. Das Protokoll informiert Sie darüber, wenn die Verwendung solcher Funktionen möglicherweise verhindert, dass die Auto-Pause-Funktion wirksam wird.

Aus den folgenden Gründen kann eine Aurora Serverless v2-Instance das Timeout-Intervall für das automatische Anhalten überschreiten, aber daran gehindert werden, anzuhalten:

- Datenbankaktivität vor dem Auto-Pause-Timeout: Die DB-Instance hat eine Verbindungsanforderung erhalten, bevor das Timeout-Intervall abgelaufen ist.
- Mitglied der globalen Datenbank: Wenn der DB-Cluster Teil einer globalen Aurora-Datenbank ist, werden die Aurora Serverless v2-Instances im Cluster nicht angehalten. Ein Cluster kann von einem eigenständigen Cluster zu einem globalen Datenbank-Cluster geändert werden.

Daher werden Instances, die zuvor automatisch angehalten wurden, möglicherweise nicht mehr angehalten und dieser Grund wird im Protokoll gemeldet. Sobald ein Cluster Mitglied einer globalen Datenbank wird, wird er erst wieder zu einem eigenständigen Cluster, wenn Sie ihn explizit trennen. Der primäre Cluster wird immer noch als Teil der globalen Datenbank betrachtet, auch wenn Sie alle sekundären Cluster trennen.

- Replikationsfähigkeit konfiguriert: Für die Writer-DB-Instance ist die Engine-spezifische Replikation aktiviert, entweder Binlog-Replikation für MySQL oder logische Replikation für PostgreSQL. Dieser Zustand könnte auch durch die Verwendung einer anderen Aurora-Funktion verursacht werden, für die die Replikation aktiviert werden muss, z. B. Null-ETL-Integrationen oder Datenbankaktivitäts-Streams (Database Activity Streams, DAS).
- Verzögerung bei kontinuierlichen Backups: Wenn das Aurora-Speichersystem das Anwenden der Speicheränderungen bis zum aktuellen Zeitpunkt nicht abgeschlossen hat, wird die Writer-Instance erst angehalten, wenn sie auf dem aktuellen Stand ist. Dieser Zustand betrifft nur die Writer-Instance und wird voraussichtlich relativ kurz sein.
- Service- oder Kundenwartungsaktion: Wenn ein Wartungsvorgang gestartet wird, wird die DB-Instance erst wieder angehalten, wenn dieser Vorgang abgeschlossen ist. Dieser Zustand umfasst eine Vielzahl von Vorgängen, die von Ihnen oder Aurora gestartet werden können, z. B. Upgrades, Klonen, Ändern von Konfigurationseinstellungen, Upgrades, Herunterladen von Protokolldateien usw. Dieses Ereignis tritt auch auf, wenn Sie das Löschen einer Instance anfordern und Aurora die Instance im Rahmen des Löschmechanismus kurzzeitig fortsetzt.
- Vorübergehendes Kommunikationsproblem: Wenn Aurora nicht feststellen kann, ob die Skalierungskonfiguration derzeit eine Mindestkapazitätseinstellung von 0 ACUs hat, wird die Instance nicht angehalten. Es ist zu erwarten, dass dies ein sehr seltenes Ereignis ist.

Performance und Skalierung für Aurora Serverless v2

Die folgenden Verfahren und Beispiele zeigen, wie Sie den Kapazitätsbereich für Aurora Serverless v2-Cluster und ihre zugehörigen DB-Instances festlegen können. Sie können die folgenden Verfahren auch verwenden, um zu überwachen, wie ausgelastet Ihre DB-Instances sind. Anschließend können Sie anhand Ihrer Ergebnisse feststellen, ob Sie den Kapazitätsbereich nach oben oder unten anpassen müssen.

Stellen Sie sicher, dass Sie mit der Funktionsweise der Skalierung in Aurora Serverless v2 vertraut sind. Der Skalierungsmechanismus ist anders als in Aurora Serverless v1. Details hierzu finden Sie unter Aurora Serverless v2-Skalierung.

Inhalt

- Auswählen des Aurora Serverless v2-Kapazitätsbereichs für einen Aurora-Cluster
  - Auswählen der minimalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster
  - Auswählen der maximalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster
  - Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-MySQL-Clusters
  - Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-PostgreSQL-Clusters
- Arbeiten mit Parametergruppen für Aurora Serverless v2
  - Standard-Parameterwerte
  - Maximale Anzahl der Verbindungen für Aurora Serverless v2
  - Parameter, die Aurora anpasst, während Aurora Serverless v2 hoch- und herunterskaliert
  - Parameter, die Aurora basierend auf der maximalen Kapazität von Aurora Serverless v2 berechnet
- Vermeiden von Fehlern aufgrund von zu wenig Speicher
- Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2
  - Wie Aurora Serverless v2-Metriken in Ihre AWS-Rechnung einfließen
  - Beispiele für CloudWatch-Befehle für Aurora Serverless v2-Metriken
- Überwachung der Aurora Serverless v2-Leistung mit Performance Insights
- Behebung von Kapazitätsproblemen bei Aurora Serverless v2

Auswählen des Aurora Serverless v2-Kapazitätsbereichs für einen Aurora-Cluster

Bei Aurora Serverless v2-DB-Instances legen Sie den Kapazitätsbereich, der für alle DB-Instances in Ihrem DB-Cluster gilt, beim Hinzufügen der ersten Aurora Serverless v2-DB-Instance zum DB-Cluster fest. Weitere Informationen zum entsprechenden Verfahren finden Sie unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster.

Sie können den Kapazitätsbereich für einen vorhandenen Cluster auch ändern. In den folgenden Abschnitten wird ausführlicher beschrieben, wie Sie geeignete Mindest- und Höchstwerte auswählen und was passiert, wenn Sie den Kapazitätsbereich ändern. Durch Ändern des Kapazitätsbereichs können sich beispielsweise die Standardwerte einiger Konfigurationsparameter ändern. Das

Anwenden aller Parameteränderungen kann einen Neustart jeder Aurora Serverless v2-DB-Instance erfordern.

Themen

- Auswählen der minimalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster
- Auswählen der maximalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster
- Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-MySQL-Clusters
- Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-PostgreSQL-Clusters

Auswählen der minimalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster

Es ist verlockend, immer 0,5 für die miminale Aurora Serverless v2-Kapazitätseinstellung auszuwählen. Dieser Wert ermöglicht es der DB-Instance, auf die kleinste Kapazität herunterzuskalieren, wenn sie vollständig im Leerlauf ist, und dabei aktiv zu bleiben. Sie können das Verhalten für das automatische Anhalten auch aktivieren, indem Sie eine Mindestkapazität von 0 ACUs angeben, wie unter Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2 erläutert. Je nachdem, wie Sie diesen Cluster verwenden und wie die anderen Einstellungen konfiguriert sind, ist jedoch eine andere Mindestkapazität ggf. am effektivsten. Berücksichtigen Sie bei der Auswahl der Mindestkapazitätseinstellung die folgenden Faktoren:

- Die Skalierungsrate für eine Aurora Serverless v2-DB-Instance hängt von ihrer aktuellen Kapazität ab. Je höher die aktuelle Kapazität, desto schneller kann sie hochskalieren. Wenn die DB-Instance schnell auf eine sehr hohe Kapazität hochskalieren muss, sollten Sie die Mindestkapazität auf einen Wert einstellen, bei dem die Skalierungsrate Ihren Anforderungen entspricht.
- Wenn Sie die DB-Instance-Klasse Ihrer DB-Instances in Erwartung einer besonders hohen oder niedrigen Workload in der Regel ändern, können Sie diese Erfahrung verwenden, um den entsprechenden Aurora Serverless v2-Kapazitätsbereich grob einzuschätzen. Informationen zum Ermitteln der Speichergröße, die in Zeiten mit geringem Datenverkehr verwendet werden soll, finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

  Angenommen, Sie verwenden die DB-Instance-Klasse db.r6g.xlarge, wenn Ihr Cluster eine geringe Workload hat. Diese DB-Instance-Klasse verfügt über 32 GiB Speicher. Somit können Sie eine minimale Einstellung der Aurora-Kapazitätseinheit (ACU) von 16 angeben, um eine Aurora Serverless v2-DB-Instance einzurichten, die auf ungefähr dieselbe Kapazität herunterskalieren kann. Das liegt daran, dass jede ACU ungefähr 2 GiB Speicher entspricht. Sie können einen

etwas niedrigeren Wert angeben, damit die DB-Instance weiter herunterskaliert wird, falls Ihre db.r6g.xlarge-DB-Instance manchmal nicht ausgelastet war.

- Wenn Ihre Anwendung dann am effizientesten arbeitet, wenn die DB-Instances eine bestimmte Datenmenge im Puffer-Cache haben, sollten Sie eine minimale ACU-Einstellung angeben, bei der der Speicher groß genug ist, um die häufig aufgerufenen Daten zu speichern. Andernfalls werden einige Daten im Puffer-Cache bereinigt, wenn Aurora Serverless v2-DB-Instances auf eine niedrigere Speichergröße herunterskalieren. Wenn die DB-Instances dann wieder hochskaliert werden, werden die Informationen im Zeitverlauf wieder in den Puffer-Cache eingelesen. Wenn für das Einlesen der Daten wieder in den Puffercache eine erhebliche I/O-Menge erforderlich ist, ist es möglicherweise effektiver, einen höheren minimalen ACU-Wert auszuwählen.
- Wenn Ihre Aurora Serverless v2-DB-Instances die meiste Zeit mit einer bestimmten Kapazität ausgeführt werden, erwägen Sie, eine Mindestkapazitätseinstellung anzugeben, die zwar niedriger als diese Baseline, aber nicht zu niedrig ist. Aurora Serverless v2 DB-Instances können am effektivsten abschätzen, wie viel und wie schnell hochskaliert werden muss, wenn die aktuelle Kapazität nicht deutlich niedriger als die erforderliche Kapazität ist.
- Wenn Ihre bereitgestellte Workload Speicheranforderungen hat, die für kleine DB-Instance-Klassen wie T3 oder T4g zu hoch sind, wählen Sie eine minimale ACU-Einstellung, die ähnlichen Speicher bietet wie eine R5- oder R6g-DB-Instance.

  Insbesondere empfehlen wir die folgende Mindestkapazität für die Verwendung mit den angegebenen Funktionen (diese Empfehlungen können sich ändern):

  - Performance Insights – 2 ACUs
  - Globale Aurora-Datenbanken – 8 ACUs (gilt nur für die primäre AWS-Region)
- In Aurora erfolgt die Replikation auf Speicherebene, sodass sich die Reader-Kapazität nicht direkt auf die Replikation auswirkt. Stellen Sie bei Aurora Serverless v2-DB-Instances für Reader, die unabhängig skaliert werden, jedoch sicher, dass die Mindestkapazität ausreicht, um Workloads in schreibintensiven Zeiten zu bewältigen, um eine Abfragelatenz zu vermeiden. Wenn die Reader-DB-Instances in den Hochstufungsstufen 2 bis 15 Leistungsprobleme haben, sollten Sie die Mindestkapazität des Clusters ggf. erhöhen. Weitere Informationen zur Entscheidung, ob Reader-DB-Instances zusammen mit dem Writer oder unabhängig skaliert werden, finden Sie unter Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader.
- Wenn Sie über einen DB-Cluster mit Aurora Serverless v2-Reader-DB-Instances verfügen, werden die Reader nicht zusammen mit der Writer-DB-Instance skaliert, wenn die Hochstufungsstufe der Reader nicht 0 oder 1 ist. In diesem Fall kann das Festlegen einer geringen Mindestkapazität zu einer übermäßigen Replikationsverzögerung führen. Das liegt daran, dass die Reader

möglicherweise nicht genug Kapazität haben, um Änderungen vom Writer zu übernehmen, wenn die Datenbank ausgelastet ist. Wir empfehlen, die Mindestkapazität auf einen Wert festzulegen, der eine mit der Writer-DB-Instance vergleichbare Menge an Speicher und CPU darstellt.

- Der Wert des `max_connections`-Parameters für Aurora Serverless v2-DB-Instances basiert auf der Speichergröße, die von den maximalen ACUs abgeleitet wird. Wenn Sie jedoch eine Mindestkapazität von 0 oder 0,5 ACUs auf PostgreSQL-kompatiblen DB-Instances angeben, ist der Höchstwert von `max_connections` auf 2 000 begrenzt.

  Wenn Sie beabsichtigen, den Aurora-PostgreSQL-Cluster für einen Workload mit hohem Verbindungsdurchsatz zu verwenden, sollten Sie eine Mindest-ACU-Einstellung von 1 oder höher verwenden. Details darüber, wie Aurora Serverless v2 den `max_connections`-Konfigurationsparameter behandelt, finden Sie unter Maximale Anzahl der Verbindungen für Aurora Serverless v2.
- Die Zeit, die eine Aurora Serverless v2-DB-Instance benötigt, um von ihrer minimalen Kapazität auf ihre maximale Kapazität zu skalieren, hängt von der Differenz zwischen ihren minimalen und maximalen ACU-Werten ab. Wenn die aktuelle Kapazität der DB-Instance groß ist, skaliert Aurora Serverless v2 in größeren Schritten, als wenn die DB-Instance mit einer geringen Kapazität beginnt. Wenn Sie also eine relativ große maximale Kapazität angeben und die DB-Instance die meiste Zeit in diesem Kapazitätsbereich bleibt, sollten Sie erwägen, die minimale ACU-Einstellung zu erhöhen. Auf diese Weise kann eine inaktive DB-Instance schneller wieder auf maximale Kapazität skaliert werden.

Auswählen der maximalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster

Es ist verlockend, immer einen hohen Wert für die maximale Aurora Serverless v2-Kapazitätseinstellung auszuwählen. Eine große maximale Kapazität ermöglicht es der DB-Instance, bei intensiver Workload am stärksten hochzuskalieren. Mit einem niedrigen Wert entfällt die Möglichkeit unerwarteter Gebühren. Je nachdem, wie Sie diesen Cluster verwenden und die anderen Einstellungen konfigurieren, kann der effektivste Wert höher oder niedriger sein, als Sie ursprünglich dachten. Berücksichtigen Sie bei der Auswahl der maximalen Kapazitätseinstellung die folgenden Faktoren:

- Die maximale Kapazität muss mindestens so hoch sein wie die Mindestkapazität. Sie können die minimale und maximale Kapazität auf den gleichen Wert festlegen. In diesem Fall skaliert sich die Kapazität jedoch niemals hoch oder herunter. Daher ist die Verwendung identischer Werte für die minimale und maximale Kapazität außerhalb von Testsituationen nicht geeignet.

- Die maximale Kapazität muss höher als 0,5 ACU sein. Sie können die minimale und maximale Kapazität in den meisten Fällen auf den gleichen Wert festlegen. Sie können 0,5 jedoch nicht sowohl für das Minimum als auch für das Maximum angeben. Verwenden Sie einen Wert von 1 oder höher für die maximale Kapazität.
- Wenn Sie die DB-Instance-Klasse Ihrer DB-Instances in Erwartung einer besonders hohen oder niedrigen Workload in der Regel ändern, können Sie diese Erfahrung verwenden, um den entsprechenden Aurora Serverless v2-Kapazitätsbereich einzuschätzen. Informationen zum Ermitteln der Speichergröße, die in Zeiten mit hohem Datenverkehr verwendet werden soll, finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.

  Angenommen, Sie verwenden die DB-Instance-Klasse db.r6g.4xlarge, wenn Ihr Cluster eine hohe Workload hat. Diese DB-Instance-Klasse verfügt über 128 GiB Speicher. Somit können Sie eine maximale ACU-Einstellung von 64 angeben, um eine Aurora Serverless v2-DB-Instance einzurichten, die auf ungefähr dieselbe Kapazität hochskalieren kann. Das liegt daran, dass jede ACU ungefähr 2 GiB Speicher entspricht. Sie können einen etwas höheren Wert angeben, damit die DB-Instance weiter hochskalieren kann, falls Ihre db.r6g.4xlarge-DB-Instance manchmal nicht genug Kapazität hat, um die Workload effektiv zu bewältigen.
- Wenn Sie eine Haushaltsobergrenze für Ihre Datenbankauslastung haben, wählen Sie einen Wert, der innerhalb dieser Obergrenze bleibt, auch wenn alle Ihre Aurora Serverless v2-DB-Instances ständig mit maximaler Kapazität laufen. Denken Sie daran, dass wenn Sie n Aurora Serverless v2-DB-Instances in Ihrem Cluster haben, die theoretische maximale Aurora Serverless v2-Kapazität, die der Cluster jederzeit verbrauchen kann, n-mal die maximale ACU-Einstellung für den Cluster ist. (Der tatsächlich verbrauchte Betrag ist möglicherweise geringer, wenn beispielsweise einige Reader unabhängig vom Writer skalieren.)
- Wenn Sie Aurora Serverless v2-Reader-DB-Instances dazu verwenden, den schreibgeschützten Workload der Writer-DB-Instance teilweise auszulagern, können Sie möglicherweise eine niedrigere Maximalkapazitätseinstellung auswählen. Damit berücksichtigen Sie die Tatsache, dass jede Reader-DB-Instance nicht gleichermaßen hoch skaliert werden muss wie im Falle, dass der Cluster nur eine einzelne DB-Instance enthält.
- Angenommen, Sie möchten sich vor übermäßiger Auslastung aufgrund falsch konfigurierter Datenbankparameter oder ineffizienter Abfragen in Ihrer Anwendung schützen. In diesem Fall können Sie eine versehentliche Überbeanspruchung vermeiden, indem Sie eine maximale Kapazitätseinstellung wählen, die niedriger ist als die absolut höchste, die Sie festlegen können.
- Wenn Spitzen aufgrund realer Benutzeraktivitäten zwar selten sind, aber dennoch auftreten, können Sie diese Situationen bei der Auswahl der maximalen Kapazitätseinstellung berücksichtigen. Wenn die Priorität darin besteht, dass die Anwendung weiterhin mit voller Leistung

und Skalierbarkeit ausgeführt wird, können Sie eine maximale Kapazitätseinstellung angeben, die höher ist, als Sie bei normaler Auslastung beobachten. Wenn es akzeptabel ist, dass die Anwendung bei sehr extremen Aktivitätsspitzen mit reduziertem Durchsatz ausgeführt wird, können Sie eine etwas niedrigere maximale Kapazitätseinstellung wählen. Stellen Sie sicher, dass Sie eine Einstellung auswählen, die immer noch über genügend Speicher- und CPU-Ressourcen verfügt, damit die Anwendung weiterhin ausgeführt wird.

- Wenn Sie Einstellungen in Ihrem Cluster aktivieren, die die Speicherauslastung für jede DB-Instance erhöhen, berücksichtigen Sie diesen Speicher bei der Entscheidung über den maximalen ACU-Wert. Zu diesen Einstellungen gehören diejenigen für Performance Insights, parallele Aurora MySQL-Abfragen, Aurora MySQL-Leistungsschema und Aurora MySQL-Binärprotokollreplikation. Stellen Sie sicher, dass der maximale ACU-Wert Aurora Serverless v2-DB-Instances erlaubt, ausreichend hoch zu skalieren, um die Workload zu bewältigen, wenn diese Funktion verwendet wird. Informationen zur Fehlerbehebung von Problemen, die durch die Kombination einer niedrigen maximalen ACU-Einstellung und Aurora-Funktionen verursacht werden, die Speicher-Overhead verursachen, finden Sie unter Vermeiden von Fehlern aufgrund von zu wenig Speicher.

Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-MySQL-Clusters

Das folgende AWS CLI-Beispiel zeigt, wie Sie den ACU-Bereich für Aurora Serverless v2-DB-Instances in einem bestehenden Aurora-MySQL-Cluster aktualisieren. Anfänglich beträgt der Kapazitätsbereich für den Cluster 8 bis 32 ACUs.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

Die DB-Instance ist inaktiv und wird auf 8 ACUs herunterskaliert. Die folgenden kapazitätsbezogenen Einstellungen gelten zu diesem Zeitpunkt für die DB-Instance. Zum Darstellen der Größe des Pufferpools in leicht lesbaren Einheiten teilen wir ihn durch 2 hoch 30, was zu einer Messung in Gibibyte (GiB) führt. Das liegt daran, dass speicherbezogene Messungen für Aurora Einheiten verwenden, die auf Zweierpotenzen und nicht auf Zehnerpotenzen basieren.

```
mysql> select @@max_connections;
+-------------------+
```

```
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                9294577664 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|   8.65625 |
+-----------+
1 row in set (0.00 sec)
```

Als Nächstes ändern wir den Kapazitätsbereich für den Cluster. Nachdem der `modify-db-cluster`-Befehl abgeschlossen ist, beträgt der ACU-Bereich für den Cluster 12,5 bis 80.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Durch Ändern des Kapazitätsbereichs können sich die Standardwerte einiger Konfigurationsparameter ändern. Aurora kann einige dieser neuen Standardwerte sofort anwenden. Einige der Parameteränderungen werden jedoch erst nach einem Neustart wirksam. Der Status `pending-reboot` gibt an, dass ein Neustart erforderlich ist, um einige Parameteränderungen anzuwenden.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

Zu diesem Zeitpunkt ist der Cluster inaktiv und die DB-Instance `serverless-v2-instance-1` verbraucht 12,5 ACUs. Der `innodb_buffer_pool_size`-Parameter ist bereits basierend auf der aktuellen Kapazität der DB-Instance angepasst. Der `max_connections`-Parameter spiegelt immer noch den Wert der früheren maximalen Kapazität wider. Das Zurücksetzen dieses Werts erfordert einen Neustart der DB-Instance.

> **Note**
>
> Wenn Sie den `max_connections`-Parameter direkt in einer benutzerdefinierten DB-Parametergruppe festlegen, ist kein Neustart erforderlich.

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               15572402176 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
```

```
+---------------+
| gibibytes     |
+---------------+
| 14.5029296875 |
+---------------+
1 row in set (0.00 sec)
```

Jetzt starten wir die DB-Instance neu und warten darauf, dass sie wieder verfügbar ist.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Der `pending-reboot`-Status ist gelöscht. Der Wert `in-sync` bestätigt, dass Aurora alle ausstehenden Parameteränderungen übernommen hat.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

Der `innodb_buffer_pool_size`-Parameter hat sich auf seine endgültige Größe für eine inaktive DB-Instance erhöht. Der `max_connections`-Parameter wurde erhöht, um einen vom maximalen ACU-Wert abgeleiteten Wert widerzuspiegeln. Die Formel, die Aurora für `max_connections` verwendet, führt zu einem Anstieg von 1 000, wenn sich die Speichergröße verdoppelt.

```
mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
```

```
|               16139681792 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|  15.03125 |
+-----------+
1 row in set (0.00 sec)

mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              4000 |
+-------------------+
1 row in set (0.00 sec)
```

Wir legen den Kapazitätsbereich auf 0,5 bis 128 ACUs fest und starten die DB-Instance neu. Jetzt hat die inaktive DB-Instance eine Puffer-Cache-Größe von weniger als 1 GiB. Daher messen wir sie nun in Mebibyte (MiB). Der `max_connections`-Wert 5 000 wird von der Speichergröße der maximalen Kapazitätseinstellung abgeleitet.

```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----------+-------------------+
| mebibytes | @@max_connections |
+-----------+-------------------+
|       672 |              5000 |
+-----------+-------------------+
1 row in set (0.00 sec)
```

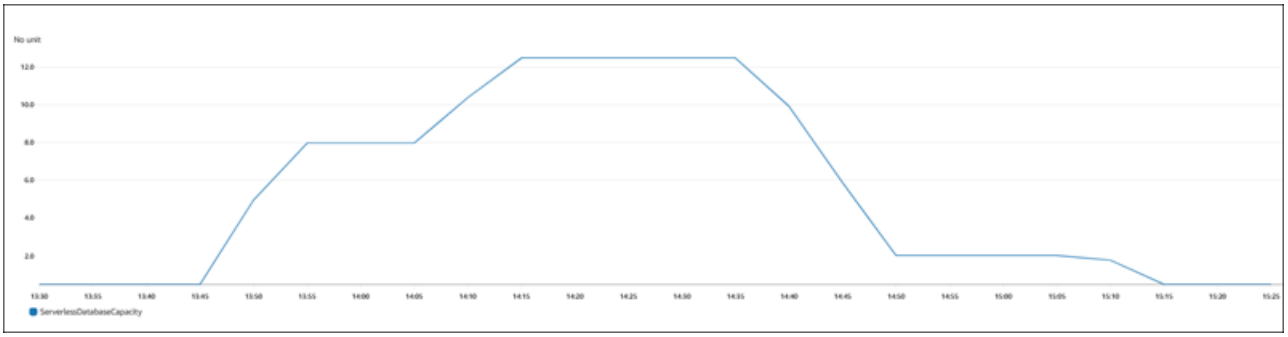
Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-PostgreSQL-Clusters

Das folgende CLI-Beispiel zeigt, wie Sie den ACU-Bereich für DB-Instances von Aurora Serverless v2 in einem bestehenden Aurora-PostgreSQL-Cluster aktualisieren.

1. Der Kapazitätsbereich für den Cluster beginnt bei 0,5 bis 1 ACU.
2. Ändern Sie den Kapazitätsbereich auf 8 bis 32 ACUs.

3. Ändern Sie den Kapazitätsbereich auf 12,5 bis 80 ACUs.
4. Ändern Sie den Kapazitätsbereich auf 0,5 bis 128 ACUs.
5. Legen Sie die Kapazität wieder auf den ursprünglichen Bereich von 0,5 bis 1 ACU fest.

In der folgenden Abbildung sind die Kapazitätsänderungen in Amazon CloudWatch dargestellt.



Die DB-Instance ist inaktiv und wird auf 0,5 ACUs herunterskaliert. Die folgenden kapazitätsbezogenen Einstellungen gelten zu diesem Zeitpunkt für die DB-Instance.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Als Nächstes ändern wir den Kapazitätsbereich für den Cluster. Nachdem der `modify-db-cluster`-Befehl abgeschlossen ist, beträgt der ACU-Bereich für den Cluster 8,0 bis 32.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

Durch Ändern des Kapazitätsbereichs können sich die Standardwerte einiger Konfigurationsparameter ändern. Aurora kann einige dieser neuen Standardwerte sofort anwenden. Einige der Parameteränderungen werden jedoch erst nach einem Neustart wirksam. Der Status `pending-reboot` gibt an, dass ein Neustart erforderlich ist, um einige Parameteränderungen anzuwenden.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

Zu diesem Zeitpunkt ist der Cluster inaktiv und die DB-Instance `serverless-v2-instance-1` verbraucht 8,0 ACUs. Der `shared_buffers`-Parameter ist bereits basierend auf der aktuellen Kapazität der DB-Instance angepasst. Der `max_connections`-Parameter spiegelt immer noch den Wert der früheren maximalen Kapazität wider. Das Zurücksetzen dieses Werts erfordert einen Neustart der DB-Instance.

> **Note**
>
> Wenn Sie den `max_connections`-Parameter direkt in einer benutzerdefinierten DB-Parametergruppe festlegen, ist kein Neustart erforderlich.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
```

```
 1425408
(1 row)
```

Wir starten die DB-Instance neu und warten darauf, dass sie wieder verfügbar ist.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Nachdem die DB-Instance neu gestartet wurde, ist der Status `pending-reboot` gelöscht. Der Wert `in-sync` bestätigt, dass Aurora alle ausstehenden Parameteränderungen übernommen hat.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

Nach dem Neustart zeigt `max_connections` den Wert der neuen maximalen Kapazität an.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
```

```
(1 row)
```

Als Nächstes ändern wir den Kapazitätsbereich für den Cluster auf 12,5 bis 80 ACUs.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Zu diesem Zeitpunkt ist der Cluster inaktiv und die DB-Instance `serverless-v2-instance-1` verbraucht 12,5 ACUs. Der `shared_buffers`-Parameter ist bereits basierend auf der aktuellen Kapazität der DB-Instance angepasst. Der `max_connections`-Wert beträgt immer noch 5 000.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

Wir führen einen weiteren Neustart durch, aber die Parameterwerte bleiben gleich. Dies liegt daran, dass für `max_connections` ein Höchstwert von 5 000 für einen DB-Cluster von Aurora Serverless v2 gilt, auf dem Aurora PostgreSQL ausgeführt wird.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
```

```
 shared_buffers
----------------
 2211840
(1 row)
```

Jetzt stellen wir den Kapazitätsbereich zwischen 0,5 und 128 ACUs ein. Der DB-Cluster wird zunächst auf 10 ACUs und dann auf 2 herunterskaliert. Wir starten die DB-Instance neu.

```
postgres=> show max_connections;
 max_connections
-----------------
 2000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Der `max_connections`-Wert für DB-Instances von Aurora Serverless v2 basiert auf der Speichergröße, die von den maximalen ACUs abgeleitet wird. Wenn Sie jedoch eine Mindestkapazität von 0 oder 0,5 ACUs auf PostgreSQL-kompatiblen DB-Instances angeben, ist der Höchstwert von `max_connections` auf 2 000 begrenzt.

Jetzt legen wir die Kapazität wieder auf den ursprünglichen Bereich von 0,5 bis 1 ACU fest und starten die DB-Instance neu. Der `max_connections`-Parameter ist auf seinen ursprünglichen Wert zurückgekehrt.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Arbeiten mit Parametergruppen für Aurora Serverless v2

Wenn Sie Ihren Aurora Serverless v2-DB-Cluster erstellen, wählen Sie eine bestimmte Aurora-DB-Engine und eine zugehörige DB-Cluster-Parametergruppe. Wenn Sie nicht damit vertraut sind, wie Aurora Parametergruppen verwendet, um Konfigurationseinstellungen konsistent auf Cluster anzuwenden, finden Sie weitere Informationen unter Parametergruppen für Amazon Aurora. Alle diese Verfahren zum Erstellen, Ändern, Anwenden sowie andere Aktionen für Parametergruppen gelten für Aurora Serverless v2.

Die Parametergruppen-Funktion funktioniert im Allgemeinen für bereitgestellte Cluster und Cluster, dieAurora Serverless v2-DB-Instances enthalten, gleich:

- Die Standardparameterwerte für alle DB-Instances im Cluster werden von der Cluster-Parametergruppe definiert.
- Sie können einige Parameter für bestimmte DB-Instances überschreiben, indem Sie eine benutzerdefinierte DB-Parametergruppe für diese DB-Instances angeben. Diesen Vorgang können Sie während des Debuggens oder der Leistungsoptimierung für bestimmte DB-Instances durchführen. Angenommen, Sie haben einen Cluster, der einige Aurora Serverless v2-DB-Instances und einige bereitgestellte DB-Instances enthält. In diesem Fall können Sie mithilfe einer benutzerdefinierten DB-Parametergruppe einige andere Parameter für die bereitgestellten DB-Instances angeben.
- Bei Aurora Serverless v2 können Sie alle Parameter verwenden, die den Wert `provisioned` im `SupportedEngineModes`-Attribut der Parametergruppe aufweisen. In Aurora Serverless v1 können Sie nur die Teilmenge von Parametern verwenden, die `serverless` im `SupportedEngineModes`-Attribut enthalten.

Themen

- Standard-Parameterwerte
- Maximale Anzahl der Verbindungen für Aurora Serverless v2
- Parameter, die Aurora anpasst, während Aurora Serverless v2 hoch- und herunterskaliert
- Parameter, die Aurora basierend auf der maximalen Kapazität von Aurora Serverless v2 berechnet

Standard-Parameterwerte

Der entscheidende Unterschied zwischen bereitgestellten DB-Instances und Aurora Serverless v2-DB-Instances besteht darin, dass Aurora alle benutzerdefinierten Parameterwerte für

bestimmte Parameter außer Kraft setzt, die sich auf die Kapazität der DB-Instance beziehen. Die benutzerdefinierten Parameterwerte gelten weiterhin für alle bereitgestellten DB-Instances in Ihrem Cluster. Weitere Details darüber, wie Aurora Serverless v2-DB-Instances die Parameter aus Aurora-Parametergruppen interpretieren, finden Sie unter Konfigurationsparameter für Aurora-Cluster. Die spezifischen Parameter, die Aurora Serverless v2 außer Kraft setzen, finden Sie unter Parameter, die Aurora anpasst, während Aurora Serverless v2 hoch- und herunterskaliert und Parameter, die Aurora basierend auf der maximalen Kapazität von Aurora Serverless v2 berechnet.

Sie können eine Liste der Standardwerte für die Standard-Parametergruppen für die verschiedenen Aurora-DB-Engines unter Verwendung des CLI-Befehls describe-db-cluster-parameters und der Abfrage der AWS-Region erhalten. Die folgenden Werte können Sie für die Optionen `--db-parameter-group-family` und `-db-parameter-group-name` für Engine-Versionen verwenden, die mit Aurora Serverless v2 kompatibel sind.

Datenbank-Engine und -Version	Parametergruppenfamilie	Name der Standard-Parametergruppe
Aurora-MySQL-Version 3	`aurora-mysql8.0`	`default.aurora-mysql8.0`
Aurora-PostgreSQL-Version 13.x	`aurora-postgresql13`	`default.aurora-postgresql13`
Aurora PostgreSQL Version 14.x	`aurora-postgresql14`	`default.aurora-postgresql14`
Aurora-PostgreSQL-Version 15.x	`aurora-postgresql15`	`default.aurora-postgresql15`
Aurora-PostgreSQL-Version 16.x	`aurora-postgresql16`	`default.aurora-postgresql16`
Aurora-PostgreSQL-Version 17.x	`aurora-postgresql17`	`default.aurora-postgresql17`

Im folgenden Beispiel wird eine Liste von Parametern aus der Standard-DB-Cluster-Gruppe für Aurora-MySQL-Version 3 und Aurora PostgreSQL 13 abgerufen. Dies sind die Aurora-MySQL- und Aurora-PostgreSQL-Versionen, die Sie mit Aurora Serverless v2 verwenden.

Linux, macOS oder Unix:

```
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-mysql8.0 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text

aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-postgresql13 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text
```

Für Windows:

```
aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-mysql8.0 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text

aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-postgresql13 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

Maximale Anzahl der Verbindungen für Aurora Serverless v2

Sowohl für Aurora MySQL als auch für Aurora PostgreSQL halten Aurora Serverless v2-DB-Instances den Parameter `max_connections` konstant, damit Verbindungen nicht unterbrochen werden, wenn die DB-Instance herunterskaliert wird. Der Standardwert für diesen Parameter leitet sich von einer Formel ab, die auf der Speichergröße der DB-Instance beruht. Weitere Informationen zur Formel und zu den Standardwerten für bereitgestellte DB-Instance-Klassen finden Sie unter Maximale Verbindungen zu einer Aurora MySQL-DB-Instance und Maximale Verbindungen zu einer Aurora PostgreSQL-DB-Instance.

Wenn Aurora Serverless v2 die Formel auswertet, verwendet es die Speichergröße basierend auf den maximalen Aurora-Kapazitätseinheiten (ACUs) für die DB-Instance, nicht den aktuellen ACU-Wert. Wenn Sie den Standardwert ändern, empfehlen wir, eine Variante der Formel zu verwenden, anstatt einen konstanten Wert anzugeben. Auf diese Weise kann Aurora Serverless v2 basierend auf der maximalen Kapazität eine angemessene Einstellung verwenden.

Wenn Sie die maximale Kapazität eines DB-Clusters von Aurora Serverless v2 ändern, müssen Sie die DB-Instances von Aurora Serverless v2 neu starten, um den `max_connections`-Wert zu aktualisieren. Dies liegt daran, dass es sich bei `max_connections` um einen statischen Parameter für Aurora Serverless v2 handelt.

Die folgende Tabelle zeigt die Standardwerte für `max_connections` für Aurora Serverless v2 basierend auf dem maximalen ACU-Wert.

Maximale ACUs	Standardmäßige maximale Verbindungen für Aurora MySQL	Standardmäßige maximale Verbindungen für Aurora PostgreSQL
1	90	189
4	135	823
8	1 000	1 669
16	2 000	3 360
32	3,000	5,000
64	4 000	5,000
128	5,000	5,000
192	6 000	5,000
256	6 000	5,000

> **Note**
>
> Der `max_connections`-Wert für DB-Instances von Aurora Serverless v2 basiert auf der Speichergröße, die von den maximalen ACUs abgeleitet wird. Wenn Sie jedoch eine Mindestkapazität von 0 oder 0,5 ACUs auf PostgreSQL-kompatiblen DB-Instances angeben, ist der Höchstwert von `max_connections` auf 2 000 begrenzt.

Konkrete Beispiele, die zeigen, wie sich `max_connections` mit dem maximalen ACU-Wert ändert, finden Sie unter Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-MySQL-Clusters und Beispiel: Ändern des Aurora Serverless v2-Kapazitätsbereichs eines Aurora-PostgreSQL-Clusters.

Parameter, die Aurora anpasst, während Aurora Serverless v2 hoch- und herunterskaliert

Bei der automatischen Skalierung muss Aurora Serverless v2 in der Lage sein, Parameter zu ändern, damit jede DB-Instance optimal für die erhöhte oder verringerte Kapazität funktioniert. Daher können Sie einige Parameter im Zusammenhang mit der Kapazität nicht außer Kraft setzen. Vermeiden Sie bei einigen Parametern, die Sie außer Kraft setzen können, eine Hartcodierung fester Werte. Die folgenden Überlegungen gelten für diese Einstellungen, die sich auf die Kapazität beziehen.

Bei Aurora MySQL ändert Aurora Serverless v2 die Größe einiger Parameter während der Skalierung dynamisch. Für die folgenden Parameter verwendet Aurora Serverless v2 keine benutzerdefinierten Parameterwerte, die Sie angeben:

- `innodb_buffer_pool_size`
- `innodb_purge_threads`
- `table_definition_cache`
- `table_open_cache`

Bei Aurora PostgreSQL ändert Aurora Serverless v2 die Größe der folgenden Parameter während der Skalierung dynamisch. Für die folgenden Parameter verwendet Aurora Serverless v2 keine benutzerdefinierten Parameterwerte, die Sie angeben:

- `shared_buffers`

Für alle anderen Parameter als die hier aufgelisteten funktionieren DB-Instances von Aurora Serverless v2 genauso wie bereitgestellte DB-Instances. Der Standardparameterwert wird von der Cluster-Parametergruppe geerbt. Sie können den Standardwert für den gesamten Cluster mit einer benutzerdefinierten Cluster-Parametergruppe ändern. Den Standardwert für bestimmte DB-Instances können Sie auch mithilfe einer benutzerdefinierten DB-Parametergruppe ändern. Die dynamischen Parameter werden sofort aktualisiert. Änderungen an statischen Parametern werden erst wirksam, nachdem Sie die DB-Instance neu gestartet haben.

Parameter, die Aurora basierend auf der maximalen Kapazität von Aurora Serverless v2 berechnet

Für die folgenden Parameter verwendet Aurora PostgreSQL auch Standardwerte, die genau wie bei `max_connections` von der Speichergröße basierend auf der maximalen ACU-Einstellung abgeleitet werden:

- `autovacuum_max_workers`
- `autovacuum_vacuum_cost_limit`
- `autovacuum_work_mem`
- `effective_cache_size`
- `maintenance_work_mem`

Vermeiden von Fehlern aufgrund von zu wenig Speicher

Wenn eine Ihrer Aurora Serverless v2-DB-Instances konsequent die Grenze ihrer maximalen Kapazität erreicht, weist Aurora auf diese Bedingung hin, indem die DB-Instance auf den Status `incompatible-parameters` festgelegt wird. Während die DB-Instance den Status `incompatible-parameters` aufweist, sind einige Operationen blockiert. Beispielsweise können Sie die Engine-Version nicht aktualisieren.

In der Regel wechselt Ihre DB-Instance in diesen Status, wenn sie aufgrund von Fehlern wegen zu wenig Speicher häufig neu gestartet wird. Aurora zeichnet ein Ereignis auf, wenn diese Art von Neustart stattfindet. Sie können das Ereignis anzeigen, indem Sie die Vorgehensweise unter Anzeigen von Amazon-RDS-Ereignissen befolgen. Eine ungewöhnlich hohe Speicherauslastung kann aufgrund von Overhead durch Aktivieren von Einstellungen wie Performance Insights und IAM-Authentifizierung auftreten. Sie kann auch durch eine hohe Workload Ihrer DB-Instance oder durch die Verwaltung der Metadaten entstehen, die mit einer großen Anzahl von Schemaobjekten verknüpft sind.

Wenn die Speicherauslastung sinkt, sodass die DB-Instance ihre maximale Kapazität nicht sehr oft erreicht, ändert Aurora den Status der DB-Instance automatisch wieder in `available`.

Zur Wiederherstellung nach diesem Zustand können Sie einige oder alle der folgenden Aktionen ausführen:

- Erhöhen Sie die untere Kapazitätsgrenze für Aurora Serverless v2-DB-Instances durch Ändern des Mindestwerts der Aurora-Kapazitätseinheit (ACU) für den Cluster. Dadurch werden Probleme vermieden, bei denen eine inaktive Datenbank auf eine Kapazität mit weniger Speicher herunterskaliert wird, als für die in Ihrem Cluster aktivierten Funktionen benötigt wird. Nachdem Sie die ACU-Einstellungen für den Cluster geändert haben, starten Sie die Aurora Serverless v2-DB-Instance neu. Dadurch wird geprüft, ob Aurora den Status wieder auf `available` zurücksetzen kann.
- Erhöhen Sie die untere Kapazitätsgrenze für Aurora Serverless v2-DB-Instances durch Ändern des maximalen Werts der Aurora-Kapazitätseinheit (ACU) für den Cluster. Auf diese Weise werden Probleme vermieden, bei denen eine ausgelastete Datenbank nicht auf eine Kapazität mit genügend Speicher für die in Ihrem Cluster aktivierten Funktionen und die Datenbank-Workload hochskaliert werden kann. Nachdem Sie die ACU-Einstellungen für den Cluster geändert haben, starten Sie die Aurora Serverless v2-DB-Instance neu. Dadurch wird geprüft, ob Aurora den Status wieder auf `available` zurücksetzen kann.
- Deaktivieren Sie Konfigurationseinstellungen, die Speicher-Overhead erfordern. Angenommen, Sie haben Funktionen wie AWS Identity and Access Management (IAM), Performance Insights oder Aurora MySQL-Binärprotokollreplikation aktiviert, verwenden sie jedoch nicht. In diesem Fall können Sie sie deaktivieren. Oder Sie können die minimalen und maximalen Kapazitätswerte für den Cluster nach oben korrigieren, um den von diesen Funktionen verwendeten Speicher zu berücksichtigen. Richtlinien zur Auswahl der minimalen und maximalen Kapazitätseinstellungen finden Sie unter Auswählen des Aurora Serverless v2-Kapazitätsbereichs für einen Aurora-Cluster.
- Reduzieren Sie die Workload der DB-Instance. Beispielsweise können Sie dem Cluster Reader-DB-Instances hinzufügen, um die Last von schreibgeschützten Abfragen auf weitere DB-Instances zu verteilen.
- Optimieren Sie den von Ihrer Anwendung verwendeten SQL-Code, um weniger Ressourcen zu verwenden. Sie können beispielsweise Ihre Abfragepläne untersuchen, das langsame Abfrageprotokoll überprüfen oder die Indizes in Ihren Tabellen anpassen. Außerdem können Sie andere traditionelle Arten von SQL-Optimierung durchführen.

Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2

Erste Schritte mit Amazon CloudWatch für Ihre Aurora Serverless v2-DB-Instance finden Sie unter Anzeigen von Aurora Serverless v2-Protokollen in Amazon CloudWatch. Weitere Informationen zur Überwachung von Aurora-DB-Clustern über CloudWatch finden Sie unter Überwachung von Protokollereignissen in Amazon CloudWatch.

Sie können Ihre Aurora Serverless v2-DB-Instances in CloudWatch anzeigen, um die von jeder DB-Instance verbrauchte Kapazität mit der `ServerlessDatabaseCapacity`-Metrik zu überwachen. Zusätzlich können Sie alle Aurora-CloudWatch-Standardmetriken überwachen, wie `DatabaseConnections` und `Queries`. Die vollständige Liste der CloudWatch-Metriken, die Sie für Aurora überwachen können, finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora. Die Metriken sind in Metriken auf Cluster-Ebene für Amazon Aurora und Metriken auf Instance-Ebene für Amazon Aurora in Metriken auf Cluster-Ebene und auf Instance-Ebene unterteilt.

Die folgenden CloudWatch-Metriken auf Instance-Ebene sind wichtig zu überwachen, damit Sie verstehen, wie Ihre Aurora Serverless v2-DB-Instances hoch- und herunterskalieren. Alle diese Metriken werden jede Sekunde berechnet. Auf diese Weise können Sie den aktuellen Status Ihrer Aurora Serverless v2-DB-Instances überwachen. Sie können Alarme einstellen, um sich gegebenenfalls benachrichtigen zu lassen, wenn sich eine Aurora Serverless v2-DB-Instance einem Schwellenwert für kapazitätsbezogene Metriken nähert. Sie können feststellen, ob die minimalen und maximalen Kapazitätseinstellungen angemessen sind oder ob Sie sie anpassen müssen. Sie können bestimmen, worauf Sie sich konzentrieren müssen, um die Effizienz Ihrer Datenbank zu optimieren.

- `ServerlessDatabaseCapacity`: Als Metrik auf Instance-Ebene gibt sie die Anzahl der ACUs an, die durch die aktuelle DB-Instance-Kapazität repräsentiert werden. Als Metrik auf Cluster-Ebene repräsentiert sie den Durchschnitt der `ServerlessDatabaseCapacity`-Werte aller Aurora Serverless v2-DB-Instances im Cluster. Diese Metrik ist nur eine Metrik auf Cluster-Ebene in Aurora Serverless v1. In Aurora Serverless v2 ist sie auf DB-Instance-Ebene und auf Cluster-Ebene verfügbar.
- `ACUUtilization`: Diese Metrik ist neu in Aurora Serverless v2. Dieser Wert wird als Prozentsatz dargestellt. Er wird als Wert der `ServerlessDatabaseCapacity`-Metrik geteilt durch den maximalen ACU-Wert des DB-Clusters berechnet. Beachten Sie die folgenden Richtlinien, um diese Metrik zu interpretieren und Maßnahmen zu ergreifen:
  - Wenn sich diese Metrik dem Wert `100.0` nähert, ist die DB-Instance so hoch wie möglich hochskaliert. Erwägen Sie, die maximale ACU-Einstellung für den Cluster zu erhöhen. Auf diese

Weise können sowohl Writer- als auch Reader-DB-Instances auf eine höhere Kapazität skaliert werden.

- Angenommen, eine schreibgeschützte Workload bewirkt, dass sich eine Reader-DB-Instance einem `ACUUtilization`-Wert von `100.0` nähert, während die Writer-DB-Instance ihrer maximalen Kapazität nicht annähernd erreicht. Erwägen Sie in diesem Fall, dem Cluster zusätzliche Reader-DB-Instances hinzuzufügen. Auf diese Weise können Sie den schreibgeschützten Teil der Workload auf mehrere DB-Instances verteilen und so die Last jeder Reader-DB-Instance reduzieren.
- Angenommen, Sie führen eine Produktionsanwendung aus, bei der Leistung und Skalierbarkeit die Hauptüberlegungen sind. In diesem Fall können Sie den maximalen ACU-Wert für den Cluster auf eine hohe Zahl festlegen. Ihr Ziel besteht darin, die `ACUUtilization`-Metrik immer unter `100.0` zu halten. Mit einem hohen maximalen ACU-Wert können Sie sicher sein, dass genügend Spielraum vorhanden ist, falls unerwartete Spitzen bei der Datenbankaktivität auftreten. Berechnet wird Ihnen nur die tatsächlich verbrauchte Datenbankkapazität.

- `CPUUtilization`: Diese Metrik wird in Aurora Serverless v2 anders interpretiert als in bereitgestellten DB-Instances. Bei Aurora Serverless v2 ist dieser Wert ein Prozentsatz, der als der aktuell CPU-Verbrauch dividiert durch die CPU-Kapazität berechnet wird, die unter dem maximalen ACU-Wert des DB-Clusters verfügbar ist. Aurora überwacht diesen Wert automatisch und skaliert Ihre Aurora Serverless v2-DB-Instance, wenn die DB-Instance konsequent einen hohen Anteil ihrer CPU-Kapazität verbraucht.

  Wenn sich diese Metrik dem Wert `100.0` nähert, hat die DB-Instance ihre maximale CPU-Kapazität erreicht. Erwägen Sie, die maximale ACU-Einstellung für den Cluster zu erhöhen. Wenn sich diese Metrik dem Wert `100.0` nähert, erwägen Sie bei einer Reader-DB-Instance, dem Cluster weitere Reader-DB-Instances hinzuzufügen. Auf diese Weise können Sie den schreibgeschützten Teil der Workload auf mehrere DB-Instances verteilen und so die Last jeder Reader-DB-Instance reduzieren.

- `FreeableMemory`: Dieser Wert stellt die Menge des nicht belegten Speichers dar, die verfügbar ist, wenn die Aurora Serverless v2-DB-Instance auf ihre maximale Kapazität skaliert wird. Für jede ACU, bei der die aktuelle Kapazität unter der maximalen Kapazität liegt, erhöht sich dieser Wert ungefähr um 2 GiB. Daher nähert sich diese Metrik erst null, wenn die DB-Instance so hoch wie möglich hochskaliert ist.

  Wenn sich diese Metrik dem Wert `0` nähert, ist die DB-Instance so weit wie möglich hochskaliert und nähert sich der Grenze ihres verfügbaren Speichers. Erwägen Sie, die maximale ACU-Einstellung für den Cluster zu erhöhen. Wenn sich diese Metrik dem Wert `0` nähert, erwägen Sie

bei einer Reader-DB-Instance, dem Cluster weitere Reader-DB-Instances hinzuzufügen. Auf diese Weise können Sie den schreibgeschützten Teil der Workload auf mehrere DB-Instances verteilen und so die Speicherauslastung für jede Reader-DB-Instance reduzieren.

- `TempStorageIOPS`: Die Anzahl der IOPS, die im lokalen Speicher durchgeführt werden, der der DB-Instance angefügt ist. Dabei sind IOPS für Lese- und Schreibvorgänge enthalten. Diese Metrik stellt eine Zählung dar und wird einmal pro Sekunde gemessen. Dies ist eine neue Metrik für Aurora Serverless v2. Details hierzu finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.
- `TempStorageThroughput`: Die Menge der mit der DB-Instance verknüpften Daten, die zu und aus dem lokalen Speicher übertragen wurden. Diese Metrik wird in Byte angegeben und einmal pro Sekunde gemessen. Dies ist eine neue Metrik für Aurora Serverless v2. Details hierzu finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.

In der Regel wird die Hochskalierung von Aurora Serverless v2-DB-Instances größtenteils durch Speicherauslastung und CPU-Aktivität verursacht. Die Metriken `TempStorageIOPS` und `TempStorageThroughput` können Ihnen helfen, die seltenen Fälle zu diagnostizieren, in denen die Netzwerkaktivität für Übertragungen zwischen Ihrer DB-Instance und lokalen Speichergeräten für unerwartete Kapazitätssteigerungen verantwortlich ist. Wenn Sie andere Netzwerkaktivitäten überwachen möchten, können Sie diese vorhandenen Metriken verwenden:

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`
- `StorageNetworkTransmitThroughput`

Sie können festlegen, dass Aurora einige oder alle Datenbankprotokolle in Amazon CloudWatch Logs veröffentlicht. Entsprechende Anleitungen für Ihre Datenbank-Engine finden Sie unter:

- the section called “Veröffentlichen von Aurora-PostgreSQL-Protokollen in CloudWatch Logs”
- the section called “Veröffentlichen von Aurora MySQL-Protokollen in CloudWatch Logs”

Wie Aurora Serverless v2-Metriken in Ihre AWS-Rechnung einfließen

Die Aurora Serverless v2-Gebühren auf Ihrer AWS-Rechnung werden basierend auf derselben `ServerlessDatabaseCapacity`-Metrik berechnet, die Sie überwachen können. Der Abrechnungsmechanismus kann von dem berechneten CloudWatch-Durchschnitt für diese Metrik abweichen, wenn Sie Aurora Serverless v2-Kapazität für nur einen Teil einer Stunde nutzen. Er kann auch abweichen, wenn die CloudWatch-Metrik für kurze Zeiträume aufgrund von Systemproblemen nicht verfügbar ist. Daher sehen Sie möglicherweise einen etwas anderen Wert von ACU-Stunden auf Ihrer Rechnung, als wenn Sie die Anzahl selbst anhand des `ServerlessDatabaseCapacity`-Durchschnittswerts berechnen.

Beispiele für CloudWatch-Befehle für Aurora Serverless v2-Metriken

Die folgenden AWS CLI-Beispiele zeigen, wie Sie die wichtigsten CloudWatch-Metriken im Zusammenhang mit Aurora Serverless v2 überwachen können. Ersetzen Sie in jedem Fall die `Value=`-Zeichenfolge für den `--dimensions`-Parameter durch den Bezeichner Ihrer eigenen Aurora Serverless v2-DB-Instance.

Im folgenden Linux-Beispiel werden die minimalen, maximalen und durchschnittlichen Kapazitätswerte für eine DB-Instance angezeigt, die alle 10 Minuten über einen Zeitraum von einer Stunde gemessen werden. Der Linux-Befehl `date` gibt die Start- und Endzeiten relativ zum aktuellen Datum und zur aktuellen Uhrzeit an. Die `sort_by`-Funktion im `--query`-Parameter sortiert die Ergebnisse chronologisch basierend auf dem Feld `Timestamp`.

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Die folgenden Linux-Beispiele veranschaulichen die Überwachung der Kapazität jeder DB-Instance in einem Cluster. Sie messen die minimale, maximale und durchschnittliche Kapazitätsauslastung jeder DB-Instance. Die Messungen werden einmal pro Stunde über einen Zeitraum von drei Stunden durchgeführt. Diese Beispiele verwenden die `ACUUtilization`-Metrik, die einen Prozentsatz der Obergrenze für ACUs darstellt, anstelle der `ServerlessDatabaseCapacity`-Metrik, die eine feste Anzahl von ACUs darstellt. Auf diese Weise müssen Sie die tatsächlichen Zahlen für die minimalen und maximalen ACU-Werte im Kapazitätsbereich nicht kennen. Sie können Prozentsätze zwischen 0 und 100 sehen.

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Im folgenden Linux-Beispiel werden ähnliche Messungen wie die vorherigen ausgeführt. In diesem Fall gelten die Messungen für die `CPUUtilization`-Metrik Die Messungen werden alle zehn Minuten über einen Zeitraum von einer Stunde durchgeführt. Die Zahlen stellen den Prozentsatz der verwendeten verfügbaren CPU dar, basierend auf den CPU-Ressourcen, die für die maximale Kapazitätseinstellung für die DB-Instance verfügbar sind.

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Im folgenden Linux-Beispiel werden ähnliche Messungen wie die vorherigen ausgeführt. In diesem Fall gelten die Messungen für die `FreeableMemory`-Metrik Die Messungen werden alle zehn Minuten über einen Zeitraum von einer Stunde durchgeführt.

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Überwachung der Aurora Serverless v2-Leistung mit Performance Insights

Mit Performance Insights können Sie die Leistung von Aurora Serverless v2-DB-Instances überwachen. Informationen zu Performance-Insights-Verfahren finden Sie unter Überwachung mit Performance Insights auf .

Die folgenden neuen Performance-Insights-Zähler gelten für Aurora Serverless v2-DB-Instances:

- `os.general.serverlessDatabaseCapacity` – Die aktuelle Kapazität der DB-Instance in ACUs. Der Wert entspricht der `ServerlessDatabaseCapacity`-CloudWatch-Metrik für die DB-Instance.
- `os.general.acuUtilization` – Der Anteil der aktuellen Kapazität an der maximal konfigurierten Kapazität in Prozent. Der Wert entspricht der `ACUUtilization`-CloudWatch-Metrik für die DB-Instance.
- `os.general.maxConfiguredAcu` – Die maximale Kapazität, die Sie für diese Aurora Serverless v2-DB-Instance konfiguriert haben. Sie wird in ACUs gemessen.
- `os.general.minConfiguredAcu` – Die Mindestkapazität, die Sie für diese Aurora Serverless v2-DB-Instance konfiguriert haben. Sie wird in ACUs gemessen.

Eine vollständige Liste der Performance-Insights-Zähler finden Sie unter Performance-Insights-Zählermetriken.

Wenn vCPU-Werte für eine Aurora Serverless v2-DB-Instance in Performance Insights angezeigt werden, stellen diese Werte Schätzungen dar, die auf dem ACU-Wert für die DB-Instance basieren. Im Standardintervall von einer Minute werden alle fraktionierten vCPU-Werte auf die nächste ganze Zahl aufgerundet. Für längere Intervalle ist der angezeigte vCPU-Wert der Durchschnitt der ganzzahligen vCPU-Werte für jede Minute.

Behebung von Kapazitätsproblemen bei Aurora Serverless v2

In einigen Fällen wird Aurora Serverless v2 nicht auf die Mindestkapazität herunterskaliert, auch wenn keine Last für die Datenbank vorliegt. Dies kann aus einem der folgenden Gründe geschehen:

- Bestimmte Funktionen können die Ressourcennutzung erhöhen und verhindern, dass die Datenbank auf die Mindestkapazität herunterskaliert wird. Nachstehend sind einige dieser Features aufgeführt:
  - Globale Aurora-Datenbanken

- Export von CloudWatch Logs
- Aktivieren von `pg_audit` auf Aurora PostgreSQL kompatiblen DB-Clustern
- „Enhanced Monitoring“ (Erweiterte Überwachung)
- Performance Insights

Weitere Informationen finden Sie unter Auswählen der minimalen Aurora Serverless v2-Kapazitätseinstellung für einen Cluster.

- Wenn eine Reader-Instance nicht auf das Minimum herunterskaliert wird und dieselbe oder eine höhere Kapazität als die Writer-Instance hat, überprüfen Sie die Prioritätsstufe der Reader-Instance. Reader-DB-Instances von Aurora Serverless v2 in Tier 0 oder 1 werden auf einer Mindestkapazität gehalten, die mindestens so hoch ist wie die der Writer-DB-Instance. Ändern Sie die Prioritätsstufe der Reader-Instance in 2 oder höher, sodass sie unabhängig von der Writer-Instance hoch- und herunterskaliert wird. Weitere Informationen finden Sie unter Auswählen der Hochstufungsstufe für einen Aurora Serverless v2-Reader.
- Legen Sie alle Datenbankparameter, die sich auf die Größe des gemeinsam genutzten Speichers auswirken, auf ihre Standardwerte fest. Wenn Sie einen höheren Wert als den Standardwert festlegen, erhöht sich der gemeinsam genutzte Speicherbedarf und verhindert, dass die Datenbank auf die Mindestkapazität herunterskaliert wird. Beispiele sind `max_connections` und `max_locks_per_transaction`.

  > **Note**
  >
  > Das Aktualisieren gemeinsam genutzter Speicherparameter erfordert einen Neustart der Datenbank, damit Änderungen wirksam werden.

- Hohe Datenbank-Workloads können die Ressourcennutzung erhöhen.
- Große Datenbank-Volumes können die Ressourcennutzung erhöhen.

  Amazon Aurora verwendet Speicher- und CPU-Ressourcen für die DB-Cluster-Verwaltung. Aurora benötigt mehr CPU und Arbeitsspeicher, um DB-Cluster mit größeren Datenbank-Volumes zu verwalten. Wenn die Mindestkapazität Ihres Clusters unter der für die Clusterverwaltung erforderlichen Mindestkapazität liegt, skaliert Ihr Cluster nicht auf die Mindestkapazität herunter.
- Hintergrundprozesse wie das Bereinigen können ebenfalls den Ressourcenverbrauch erhöhen.
- Einschränkungen der Plattformversion können sich auf die Skalierungsmöglichkeiten auswirken. Der verfügbare Skalierungsbereich für einen bestimmten Cluster wird sowohl von der Engine-Version als auch von der Hardware (Plattformversion) bestimmt. Es ist möglich, eine

leistungsstärkere Engine-Version auf einer weniger leistungsstarken Plattformversion auszuführen und umgekehrt.

Wenn die Datenbank immer noch nicht auf die konfigurierte Mindestkapazität herunterskaliert wird, beenden Sie die Datenbank und starten Sie sie neu, um alle Speicherfragmente zurückzugewinnen, die sich im Laufe der Zeit angesammelt haben könnten. Das Stoppen und Starten einer Datenbank führt zu Ausfallzeiten, daher empfehlen wir, achtsam vorzugehen.

Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2

Sie können angeben, dass Aurora Serverless v2-DB-Instances auf 0 ACUs herunterskaliert und automatisch angehalten werden, wenn innerhalb eines bestimmten Zeitraums keine durch Benutzeraktivitäten initiierte Verbindungen vorhanden sind. Dazu geben Sie für Ihren DB-Cluster einen ACU-Mindestwert von 0 an. Ihnen wird die Instance-Kapazität nicht in Rechnung gestellt, solange sich eine Instance im angehaltenen Status befindet. Die Aktivierung der Funktion zum automatischen Anhalten und Fortsetzen (Auto-Pause) für Aurora-Cluster, die wenig genutzt werden oder längere Zeit inaktiv sind, kann Ihnen helfen, die Kosten für Ihre Datenbankflotte zu kontrollieren.

> **Note**
>
> Die Auto-Pause-Funktion ist für Aurora Serverless v2 mit Aurora PostgreSQL und Aurora MySQL verfügbar. Möglicherweise müssen Sie die Engine-Version Ihrer Aurora-Datenbank aktualisieren, um diese Funktion nutzen zu können. Informationen zu den Engine-Versionen, für die eine Mindestkapazität von 0 ACUs verfügbar ist, finden Sie unter Kapazität von Aurora Serverless v2.

Themen

- Überblick über die Auto-Pause-Funktion von Aurora Serverless v2
- Voraussetzungen und Einschränkungen für die Auto-Pause-Funktion in Aurora Serverless v2
- Aktivieren und Deaktivieren der Auto-Pause-Funktion
- Funktionsweise der Auto-Pause-Funktion in Aurora Serverless v2
- Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für verschiedene Arten von Aurora-Clustern
- Überwachung von Aurora-Clustern, die die Auto-Pause-Funktion verwenden
- Fehlerbehebung für die Auto-Pause-Funktion von Aurora Serverless v2
- Überlegungen zum Anwendungsdesign für die Auto-Pause-Funktion von Aurora Serverless v2

Überblick über die Auto-Pause-Funktion von Aurora Serverless v2

Aurora Serverless v2-DB-Instances können nach einem Zeitraum ohne Benutzerverbindungen automatisch angehalten und automatisch fortgesetzt werden, wenn eine Verbindungsanfrage

eingeht. Die Funktion zum automatischen Anhalten/Fortsetzen in Aurora Serverless v2 hilft bei der Kostenkontrolle für Systeme, die kein strenges Service Level Objective (SLO) haben. Sie können diese Funktion beispielsweise für Cluster aktivieren, die für Entwicklung und Tests verwendet werden, oder für interne Anwendungen, bei denen eine kurze Pause akzeptabel ist, während die Verbindung zur Datenbank wieder aufgenommen wird. Wenn Ihre Workload Phasen der Inaktivität aufweist und leichte Verzögerungen beim Verbindungsaufbau toleriert werden können, während die Verbindung zur Instance wieder aufgenommen wird, sollten Sie die Auto-Pause-Funktion für Ihre Aurora Serverless v2-Instances in Betracht ziehen, um die Kosten zu senken.

Sie steuern dieses Verhalten, indem Sie angeben, ob die Aurora Serverless v2-DB-Instances in einem Cluster automatisch angehalten werden können oder nicht und wie lange jede Instance inaktiv sein muss, bevor sie angehalten wird. Um das die Auto-Pause-Verhalten für alle Aurora Serverless v2-DB-Instances in einem Aurora-Cluster zu aktivieren, legen Sie den Mindestkapazitätswert für den Cluster auf 0 ACUs fest.

Wenn Sie zuvor die Aurora Serverless v1-Funktion genutzt haben, die nach einer gewissen Zeit der Inaktivität auf 0 ACUs skaliert hat, können Sie ein Upgrade auf Aurora Serverless v2 ausführen und die entsprechende Auto-Pause-Funktion verwenden.

Die Vorteile der Auto-Pause-Funktion zur Kosteneinsparung ähneln denen der Stop/Start-Funktion für Cluster. Das automatische Anhalten für Aurora Serverless v2 bietet den zusätzlichen Vorteil einer schnelleren Wiederaufnahme als das Starten eines gestoppten Clusters. Außerdem wird der Prozess automatisiert, mit dem bestimmt wird, wann jede DB-Instance angehalten und wieder fortgesetzt werden soll.

Die Auto-Pause-Funktion bietet außerdem zusätzliche Granularität bei der Kontrolle der Kosten für Rechenressourcen innerhalb Ihres Clusters. Sie können festlegen, dass einige Reader-Instances angehalten werden, auch wenn die Writer-Instance und andere Reader im Cluster jederzeit aktiv bleiben. Dazu weisen Sie den Reader-Instances, die unabhängig von anderen Instances angehalten werden können, eine Failover-Priorität im Bereich 2–15 zu.

Die Writer-Instance und alle Reader-Instances mit der Failover-Priorität 0 und 1 werden immer gleichzeitig angehalten und fortgesetzt. Daher werden die Instances in dieser Gruppe angehalten, nachdem keine von ihnen für das angegebene Zeitintervall Verbindungen hergestellt hat.

Aurora-DB-Cluster können eine Kombination aus Writer- und Reader-DB-Instances sowie bereitgestellten und Aurora Serverless v2-DB-Instances enthalten. Um diese Funktion effektiv nutzen zu können, ist es daher hilfreich, die folgenden Aspekte des Auto-Pause-Mechanismus zu kennen:

- die Umstände, unter denen eine DB-Instance möglicherweise automatisch angehalten wird.

- wann verhindert werden könnte, dass eine DB-Instance angehalten wird. Wenn Sie beispielsweise einige Aurora-Funktionen aktivieren oder bestimmte Arten von Vorgängen auf dem Cluster ausführen, können Instances möglicherweise nicht angehalten werden, auch wenn keine Verbindungen zu diesen Instances bestehen.
- die Folgen für die Überwachung und Abrechnung, wenn eine Instance pausiert ist.
- welche Aktionen bewirken, dass eine DB-Instance die Verarbeitung fortsetzt.
- wie sich die Kapazität einer DB-Instance zum Zeitpunkt der Ereignisse zum Anhalten und Fortsetzen ändert.
- wie Sie das Leerlaufintervall steuern, bevor eine DB-Instance angehalten wird.
- wie Sie die Anwendungslogik so programmieren, dass sie den Zeitraum behandelt, in dem eine DB-Instance die Verarbeitung fortsetzt.

Voraussetzungen und Einschränkungen für die Auto-Pause-Funktion in Aurora Serverless v2

Bevor Sie die Auto-Pause-Funktion verwenden, überprüfen Sie, welche Engine-Versionen Sie verwenden möchten. Prüfen Sie außerdem, ob die Auto-Pause-Funktion in Kombination mit den anderen Aurora-Funktionen funktioniert, die Sie verwenden möchten. Sie können die Auto-Pause-Funktion nicht aktivieren, wenn Sie eine Engine-Version verwenden, die sie nicht unterstützt. Bei inkompatiblen Funktionen wird keine Fehlermeldung angezeigt, wenn Sie sie in Kombination mit der Auto-Pause-Funktion verwenden. Wenn der Cluster inkompatible Funktionen oder Einstellungen verwendet, werden die Aurora Serverless v2-Instances nicht automatisch angehalten.

- Wenn Sie Aurora PostgreSQL verwenden, muss auf der Datenbank-Engine mindestens Version 16.3, 15.7, 14.12 oder 13.15 ausgeführt werden.
- Wenn Sie Aurora MySQL verwenden, muss auf der Datenbank-Engine Version 3.08.0 oder höher ausgeführt werden.
- Eine vollständige Liste der Engine-Versionen und AWS-Regionen, in denen diese Funktion verfügbar ist, finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2.
- Wenn eine Aurora Serverless v2-Instance fortgesetzt wird, ist ihre Kapazität möglicherweise geringer als zu dem Zeitpunkt, als sie angehalten wurde. Details hierzu finden Sie unter Unterschiede im Verhalten der Auto-Pause-Funktion zwischen Aurora Serverless v2 und Aurora Serverless v1.

Bestimmte Bedingungen oder Einstellungen verhindern, dass Aurora Serverless v2-Instances automatisch angehalten werden. Weitere Informationen finden Sie unter Situationen, in denen Aurora Serverless v2 nicht automatisch angehalten wird.

Aktivieren und Deaktivieren der Auto-Pause-Funktion

Sie können die Auto-Pause-Funktion auf Cluster-Ebene aktivieren und deaktivieren. Dazu verwenden Sie die gleichen Verfahren wie beim Anpassen der Mindest- und Höchstkapazität für den Cluster. Die Auto-Pause-Funktion wird durch eine Mindestkapazität von 0 ACUs dargestellt.

Themen

- Aktivieren der Auto-Pause-Funktion für Aurora Serverless v2-Instances in einem Cluster
- Konfigurierbares Timeout-Intervall für die Auto-Pause-Funktion in Aurora Serverless v2
- Fortsetzen einer automatisch angehaltenen Aurora Serverless v2-Instance
- Deaktivieren der Auto-Pause-Funktion für Aurora Serverless v2-Instances in einem Cluster

Aktivieren der Auto-Pause-Funktion für Aurora Serverless v2-Instances in einem Cluster

Folgen Sie dem Verfahren unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster. Wählen Sie als Mindestkapazität 0 ACUs aus. Wenn Sie eine Mindestkapazität von 0 ACUs auswählen, können Sie auch angeben, wie lange die Instance inaktiv sein soll, bevor sie automatisch angehalten wird.

Das folgende CLI-Beispiel zeigt, wie Sie einen Aurora-Cluster mit aktivierter Auto-Pause-Funktion und einem auf zehn Minuten (600 Sekunden) eingestellten Auto-Pause-Intervall erstellen.

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0 \
    --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=4,SecondsUntilAutoPause=600 \
    --master-username myuser \
    --manage-master-user-password
```

Das folgende CLI-Beispiel zeigt, wie Sie die Auto-Pause-Funktion für einen vorhandenen Aurora-Cluster aktivieren. In diesem Beispiel wird das Auto-Pause-Intervall auf eine Stunde (3600 Sekunden) festgelegt.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration
 MinCapacity=0,MaxCapacity=80,SecondsUntilAutoPause=3600

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 0,
    "MaxCapacity": 80.0,
    "SecondsUntilAutoPause": 3600
}
```

Konfigurierbares Timeout-Intervall für die Auto-Pause-Funktion in Aurora Serverless v2

Das Timeout-Intervall wird im `ServerlessV2ScalingConfiguration`-Attribut Ihres Aurora-Clusters dargestellt. Sie können dieses Intervall beim Erstellen oder Ändern eines Aurora-Clusters in der AWS-Managementkonsole angeben, wenn die Mindestkapazität auf 0 ACUs festgelegt ist. Sie können es in der AWS CLI angeben, indem Sie den `--serverless-v2-scaling-configuration`-Parameter verwenden, wenn Sie einen Aurora-Cluster erstellen oder ändern. Sie können es in der RDS-API angeben, indem Sie den `ServerlessV2ScalingConfiguration`-Parameter verwenden, wenn Sie einen Aurora-Cluster erstellen oder ändern.

Das Mindestintervall, das Sie festlegen können, beträgt 300 Sekunden (fünf Minuten). Dies ist die Standardeinstellung, wenn Sie kein Intervall angeben. Das maximale Intervall, das Sie festlegen können, beträgt 86.400 Sekunden (einen Tag).

Angenommen, Sie deaktivieren die Auto-Pause-Funktion für einen Cluster, indem Sie die Mindestkapazität des Clusters auf einen Wert ungleich 0 ändern. In diesem Fall wird die Intervalleigenschaft aus dem `ServerlessV2ScalingConfiguration`-Attribut entfernt. Das Fehlen dieser Eigenschaft ist eine zusätzliche Bestätigung dafür, dass die Auto-Pause-Funktion für diesen Cluster deaktiviert ist. Wenn Sie die Auto-Pause-Funktion später wieder aktivieren, können Sie zu diesem Zeitpunkt erneut ein beliebiges benutzerdefiniertes Intervall angeben.

Fortsetzen einer automatisch angehaltenen Aurora Serverless v2-Instance

- Wenn Sie eine Verbindung zu einer angehaltenen Aurora Serverless v2-Instance herstellen, wird die Verbindung automatisch fortgesetzt und akzeptiert.
- Ein Verbindungsversuch, der keine gültigen Anmeldeinformationen enthält, führt dennoch dazu, dass die DB-Instance fortgesetzt wird.
- Wenn Sie eine Verbindung über den Writer-Endpunkt herstellen, setzt Aurora die Writer-DB-Instance fort, falls sie automatisch angehalten wurde. Gleichzeitig setzt Aurora alle automatisch angehaltenen Reader-Instances fort, die die Failover-Priorität 0 oder 1 haben, was bedeutet, dass ihre Kapazität an die Kapazität der Writer-Instance gebunden ist.
- Wenn Sie eine Verbindung über den Reader-Endpunkt herstellen, wählt Aurora nach dem Zufallsprinzip eine Reader-Instance aus. Wenn diese Reader-Instance angehalten ist, setzt Aurora sie fort. Aurora setzt außerdem zuerst die Writer-Instance fort, da die Writer-Instance immer aktiv sein muss, wenn Reader-Instances aktiv sind. Wenn Aurora diese Writer-Instance fortsetzt, führt dies auch dazu, dass alle Reader-Instances in den Failover-Hochstufungsstufen 0 und 1 fortgesetzt werden.
- Wenn Sie über die RDS-Daten-API eine Anfrage an Ihren Cluster senden, setzt Aurora die Writer-Instance fort, falls sie angehalten wurde. Dann verarbeitet Aurora die Abfrage der Daten-API.
- Wenn Sie den Wert eines Konfigurationsparameters in einer DB-Cluster-Parametergruppe ändern, setzt Aurora automatisch alle angehaltenen Aurora Serverless v2-Instances in allen Clustern fort, die diese Cluster-Parametergruppe verwenden. Wenn Sie einen Parameterwert in einer DB-Parametergruppe ändern, setzt Aurora entsprechend automatisch alle angehaltenen Aurora Serverless v2-Instances fort, die diese DB-Parametergruppe verwenden. Das gleiche Verhalten für das automatische Fortsetzen gilt, wenn Sie einen Cluster ändern, um eine andere Cluster-Parametergruppe zuzuweisen, oder wenn Sie eine Instance ändern, um eine andere DB-Parametergruppe zuzuweisen.
- Wenn Sie eine Rückverfolgungsanfrage ausführen, wird die Aurora Serverless v2-Writer-Instance automatisch fortgesetzt, falls sie angehalten wurde. Aurora verarbeitet die Rückverfolgungsanfrage, nachdem die Writer-Instance fortgesetzt wurde. Sie können die Rückverfolgung zu einer Zeit ausführen, zu der eine Aurora Serverless v2-Instance angehalten war.
- Das Erstellen eines Cluster-Snapshots oder das Löschen eines Snapshots führt nicht dazu, dass Aurora Serverless v2-Instances wieder aufgenommen werden.
- Durch das Erstellen eines Aurora-Klons setzt Aurora die Writer-Instance des Clusters fort, der geklont wird.

- Wenn eine angehaltene Instance eine große Anzahl von Verbindungsanfragen empfängt, bevor die Fortsetzung abgeschlossen ist, können einige Sitzungen möglicherweise keine Verbindung herstellen. Wir empfehlen, die Wiederholungslogik für Verbindungen zu Aurora-Clustern zu implementieren, bei denen einige Aurora Serverless v2-Instances mit aktivierter Auto-Pause-Funktion vorhanden sind. Sie können beispielsweise jede fehlgeschlagene Verbindung dreimal wiederholen.
- Aurora kann einige Arten kleinerer interner Wartungsarbeiten durchführen, ohne eine Instance zu aktivieren. Für bestimmte Wartungsarten, die während des Wartungsfensters des Clusters durchgeführt werden, muss Aurora die Instance jedoch fortsetzen. Wenn die Wartung abgeschlossen ist, wird die Instance automatisch wieder angehalten, wenn nach dem angegebenen Intervall keine Aktivität mehr stattfindet.

> **Note**
>
> Aurora setzt eine angehaltene Instance für Engine-spezifische geplante Aufträge, wie die in der `pg_cron`-PostgreSQL-Erweiterung oder dem MySQL-Ereignisplaner, nicht automatisch fort. Um sicherzustellen, dass solche Aufträge ausgeführt werden, stellen Sie vor dem geplanten Zeitpunkt manuell eine Verbindung zur Instance her. Aurora stellt keine Aufträge in die Warteschlange, bei denen die geplante Zeit in der Zeit liegt, in der die DB-Instance angehalten ist. Solche Aufträge werden übersprungen, wenn die Instance später wieder fortgesetzt wird.

- Wenn der Aurora-Cluster einen Failover durchläuft, während eine Aurora Serverless v2-Instance automatisch angehalten ist, setzt Aurora möglicherweise eine Instance fort und stuft diese Instance dann zum Writer hoch. Das Gleiche kann passieren, wenn eine oder mehrere DB-Instances aus dem Cluster entfernt werden, während eine Instance angehalten ist. In diesem Fall wird die Instance sofort zum Writer, wenn sie fortgesetzt wird.
- Operationen, die die Eigenschaften des Clusters ändern, führen auch dazu, dass alle automatisch angehaltenen Aurora Serverless v2-Instances fortgesetzt werden. Beispielsweise wird eine automatisch angehaltene Instance für folgende Operationen fortgesetzt:
  - Ändern des Skalierungsbereichs des Clusters.
  - Aktualisieren der Engine-Version des Clusters.
  - Beschreiben oder Herunterladen von Protokolldateien aus einer angehaltenen Instance. Sie können historische Protokolldaten von angehaltenen Instances untersuchen, indem Sie Protokolluploads auf CloudWatch aktivieren und die Logs über CloudWatch analysieren.

Deaktivieren der Auto-Pause-Funktion für Aurora Serverless v2-Instances in einem Cluster

Folgen Sie dem Verfahren unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster. Wählen Sie als Mindestkapazität einen Wert von 0,5 oder höher aus. Wenn Sie die Auto-Pause-Funktion deaktivieren, wird das Intervall, in dem sich die Instance im Leerlauf befindet, zurückgesetzt. Wenn Sie die Auto-Pause-Funktion wieder aktivieren, geben Sie ein neues Timeout-Intervall an.

Das folgende CLI-Beispiel zeigt, wie Sie die Auto-Pause-Funktion für einen vorhandenen Aurora-Cluster deaktivieren. Die `describe-db-clusters`-Ausgabe zeigt, dass das `SecondsUntilAutoPause`-Attribut entfernt wird, wenn die Mindestkapazität auf einen Wert ungleich 0 gesetzt wird.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=2,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 2,
    "MaxCapacity": 80.0
}
```

Funktionsweise der Auto-Pause-Funktion in Aurora Serverless v2

Sie können die folgenden Informationen verwenden, um Ihre Nutzung der Auto-Pause-Funktion zu planen. Wenn Sie wissen, unter welchen Umständen Instances angehalten oder fortgesetzt werden oder aktiv bleiben, können Sie ein ausgewogenes Verhältnis zwischen Verfügbarkeit, Reaktionsfähigkeit und Kosteneinsparungen finden.

Themen

- Was passiert, wenn Aurora Serverless v2-Instances angehalten werden
- Was passiert, wenn automatisch angehaltene Aurora Serverless v2-Instances fortgesetzt werden
- Funktionsweise der Instance-Abrechnung für automatisch angehaltene Aurora Serverless v2-Cluster
- Situationen, in denen Aurora Serverless v2 nicht automatisch angehalten wird
- Funktionsweise der Auto-Pause-Funktion mit der Stoppen/Starten-Funktion für Cluster

- Funktionsweise von Wartung und Upgrades für automatisch angehaltene Aurora Serverless v2-Cluster
- Unterschiede im Verhalten der Auto-Pause-Funktion zwischen Aurora Serverless v2 und Aurora Serverless v1

Was passiert, wenn Aurora Serverless v2-Instances angehalten werden

Wenn eine Aurora Serverless v2-DB-Instance nach einem Zeitraum ohne Verbindungen angehalten wird:

- Aurora beginnt, die Instance nach Ablauf des angegebenen Intervalls ohne Verbindungen zur Instance anzuhalten, unabhängig davon, wie viele ACUs die Instance zu diesem Zeitpunkt hat.
- Das Anhalten erfolgt nicht sofort. Eine Aurora Serverless v2-Instance, die kurz davor steht, automatisch angehalten zu werden, wartet möglicherweise kurz, bis alle Änderungen am Aurora-Speicher übernommen wurden.
- Die Instance-Gebühren für diese Instance werden ausgesetzt. Die `ServerlessV2Usage`-Metrik hat den Wert 0, während die Instance angehalten ist.
- Der Statuswert für die Instance ändert sich nicht. Der Status wird weiterhin als „verfügbar“ angezeigt.
- Die Instance beendet das Schreiben in die Datenbank-Protokolldateien. Sie sendet keine Metriken mehr an CloudWatch, außer dass null 0 Prozent für `CPUUtilization` und `ACUUtilization` sowie 0 für `ServerlessDatabaseCapacity` registriert werden.
- Aurora gibt Ereignisse aus, wenn mit dem Anhalten einer Aurora Serverless v2-DB-Instance begonnen wird, das Anhalten beendet wird und wenn der Mechanismus zum Anhalten unterbrochen wird oder nicht erfolgreich ist. Weitere Informationen zu diesen Ereignissen finden Sie unter DB-Instance-Ereignisse.

Was passiert, wenn automatisch angehaltene Aurora Serverless v2-Instances fortgesetzt werden

Wenn eine Aurora Serverless v2-DB-Instance fortgesetzt wird, nachdem sie automatisch angehalten wurde, gilt Folgendes:

- Alle Parameteränderungen, die sich in `pending-reboot`-Änderungen befinden, werden angewendet, wenn die Instance fortgesetzt wird.

- Aurora gibt Ereignisse auf Instance-Ebene aus, wenn jede Aurora Serverless v2-DB-Instance mit dem Fortsetzen beginnt, das Fortsetzen beendet wird und wenn die Instance aus irgendeinem Grund nicht fortgesetzt werden kann. Weitere Informationen zu diesen Ereignissen finden Sie unter DB-Instance-Ereignisse.
- Alle angeforderten Verbindungen werden hergestellt, nachdem die DB-Instance das Fortsetzen abgeschlossen hat. Da die typische Zeit für das Fortsetzen etwa 15 Sekunden betragen kann, empfehlen wir, dass Sie alle Einstellungen für das Client-Timeout so anpassen, dass sie länger als 15 Sekunden sind. Beispielsweise können Sie in Ihren JDBC-Treibereinstellungen die Werte für die Einstellungen `connectTimeout` und `sslResponseTimeout` so anpassen, dass sie länger als 15 Sekunden sind.

> **Note**
>
> Wenn eine Aurora Serverless v2-Instance länger als 24 Stunden angehalten wird, kann Aurora die Instance in einen tieferen Ruhezustand versetzen, bei dem das Fortsetzen länger dauert. In diesem Fall kann die Zeit bis zur Fortsetzung 30 Sekunden oder länger betragen, was in etwa einem Neustart der Instance entspricht. Wenn Ihre Datenbank Inaktivitätsperioden aufweist, die länger als einen Tag andauern, empfehlen wir, Verbindungs-Timeouts auf 30 Sekunden oder mehr festzulegen.

Funktionsweise der Instance-Abrechnung für automatisch angehaltene Aurora Serverless v2-Cluster

Während eine Aurora Serverless v2-Instance automatisch angehalten ist, ist ihre Instance-Gebühr gleich 0. Die `ServerlessV2Usage`-Metrik ist in diesem Zeitraum 0. AWS berechnet weiterhin Gebühren für Aurora-Speicher und andere Aspekte des Clusters, die nicht an diese spezifische DB-Instance gebunden sind.

Situationen, in denen Aurora Serverless v2 nicht automatisch angehalten wird

- Wenn der Mindestkapazitätswert für Ihren DB-Cluster höher als 0 ACUs ist, werden die Aurora Serverless v2-Instances im Cluster nicht automatisch angehalten. Wenn Sie bereits Cluster mit Aurora Serverless v2-Instances aus der Zeit haben, als die Auto-Pause-Funktion noch nicht verfügbar war, lag die niedrigste Mindestkapazitätseinstellung bei 0,5 ACUs. Um die Auto-Pause-Funktion mit solchen Clustern zu verwenden, ändern Sie die Einstellung für die Mindestkapazität auf 0 ACUs.

- Wenn vom Benutzer initiierte Verbindungen zu einer Aurora Serverless v2-Instance geöffnet sind, wird die Instance nicht angehalten. Die Instance wird auch nicht angehalten, während Aktivitäten wie Patches und Upgrades ausgeführt werden. Administrative Verbindungen, die Aurora für Zustandsprüfungen verwendet, werden nicht als Aktivität gezählt und verhindern nicht, dass die Instance angehalten wird.
- In einem Aurora-PostgreSQL-Cluster, für den die logische Replikation aktiviert ist, oder in einem Aurora-MySQL-Cluster, für den die Binlog-Replikation aktiviert ist, werden die Writer-Instance und alle Reader-Instances in den Failover-Hochstufungsstufen 0 und 1 nicht automatisch angehalten. Aurora führt eine konstante Mindestanzahl von Aktivitäten aus, um den Zustand der Replikationsverbindung zu überprüfen.

  Für Cluster mit aktivierter Replikation können Sie weiterhin Aurora-Reader-Instances im Quell-Cluster haben, die automatisch angehalten werden. Setzen Sie dazu ihre Failover-Priorität auf einen anderen Wert als 0 oder 1. Auf diese Weise können sie unabhängig von der Writer-Instance angehalten werden.
- Wenn Ihr Aurora-Cluster über einen zugehörigen RDS-Proxy verfügt, behält der Proxy eine offene Verbindung zu jeder DB-Instance im Cluster bei. Daher werden Aurora Serverless v2-Instances in einem solchen Cluster nicht automatisch angehalten.
- Wenn Ihr Cluster der primäre Cluster in einer globalen Aurora-Datenbank ist, hält Aurora die Aurora Serverless v2-Writer-Instance nicht automatisch an. Das liegt daran, dass auf der Writer-Instance ein konstantes Aktivitätsniveau erforderlich ist, um die anderen Cluster in der globalen Datenbank zu verwalten. Da die Writer-Instance aktiv bleibt, werden auch alle Aurora Serverless v2-Reader-Instances mit der Failover-Priorität 0 oder 1 nicht automatisch angehalten.
- Aurora Serverless v2-Instances in den sekundären Clustern einer globalen Aurora-Datenbank werden nicht automatisch angehalten. Wenn ein DB-Cluster zu einem eigenständigen Cluster heraufgestuft wird, wird die Auto-Pause-Funktion wirksam, sofern alle anderen Bedingungen erfüllt sind.
- In einem Cluster mit einer zugehörigen Null-ETL-Integration zu Redshift werden die Writer-Instance und alle Reader-Instances in den Failover-Promotion-Hochstufungsstufen 0 und 1 nicht automatisch angehalten.
- Wenn für eine Aurora PostgreSQL-Instance die Babelfish-Funktion aktiviert ist, verhindern alle Verbindungen und Aktivitäten am T-SQL-Port zusätzlich zur Aktivität auf dem Hauptdatenbankport für die Instance, dass die Instance automatisch angehalten wird.

Funktionsweise der Auto-Pause-Funktion mit der Stoppen/Starten-Funktion für Cluster

Sie können einen Aurora-Cluster beenden und starten, wenn die Auto-Pause-Funktion aktiviert ist. Es spielt keine Rolle, ob einige Instances angehalten sind. Wenn Sie den Cluster erneut starten, werden alle angehaltenen Aurora Serverless v2-Instances automatisch fortgesetzt.

Solange ein Aurora-Cluster gestoppt ist, werden angehaltene Aurora Serverless v2-Instances nicht automatisch aufgrund von Verbindungsversuchen fortgesetzt. Sobald der Cluster erneut gestartet wird, gelten die üblichen Mechanismen zum Anhalten und Fortsetzen von Aurora Serverless v2-Instances.

Funktionsweise von Wartung und Upgrades für automatisch angehaltene Aurora Serverless v2-Cluster

- Wenn eine Aurora Serverless v2-Instance automatisch angehalten wurde und Sie versuchen, den Aurora-Cluster zu aktualisieren, setzt Aurora die Instance fort und aktualisiert sie.
- Aurora setzt alle automatisch angehaltenen Aurora Serverless v2-Instances regelmäßig fort, um Wartungsarbeiten wie Unterversion-Upgrades und Änderungen an Eigenschaften wie Parametergruppen durchzuführen.
- Nachdem eine Aurora Serverless v2-Instance für einen administrativen Vorgang wie ein Upgrade oder die Wartung wieder aktiviert wurde, wartet Aurora mindestens 20 Minuten, bevor diese Instance wieder angehalten wird. Dadurch können alle Hintergrundvorgänge abgeschlossen werden. Durch den Zeitraum von zwanzig Minuten wird außerdem vermieden, dass die Instance mehrmals angehalten und fortgesetzt wird, wenn auf der Instance mehrere Verwaltungsvorgänge nacheinander ausgeführt werden.

Unterschiede im Verhalten der Auto-Pause-Funktion zwischen Aurora Serverless v2 und Aurora Serverless v1

- Die Fortsetzungszeit wurde in Aurora Serverless v2 im Vergleich zu Aurora Serverless v1 verbessert. Bis zur Fortsetzung dauert es in der Regel etwa 15 Sekunden, wenn die Instance für weniger als 24 Stunden angehalten wurde. Wenn die Instance länger als 24 Stunden angehalten ist, kann das Fortsetzen länger dauern.
- Dadurch, wie Aurora Serverless v2 auf Multi-AZ-Cluster angewendet wird, können einige DB-Instances im Cluster möglicherweise angehalten sein, während andere aktiv sind. Die Writer-Instance wird immer dann fortgesetzt, wenn ein Reader ausgeführt wird, da der Writer zur

Koordination bestimmter Aktivitäten innerhalb des Clusters benötigt wird. Da Aurora Serverless v1 keine Reader-Instances verwendet, wäre immer der gesamte Cluster angehalten oder aktiv.

- Wenn der Reader-Endpunkt nach dem Zufallsprinzip eine Reader-Instance auswählt, zu der eine Verbindung hergestellt werden soll, ist diese Reader-Instance möglicherweise bereits aktiv oder wurde automatisch angehalten. Daher kann die Zeit für den Zugriff auf die Reader-Instance variieren und schwieriger vorherzusagen sein. Multi-AZ-Cluster, die Aurora Serverless v2 und die Auto-Pause-Funktion verwenden, könnten daher von der Einrichtung benutzerdefinierter Endpunkte für bestimmte schreibgeschützte Anwendungsfälle profitieren, anstatt alle schreibgeschützten Sitzungen an den Reader-Endpunkt weiterzuleiten.
- Aurora Serverless v2-Instances werden mit derselben Häufigkeit wie bereitgestellte Instances Wartungsvorgängen unterzogen. Da Aurora Instances automatisch fortsetzt, wenn eine solche Wartung erforderlich ist, stellen Sie möglicherweise fest, dass Aurora Serverless v2-Instances häufiger fortgesetzt werden als Aurora Serverless v1-Cluster.

Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für verschiedene Arten von Aurora-Clustern

Die Überlegungen zur Auto-Pause-Funktion hängen davon ab, wie viele Instances sich in Ihrem Aurora-Cluster befinden, welche Failover-Hochstufungsstufen die Reader-Instances haben und ob alle Instances Aurora Serverless v2 oder eine Kombination von Aurora Serverless v2 und bereitgestellt sind.

Themen

- Empfohlene Aurora-Cluster-Layouts bei Verwendung der Auto-Pause-Funktion
- Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für die Writer-Instance in einem DB-Cluster
- Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für Multi-AZ-Cluster
- Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für Cluster mit bereitgestellten Instances

Empfohlene Aurora-Cluster-Layouts bei Verwendung der Auto-Pause-Funktion

Wenn die Auto-Pause-Funktion aktiviert ist, können Sie Ihren Aurora-Cluster so einrichten, dass er das richtige Gleichgewicht zwischen Hochverfügbarkeit, schneller Reaktion und Skalierbarkeit für Ihren Anwendungsfall bietet. Dazu wählen Sie die entsprechende Kombination aus Aurora Serverless

v2-Instances, bereitgestellten Instances und Failover-Hochstufungsstufen für die DB-Instances in Ihrem Cluster aus.

Die folgenden Konfigurationstypen zeigen unterschiedliche Kompromisse zwischen Hochverfügbarkeit und Kostenoptimierung für Ihren Cluster:

- Für ein Entwicklungs- und Testsystem können Sie einen Single-AZ-DB-Cluster mit einer Aurora Serverless v2-DB-Instance einrichten. Die einzelne Instance bearbeitet alle Lese- und Schreibanforderungen. Wenn der Cluster für längere Zeit nicht verwendet wird, wird die DB-Instance angehalten. Zu diesem Zeitpunkt werden auch die DB-Datenverarbeitungskosten für Ihren Cluster angehalten.
- Für ein System, in dem eine Anwendung ausgeführt wird, bei der Hochverfügbarkeit Priorität hat, der Cluster aber immer noch Zeiten hat, in denen er vollständig inaktiv ist, können Sie einen Multi-AZ-Cluster einrichten, bei dem sowohl die Writer- als auch die Reader-DB-Instances Aurora Serverless v2 sind. Stellen Sie die Reader-Instance auf die Failover-Priorität 0 oder 1 ein, sodass die Writer- und die Reader-Instance gleichzeitig angehalten und fortgesetzt werden. Jetzt profitieren Sie von einem schnellen Failover, während der Cluster aktiv ist. Wenn der Cluster länger als der Auto-Pause-Schwellenwert inaktiv bleibt, werden die DB-Instance-Gebühren für beide Instances angehalten. Wenn der Cluster die Verarbeitung fortsetzt, benötigt die erste Datenbanksitzung eine kurze Zeit, um eine Verbindung herzustellen.
- Angenommen, Ihr Cluster ist ständig mit einer minimalen Aktivität aktiv und erfordert eine schnelle Reaktion für jede Verbindung. In diesem Fall können Sie einen Cluster mit mehr als einer Aurora Serverless v2-Reader-Instance erstellen und die Kapazitäten einiger Reader-Instances vom Writer entkoppeln. Geben Sie die Failover-Priorität 0 oder 1 für die Writer-Instance und eine Reader-Instance an. Geben Sie für die anderen Reader-Instances eine Priorität größer 1 an. Auf diese Weise können die Reader-Instances der höheren Prioritätsstufen automatisch angehalten werden, auch wenn der Writer und einer der Reader aktiv bleiben.

  In diesem Fall können Sie einige andere Techniken einsetzen, um sicherzustellen, dass der Cluster kontinuierlich verfügbar bleibt und gleichzeitig in Leerlaufphasen auf eine niedrige Kapazität herunterskaliert wird:

  - Sie können bereitgestellte Instances für den Writer und den Reader mit Priorität 0 oder 1 verwenden. Auf diese Weise werden zwei DB-Instances nie automatisch angehalten und sind immer verfügbar, um den Datenbankverkehr zu verarbeiten und Failover durchzuführen.
  - Sie können einen benutzerdefinierten Endpunkt einrichten, der die Aurora Serverless v2-Instances der höheren Prioritätsstufen enthält, aber nicht den Writer oder die Reader der Hochstufungsstufen 0 oder 1. Auf diese Weise können Sie schreibgeschützte Sitzungen,

die nicht latenzempfindlich sind, an die Reader weiterleiten, die möglicherweise automatisch angehalten werden. Sie können vermeiden, den Reader-Endpunkt für solche Anfragen zu verwenden, da Aurora Reader-Endpunktverbindungen möglicherweise an die Always-Awake-Reader-Instance oder an eine der automatisch angehaltenen Instances weiterleitet. Mithilfe des benutzerdefinierten Endpunkts können Sie Verbindungen zu verschiedenen Gruppen von Instances weiterleiten, je nachdem, ob Sie eine schnelle Reaktion oder zusätzliche Skalierungskapazität bevorzugen.

Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für die Writer-Instance in einem DB-Cluster

Wenn ein Aurora-DB-Cluster nur eine einzige DB-Instance enthält, ist der Mechanismus für das automatische Anhalten und Fortsetzen der DB-Instance unkompliziert. Er hängt nur von der Aktivität auf der Writer-Instance ab. Möglicherweise verfügen Sie über eine solche Konfiguration für Cluster, die für Entwicklung und Tests verwendet werden, oder für die Ausführung von Anwendungen, bei denen Hochverfügbarkeit nicht entscheidend ist. Beachten Sie, dass Aurora in einem Single-Instance-Cluster Verbindungen über den Reader-Endpunkt zur Writer-DB-Instance weiterleitet. Bei einem Single-Instance-DB-Cluster führt der Versuch, eine Verbindung mit dem Reader-Endpunkt herzustellen, dazu, dass die automatisch angehaltene Writer-DB-Instance fortgesetzt wird.

Die folgenden zusätzlichen Faktoren gelten für Aurora-Cluster mit mehreren DB-Instances:

- In einem Aurora-DB-Cluster wird in der Regel häufig auf die Writer-DB-Instance zugegriffen. Daher stellen Sie möglicherweise fest, dass die Writer-DB-Instance auch dann aktiv bleibt, wenn eine oder mehrere der Reader-DB-Instances automatisch angehalten werden.
- Bestimmte Aktivitäten auf den Reader-DB-Instances setzen voraus, dass diese Writer-DB-Instance verfügbar ist. Daher können die Writer-DB-Instances erst angehalten werden, wenn auch alle Reader-Instances angehalten wurden. Wenn Sie eine Reader-Instance fortsetzen, wird die Writer-Instance automatisch fortgesetzt, auch wenn Ihre Anwendung nicht direkt auf die Writer-Instance zugreift.
- Aurora Serverless v2-Reader-Instances in den Failover-Hochstufungsstufen 0 und 1 werden skaliert, um ihre Kapazität mit der Writer-Instance synchron zu halten. Wenn also eine Aurora Serverless v2-Writer-Instance fortgesetzt wird, gilt dies auch für alle Aurora Serverless v2-Reader-Instances, die sich in den Hochstufungsstufen 0 und 1 befinden.

Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für Multi-AZ-Cluster

In einem Aurora-DB-Cluster, der sowohl eine Writer- als auch eine oder mehrere Reader-DB-Instances enthält, können einige Aurora Serverless v2-DB-Instances angehalten sein, während andere DB-Instances aktiv sind. Die Writer-Instance und alle Reader-Instances mit der Failover-Priorität 0 und 1 werden immer alle gleichzeitig angehalten und fortgesetzt. Reader-Instances mit einer anderen Priorität als 0 oder 1 können unabhängig von den anderen Instances angehalten und fortgesetzt werden.

Wenn Sie diese Funktion für Cluster mit mehreren Reader-Instances verwenden, können Sie die Kosten verwalten, ohne Abstriche bei der Hochverfügbarkeit machen zu müssen. Die Writer-Instance und ein oder zwei weitere Reader-Instances können jederzeit aktiv bleiben, während weitere Reader-Instances angehalten werden können, wenn sie nicht für das Verarbeiten von mit hohem Leseverkehrsvolumen benötigt werden.

Ob eine Reader-Instance automatisch angehalten werden kann, hängt davon ab, ob ihre Kapazität unabhängig skaliert werden kann oder ob die Kapazität an die Kapazität der Writer-DB-Instance gebunden ist. Diese Skalierungseigenschaft hängt von der Failover-Priorität der Reader-DB-Instance ab. Wenn die Priorität des Readers 0 oder 1 ist, verfolgt die Kapazität des Readers die Kapazität der Writer-DB-Instance. Damit Reader-DB-Instances in den unterschiedlichsten Situationen automatisch angehalten werden können, setzen Sie ihre Priorität daher auf einen höheren Wert als 1 fest.

Die Fortsetzung einer Reader-Instance kann etwas länger dauern als die Fortsetzung einer Writer-Instance. Stellen Sie eine Verbindung zum Cluster-Endpunkt her, um die schnellste Reaktion zu erhalten, falls Instances möglicherweise angehalten wurden.

Funktionsweise der Auto-Pause-Funktion von Aurora Serverless v2 für Cluster mit bereitgestellten Instances

Alle bereitgestellten DB-Instances in Ihrem Aurora-DB-Cluster werden nicht automatisch angehalten. Nur Aurora Serverless v2-DB-Instances mit der `db.serverless`-Instance-Klasse können die Auto-Pause-Funktion verwenden.

Wenn Ihr Aurora-Cluster bereitgestellte DB-Instances enthält, werden Aurora Serverless v2-Writer-Instances nicht automatisch angehalten. Das liegt an der Anforderung, dass die Writer-Instance verfügbar bleiben muss, solange Reader-Instances aktiv sind. Die Tatsache, dass der Aurora Serverless v2-Writer aktiv bleibt, bedeutet auch, dass alle Aurora Serverless v2-Reader-Instances

mit der Failover-Priorität 0 und 1 in einem Hybrid-Cluster, der bereitgestellte Instances enthält, nicht automatisch angehalten werden.

Überwachung von Aurora-Clustern, die die Auto-Pause-Funktion verwenden

Um Aurora zu überwachen, sollten Sie bereits mit den Überwachungsverfahren in Überwachen von Amazon Aurora-Metriken mit Amazon CloudWatch und den unter Amazon Aurora-Referenz für Metriken aufgeführten CloudWatch-Metriken vertraut sein. Beachten Sie, dass bei der Überwachung von Aurora-Clustern, die die Auto-Pause-Funktion verwenden, einige besondere Überlegungen zu beachten sind:

- Es kann Zeiträume geben, in denen Aurora Serverless v2-Instances Protokolldaten und die meisten Metriken nicht aufzeichnen, weil die Instances angehalten sind. Die einzigen Metriken, die an CloudWatch gesendet werden, während eine Instance angehalten ist, sind 0 Prozent für `CPUUtilization` und `ACUUtilization` und 0 für `ServerlessDatabaseCapacity`.
- Sie können überprüfen, ob Aurora Serverless v2-Instances häufiger oder seltener angehalten werden als erwartet. Prüfen Sie dazu, wie oft sich die `ServerlessDatabaseCapacity`-Metrik von einem Wert ungleich 0 in 0 ändert und wie lange sie Null bleibt. Wenn die Instances nicht so lange angehalten werden, wie Sie es erwarten, sparen Sie nicht so viel an Kosten, wie Sie könnten. Wenn die Instances häufiger als beabsichtigt angehalten und fortgesetzt werden, kann es bei der Reaktion auf Verbindungsanfragen zu unnötiger Latenz in Ihrem Cluster kommen. Informationen zu den Faktoren, die beeinflussen, ob und wie oft Aurora Serverless v2-Instances angehalten werden können, finden Sie unter Voraussetzungen und Einschränkungen für die Auto-Pause-Funktion in Aurora Serverless v2, Situationen, in denen Aurora Serverless v2 nicht automatisch angehalten wird und Fehlerbehebung für die Auto-Pause-Funktion von Aurora Serverless v2.
- Sie können auch eine Protokolldatei untersuchen, in der Vorgänge für das automatische Anhalten und Fortsetzen für eine Aurora Serverless v2-Instance aufgezeichnet werden. Wenn eine Instance nach Ablauf des Timeout-Intervalls nicht angehalten wurde, enthält diese Protokolldatei auch den Grund, warum das automatische Anhalten nicht stattgefunden hat. Weitere Informationen finden Sie unter Überwachen von Aurora Serverless v2-Aktivitäten zum Anhalten und Fortsetzen.

Themen

- Überprüfen, ob eine Aurora Serverless v2-Instance angehalten wurde
- Ereignisse für die Operationen zum automatischen Anhalten und Fortsetzen
- Funktionsweise des automatischen Anhaltens mit Performance Insights und Enhanced Monitoring
- Interaktion der Auto-Pause-Funktion von Aurora Serverless v2 mit Aurora-Metriken

Überprüfen, ob eine Aurora Serverless v2-Instance angehalten wurde

Um festzustellen, ob sich eine Aurora Serverless v2-Instance im angehaltenen Zustand befindet, können Sie die `ACUUtilization`-Metrik für die Instance beobachten. Diese Metrik hat den Wert 0, während die Instance angehalten ist.

Während eine Aurora Serverless v2-Instance angehalten ist, wird ihr Statuswert immer noch als Verfügbar aufgeführt. Dasselbe gilt, wenn eine angehaltene Aurora Serverless v2-Instance gerade fortgesetzt wird. Das liegt daran, dass Sie erfolgreich eine Verbindung zu einer solchen Instance herstellen können, auch wenn es bei der Verbindung zu einer leichten Verzögerung kommt.

Alle Metriken, die sich auf die Verfügbarkeit von Aurora-Instances beziehen, berücksichtigen den Zeitraum, in dem die Instance angehalten ist, als Zeit, in der die Instance verfügbar war.

Ereignisse für die Operationen zum automatischen Anhalten und Fortsetzen

Aurora gibt Ereignisse für Aurora Serverless v2-Instances aus, wenn Operationen zum automatischen Anhalten oder Fortsetzen beginnen, beendet oder abgebrochen werden. Die Ereignisse im Zusammenhang mit der Auto-Pause-Funktion sind `RDS-EVENT-0370` bis `RDS-EVENT-0374`. Weitere Informationen zu diesen Ereignissen finden Sie unter DB-Instance-Ereignisse.

Funktionsweise des automatischen Anhaltens mit Performance Insights und Enhanced Monitoring

Während eine Aurora Serverless v2-Instance angehalten ist, sammelt Aurora keine Monitoring-Informationen für diese Instance, weder über Performance Insights noch über Enhanced Monitoring. Wenn die Instance fortgesetzt wird, kann es zu einer kurzen Verzögerung kommen, bevor Aurora die Erfassung dieser Überwachungsinformationen wieder aufnimmt.

Interaktion der Auto-Pause-Funktion von Aurora Serverless v2 mit Aurora-Metriken

Während eine Aurora Serverless v2-Instance angehalten ist, gibt sie die meisten CloudWatch-Metriken nicht aus und schreibt auch keine Informationen in ihre Datenbankprotokolle. Die einzigen Metriken, die an CloudWatch gesendet werden, während eine Instance angehalten ist, sind 0 Prozent für `CPUUtilization` und `ACUUtilization` und 0 für `ServerlessDatabaseCapacity`.

Wenn CloudWatch Statistiken zur Verfügbarkeit und Betriebszeit von Instances oder Clustern berechnet, gelten Aurora Serverless v2-Instances während der Zeit, in der sie angehalten sind, als verfügbar.

Wenn Sie eine AWS CLI- oder eine RDS-API-Aktion initiieren, um die Protokolle für eine angehaltene Instance zu beschreiben oder herunterzuladen, wird die Aurora Serverless v2-Instance automatisch fortgesetzt, um die Protokollinformationen verfügbar zu machen.

Beispiel für CloudWatch-Metriken

Die folgenden AWS CLI-Beispiele zeigen, wie Sie beobachten können, wie sich die Kapazität einer Instance im Laufe der Zeit ändert. Während dieses Zeitraums wird die Instance automatisch angehalten und dann wieder fortgesetzt. Während der Pause meldet die `ServerlessDatabaseCapacity`-Metrik einen Wert von 0. Um festzustellen, ob die Instance zu irgendeinem Zeitpunkt während des Zeitraums angehalten war, prüfen wir, ob die Mindestkapazität in diesem Zeitraum 0 war.

Das folgende Linux-Beispiel zeigt eine Aurora Serverless v2-Instance, die für einige Zeit automatisch angehalten wurde. Wir testen `ServerlessDatabaseCapacity` jede Minute über einen Zeitraum von drei Minuten. Der ACU-Mindestwert von 0,0 bestätigt, dass die Instance zu einem Zeitpunkt in jeder Minute angehalten wurde.

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '3 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:11:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:12:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
```

Als Nächstes versuchen wir, eine Verbindung zu der angehaltenen Aurora Serverless v2-Instance herzustellen. In diesem Beispiel verwenden wir absichtlich ein falsches Passwort, sodass der Verbindungsversuch nicht erfolgreich ist. Trotz des Fehlers veranlasst der Verbindungsversuch Aurora, die angehaltene Instance fortzusetzen.

```
$ mysql -h my_cluster_endpoint.rds.amazonaws.com -u admin -pwrong-password
```

```
ERROR 1045 (28000): Access denied for user 'admin'@'ip_address' (using password: YES)
```

Das folgende Linux-Beispiel zeigt, dass die angehaltene Instance fortgesetzt wurde, etwa fünf Minuten lang inaktiv blieb und dann wieder in den angehaltenen Zustand zurückkehrte. Die Instance wurde mit einer Kapazität von 2,0 ACUs fortgesetzt. Dann wurde sie leicht hochskaliert, beispielsweise um eine interne Bereinigung durchzuführen. Da innerhalb des fünfminütigen Timeouts keine Benutzerverbindungsversuche eingingen, ging die Instance von 4,0 ACUs direkt in den angehaltenen Zustand über.

```
$ aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '8 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --statistics Minimum \
  --namespace "AWS/RDS" --dimensions Name=DBInstanceIdentifier,Value=my-autopause-
instance \
  --output text | sort -k 3

ServerlessDatabaseCapacity
DATAPOINTS 0.0 2024-08-02T22:13:00+00:00 None
DATAPOINTS 2.0 2024-08-02T22:14:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:15:00+00:00 None
DATAPOINTS 3.0 2024-08-02T22:16:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:17:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:18:00+00:00 None
DATAPOINTS 4.0 2024-08-02T22:19:00+00:00 None
DATAPOINTS 0.0 2024-08-02T22:20:00+00:00 None
```

Wenn der Bericht zeigen soll, wie schnell die Instance hochskaliert wurde, um die Workload zu bewältigen, können wir `Maximum` statt `Minimum` für die Statistik angeben. Für die Kapazitätsplanung und Kostenschätzung können wir einen längeren Zeitraum angeben und die `Average`-Statistik verwenden. Auf diese Weise können wir typische Kapazitätswerte in Zeiten hoher Aktivität, geringer Aktivität und im angehaltenen Zustand ermitteln. Um das Verhalten genau zu den Zeiten des Anhaltens und Fortsetzens zu untersuchen, können wir einen Zeitraum von einer Sekunde angeben und ein kürzeres Zeitintervall untersuchen. Die Zeitstempelwerte in der Ausgabe, z. B. `2024-08-02T22:13:00+00:00`, demonstrieren das Format zur Angabe genauer Parameter für die Optionen `--start-time` und `--end-time`.

Fehlerbehebung für die Auto-Pause-Funktion von Aurora Serverless v2

Wenn Sie feststellen, dass Aurora Serverless v2-Instances nicht so oft angehalten werden wie erwartet, suchen Sie nach den folgenden möglichen Ursachen:

- Vergewissern Sie sich, dass die Aurora-Version, die Sie verwenden, eine Mindestkapazität von 0 ACUs unterstützt. Informationen zu den Kapazitätsbereichen der verschiedenen Aurora-Versionen finden Sie unter Kapazität von Aurora Serverless v2.
- Vergewissern Sie sich, dass der Mindestkapazitätswert für den Cluster auf 0 ACUs festgelegt ist.
- Vergewissern Sie sich, dass die fragliche Instance tatsächlich die Aurora Serverless v2-Instance-Klasse `db.serverless` und keine der bereitgestellten Instance-Klassen verwendet.
- Vergewissern Sie sich, dass der Cluster keine der inkompatiblen Funktionen oder Einstellungen von Voraussetzungen und Einschränkungen für die Auto-Pause-Funktion in Aurora Serverless v2 verwendet.
- Untersuchen Sie die Protokolldatei, aus der hervorgeht, wann Aurora Serverless v2-Instances angehalten oder fortgesetzt wurden oder Aurora eine Instance aus irgendeinem Grund nicht anhalten oder fortsetzen konnte. Weitere Informationen finden Sie unter Überwachen von Aurora Serverless v2-Aktivitäten zum Anhalten und Fortsetzen.
- Überprüfen Sie, ob Clients oder Anwendungen die Verbindungen über einen langen Zeitraum geöffnet halten. Prüfen Sie umgekehrt, ob Anwendungen, die die RDS-Daten-API oder Lambda-Funktionen verwenden, häufig Anfragen senden, sodass die Instance nie lange genug inaktiv ist, um angehalten zu werden. Sie können die CloudWatch-Metriken wie `ConnectionAttempts` und `DatabaseConnections` untersuchen. Weitere Informationen finden Sie unter Metriken auf Instance-Ebene für Amazon Aurora.
- Wenn eine Reader-Instance selten oder nie angehalten wird, überprüfen Sie ihre Failover-Priorität. Wenn der Reader für die Leseskalierung und nicht als Standby im Falle eines Failovers verwendet wird, legen Sie dafür eine Priorität im Bereich von 2 bis 15 fest.
- Wenn die Writer-Instance selten oder nie angehalten wird, überprüfen Sie Ihre Nutzung der Reader-Instances. Der Writer kann nur angehalten werden, wenn der gesamte Cluster inaktiv ist. Das liegt daran, dass eine Verbindung zu einer beliebigen Reader-Instance dazu führt, dass der Writer fortgesetzt wird.
- Wenn Ihre Anwendung bei Verbindungsanfragen Timeouts erhält, während die Aurora Serverless v2-Instances fortgesetzt werden, sollten Sie erwägen, das von Ihrem Anwendungscode oder dem zugrunde liegenden Datenbank-Framework verwendete Timeout-Intervall zu verlängern. Längere Verbindungs-Timeouts verringern die Wahrscheinlichkeit, dass Verbindungen während

der Fortsetzung von Aurora Serverless v2-Instances fehlschlagen. Die längeren Timeouts können jedoch auch dazu führen, dass Ihre Anwendung Verfügbarkeitsprobleme in Ihrem Cluster langsamer erkennt.

Umgekehrt sollten Sie erwägen, das Auto-Pause-Intervall zu verlängern, damit Anwendungen nicht so oft auf angehaltene Instances stoßen.

Wenn für Ihre Anwendung kein logisches Gleichgewicht zwischen dem Auto-Pause-Verhalten und der Reaktionsfähigkeit des Clusters besteht, ist dieser Cluster möglicherweise kein geeigneter Kandidat für die Verwendung der Auto-Pause-Funktion.

- Wenn Sie abschätzen, wie lange Ihre Aurora Serverless v2-Instances angehalten werden, sollten Sie sich darüber im Klaren sein, dass es bestimmte Faktoren gibt, durch die genaue Vorhersagen schwierig sind.
  - Instances können regelmäßig fortgesetzt werden, um Wartungsarbeiten oder Unterversion-Upgrades durchzuführen oder Änderungen an Parametergruppen vorzunehmen.
  - Bei Multi-AZ-Clustern gibt es Situationen, in denen das Fortsetzen einer Instance dazu führt, dass auch andere Instances fortgesetzt werden. Das Fortsetzen eines Readers führt immer dazu, dass der Writer fortgesetzt wird. Wenn der Writer fortgesetzt wird, werden immer alle Reader mit der Failover-Priorität 0 und 1 fortgesetzt.

  Wir empfehlen, die tatsächliche Dauer des angehaltenen Zustands über einen Zeitraum von mehreren Tagen mit einer realistischen Workload zu messen. Verwenden Sie dann diese Messungen, um eine Baseline für den Anteil der Zeit festzulegen, in dem Sie erwarten können, dass eine Instance angehalten ist.
- Möglicherweise stellen Sie fest, dass interne Operationen wie die MySQL-Bereinigung, der MySQL-Ereignisplaner, die PostgreSQL-Selbstbereinigung oder PostgreSQL-Aufträge, die über die `pg_cron`-Erweiterung geplant wurden, nicht ausgeführt oder abgeschlossen werden. Die Instance setzt die Ausführung solcher Operationen, für die keine Benutzerverbindung zur Datenbank erforderlich ist, nicht automatisch fort. Wenn solche internen Operationen nach Ablauf des Auto-Pause-Timeouts ausgeführt werden, werden Wartungsaufgaben wie die MySQL-Bereinigung und PostgreSQL-Selbstbereinigung abgebrochen. Geplante Aufträge aus dem MySQL-Ereignisplaner oder der `pg_cron`-PostgreSQL-Erweiterung werden ebenfalls abgebrochen, falls sie in Bearbeitung sind, wenn Aurora die Operation zum Anhalten initiiert.

  Wenn Sie sicherstellen müssen, dass die Instance regelmäßig aktiv ist, um geplante Operationen auszuführen, können Sie vor der Startzeit des Auftrags eine Verbindung herstellen, um die Instance fortzusetzen. Sie können auch das Timeout-Intervall für die Auto-Pause-Funktion

erhöhen, sodass Operationen wie die Selbstbereinigung länger ausgeführt werden können, nachdem die Benutzeraktivität beendet wurde. Sie können auch Mechanismen wie Lambda-Funktionen verwenden, um Datenbankoperationen nach einem Zeitplan auszuführen, sodass die Instance bei Bedarf automatisch fortgesetzt wird.

Überlegungen zum Anwendungsdesign für die Auto-Pause-Funktion von Aurora Serverless v2

Wenn eine Aurora Serverless v2-DB-Instance fortgesetzt wird, nachdem sie automatisch angehalten wurde, beginnt sie mit einer relativ geringen Kapazität und wird von dort aus hochskaliert. Diese Startkapazität gilt auch dann, wenn die DB-Instance unmittelbar, bevor sie automatisch angehalten wurde, über eine höhere Kapazität verfügte.

Verwenden Sie diese Funktion für Anwendungen, die beim Verbindungsaufbau ein Intervall von etwa 15 Sekunden tolerieren können. Dies entspricht dem typischen Fall, dass eine Aurora Serverless v2-Instance aufgrund einer oder einer geringen Anzahl eingehender Verbindungen fortgesetzt wird. Wenn die Instance länger als 24 Stunden angehalten war, kann das Fortsetzen länger dauern.

Wenn Ihre Anwendung Aurora Serverless v1 und die Funktion zum automatischen Anhalten bereits verwendet hat, haben Sie möglicherweise bereits solche Timeout-Intervalle für Verbindungsversuche eingerichtet. Wenn Sie bereits Aurora Serverless v2 in Kombination mit der Aurora-Stop/Start-Funktion für Cluster verwenden, ist die Zeit für das Fortsetzen automatisch angehaltener Aurora Serverless v2-Instances in der Regel viel kürzer als die Zeit zum Starten eines gestoppten Clusters.

Wenn Sie die Verbindungslogik in Ihrer Anwendung codieren, versuchen Sie, die Verbindung erneut herzustellen, wenn beim ersten Versuch ein Fehler auftritt, der eine vorübergehende Ursache hat. (Falls der Fehler auf einen Authentifizierungsfehler zurückzuführen ist, korrigieren Sie die Anmeldeinformationen, bevor Sie es erneut versuchen.) Ein Fehler, der unmittelbar nach dem Fortsetzen auftritt, kann ein Timeout oder ein Fehler im Zusammenhang mit Datenbanklimits sein. Durch den erneuten Versuch können Probleme in den selteneren Fällen gelöst werden, in denen eine Aurora Serverless v2-Instance aufgrund einer hohen Anzahl gleichzeitiger Verbindungsanfragen fortgesetzt wird. In diesem Fall kann die Verarbeitung einiger Verbindungen länger als gewöhnlich dauern oder beim ersten Versuch die maximale Anzahl gleichzeitiger Verbindungen überschritten werden.

Lassen Sie während der Anwendungsentwicklung und beim Debuggen keine Clientsitzungen oder Programmiertools mit Verbindungen zur Datenbank offen. Aurora hält eine Instance nicht an, wenn vom Benutzer initiierte Verbindungen geöffnet sind, auch wenn die Verbindungen keine SQL-

Anweisungen oder Transaktionen ausführen. Wenn eine Aurora Serverless v2-Instance in einem Aurora-Cluster nicht angehalten werden kann, können auch andere Instances im Cluster daran gehindert werden, angehalten zu werden. Weitere Informationen finden Sie unter Situationen, in denen Aurora Serverless v2 nicht automatisch angehalten wird.

Aurora gibt Ereignisse aus, wenn eine Aurora Serverless v2-DB-Instance mit dem Fortsetzen beginnt, das Fortsetzen beendet wird und wenn die Instance aus irgendeinem Grund nicht fortgesetzt werden kann. Weitere Informationen zu diesen Ereignissen finden Sie unter DB-Instance-Ereignisse.

Migration zu Aurora Serverless v2

Konvertieren Sie einen vorhandenen DB-Cluster zur Verwendung von Aurora Serverless v2 wie folgt:

- Aktualisieren Sie von einem bereitgestellten Aurora-DB-Cluster aus.
- Aktualisieren Sie von einem Aurora Serverless v1-Cluster aus.
- Migration von einer On-Premises-Datenbank zu einem Aurora Serverless v2-Cluster.

Wenn auf Ihrem aktualisierten Cluster die entsprechende Engine-Version ausgeführt wird, wie unter Anforderungen und Einschränkungen für Aurora Serverless v2 aufgeführt, können Sie mit dem Hinzufügen der Aurora Serverless v2-DB-Instances beginnen. Die erste DB-Instance, die Sie dem aktualisierten Cluster hinzufügen, muss eine bereitgestellte DB-Instance sein. Dann können Sie die Verarbeitung für die Schreib-Workload, die Lese-Workload oder beides auf die Aurora Serverless v2DB-Instances umstellen.

Inhalt

- Aktualisieren oder Umstellen vorhandener Cluster auf die Verwendung von Aurora Serverless v2
  - Upgrade-Pfade für MySQL-kompatible Cluster zur Verwendung von Aurora Serverless v2
  - Upgrade-Pfade für PostgreSQL-kompatible Cluster zur Verwendung von Aurora Serverless v2
- Umstellung von einem bereitgestellten Cluster zu Aurora Serverless v2
- Aurora Serverless v2 und Aurora Serverless v1 im Vergleich
  - Aurora Serverless v2- und Aurora Serverless v1-Anforderungen im Vergleich
  - Skalierung und Verfügbarkeit von Aurora Serverless v2 und Aurora Serverless v1 im Vergleich
  - Funktionsunterstützung von Aurora Serverless v2 und Aurora Serverless v1 im Vergleich
  - Anpassen von Aurora Serverless v1-Anwendungsfälle an Aurora Serverless v2
- Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2

- Mit Aurora MySQL kompatible DB-Cluster
- Mit Aurora PostgreSQL kompatible DB-Cluster
- Migrieren einer On-Premises-Datenbank zu Aurora Serverless v2

> **Note**
>
> In diesen Themen wird beschrieben, wie ein vorhandener DB-Cluster konvertiert wird. Weitere Informationen zum Erstellen eines neuen DB-Clusters von Aurora Serverless v2 finden Sie unter Erstellen eines DB-Clusters, der Aurora Serverless v2 verwendet.

Aktualisieren oder Umstellen vorhandener Cluster auf die Verwendung von Aurora Serverless v2

Wenn Ihr bereitgestellter Cluster über eine Engine-Version verfügt, die Aurora Serverless v2 unterstützt, ist für die Umstellung auf Aurora Serverless v2 kein Upgrade erforderlich. In diesem Fall können Sie Ihrem ursprünglichen Aurora Serverless v2-DB-Instances Cluster hinzufügen. Sie können den Cluster umstellen, um alle Aurora Serverless v2-DB-Instances zu verwenden. Sie können auch eine Kombination aus Aurora Serverless v2 und bereitgestellten DB-Instances im selben DB-Cluster verwenden. Die Aurora-Engine-Versionen, die Aurora Serverless v2 unterstützen, finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v2.

Wenn Sie eine niedrigere Engine-Version ausführen, die Aurora Serverless v2 nicht unterstützt, führen Sie diese allgemeinen Schritte aus:

1. Aktualisieren Sie den Cluster.
2. Erstellen Sie eine bereitgestellte Writer-DB-Instance für den aktualisierten Cluster.
3. Ändern Sie den Cluster, sodass er Aurora Serverless v2-DB-Instances verwendet.

> **Important**
>
> Wenn Sie ein größeres Versions-Upgrade auf eine Aurora Serverless v2-kompatible Version unter Verwendung von Snapshot-Wiederherstellung oder -Klonen durchführen, muss die erste DB-Instance, die Sie dem neuen Cluster hinzufügen, eine bereitgestellte DB-Instance sein. Dieser Hinzufügevorgang startet die letzte Phase des Upgrade-Prozesses.

Bis zu dieser letzten Phase verfügt der Cluster nicht über die Infrastruktur, die für die Unterstützung von Aurora Serverless v2 erforderlich ist. Daher beginnen diese aktualisierten Cluster immer mit einer bereitgestellten Writer-DB-Instance. Dann können Sie die bereitgestellte DB-Instance in eine Aurora Serverless v2-Instance konvertieren oder ein entsprechendes Failover durchführen.

Das Aktualisieren von Aurora Serverless v1 auf Aurora Serverless v2 umfasst das Erstellen eines bereitgestellten Clusters als Zwischenschritt. Anschließend führen Sie dieselben Upgrade-Schritte aus wie beim Start mit einem bereitgestellten Cluster.

Upgrade-Pfade für MySQL-kompatible Cluster zur Verwendung von Aurora Serverless v2

Wenn auf Ihrem ursprünglichen Cluster Aurora MySQL ausgeführt wird, wählen Sie je nach Engine-Version und Engine-Modus Ihres Clusters das entsprechende Verfahren aus.

Wenn Ihr ursprünglicher Aurora-MySQL-Cluster dieser ist	Gehen Sie wie folgt vor, um auf Aurora Serverless v2 umzustellen.
Bereitgestellter Cluster mit Aurora MySQL Version 3, kompatibel mit MySQL 8.0	Dies ist die letzte Phase für alle Konvertierungen aus bestehenden Aurora-MySQL-Clustern. Führen Sie ggf. ein Nebenversions-Upgrade auf Version 3.02.0 oder höher durch. Verwenden Sie eine bereitgestellte DB-Instance für die Writer-DB-Instance. Fügen Sie eine Aurora Serverless v2-Reader-DB-Instance hinzu. Führen Sie ein Failover durch, um diese zur Writer-DB-Instance hochzustufen. (Optional) Konvertieren Sie andere bereitgestellte DB-Instances im Cluster in Aurora Serverless v2. Oder fügen Sie neue DB-Instances von Aurora Serverless v2 hinzu und entfernen Sie die bereitgestellten DB-Instances.

Wenn Ihr ursprünglicher Aurora-MySQL-Cluster dieser ist	Gehen Sie wie folgt vor, um auf Aurora Serverless v2 umzustellen.
	Das vollständige Verfahren und Beispiele finden Sie unter Umstellung von einem bereitgestellten Cluster zu Aurora Serverless v2.
Bereitgestellter Cluster mit Aurora MySQL Version 2, kompatibel mit MySQL 5.7	Führen Sie ein Hauptversions-Upgrade auf Aurora MySQL Version 3.02.0 oder höher durch. Befolgen Sie dann die Vorgehensweise für Aurora-MySQL-Version 3, um den Cluster auf die Verwendung von Aurora Serverless v2-DB-Instances umzustellen.
Aurora Serverless v1-Cluster mit Aurora MySQL Version 2, kompatibel mit MySQL 5.7	Hilfestellung zur Planung Ihrer Konvertierung von Aurora Serverless v1 finden Sie unter Aurora Serverless v2 und Aurora Serverless v1 im Vergleich. Folgen Sie dann dem Verfahren unter Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2.

Upgrade-Pfade für PostgreSQL-kompatible Cluster zur Verwendung von Aurora Serverless v2

Wenn auf Ihrem ursprünglichen Cluster Aurora PostgreSQL ausgeführt wird, wählen Sie je nach Engine-Version und Engine-Modus Ihres Clusters das entsprechende Verfahren aus.

Wenn Ihr ursprünglicher Aurora-PostgreSQL-Cluster dieser ist	Gehen Sie wie folgt vor, um auf Aurora Serverless v2 umzustellen.
Bereitgestellter Cluster, auf dem Aurora-PostgreSQL-Version 13 ausgeführt wird	Dies ist die letzte Phase für alle Konvertierungen aus bestehenden Aurora-PostgreSQL-Clustern.

Wenn Ihr ursprünglicher Aurora-PostgreSQL-Cluster dieser ist	Gehen Sie wie folgt vor, um auf Aurora Serverless v2 umzustellen.
	Führen Sie ggf. ein Nebenversions-Upgrade auf Version 13.6 oder höher durch. Fügen Sie eine bereitgestellte DB-Instance für die Writer-DB-Instance hinzu. Fügen Sie eine Aurora Serverless v2-Reader-DB-Instance hinzu. Führen Sie ein Failover aus, damit diese Aurora Serverless v2-Instance zur Writer-DB-Instance wird. (Optional) Konvertieren Sie andere bereitgestellte DB-Instances im Cluster zu Aurora Serverless v2. Oder fügen Sie neue DB-Instances von Aurora Serverless v2 hinzu und entfernen Sie die bereitgestellten DB-Instances. Das vollständige Verfahren und Beispiele finden Sie unter Umstellung von einem bereitgestellten Cluster zu Aurora Serverless v2.
Bereitgestellter Cluster, auf dem Aurora PostgreSQL Version 11 oder 12 ausgeführt wird	Führen Sie ein Hauptversions-Upgrade auf Aurora PostgreSQL Version 13.6 oder höher durch. Befolgen Sie dann die Vorgehensweise für Aurora PostgreSQL Version 13, um den Cluster auf die Verwendung von Aurora Serverless v2-DB-Instances umzustellen.
Aurora Serverless v1-Cluster, auf dem Aurora PostgreSQL Version 11 oder 13 ausgeführt wird	Hilfestellung zur Planung Ihrer Konvertierung von Aurora Serverless v1 finden Sie unter Aurora Serverless v2 und Aurora Serverless v1 im Vergleich. Folgen Sie dann dem Verfahren unter Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2.

Umstellung von einem bereitgestellten Cluster zu Aurora Serverless v2

Stellen Sie einen bereitgestellten Cluster wie folgt auf die Verwendung von Aurora Serverless v2 um:

1. Prüfen Sie, ob der bereitgestellte Cluster aktualisiert werden muss, um mit Aurora Serverless v2-DB-Instances verwendet werden zu können. Die Aurora-Versionen, die mit Aurora Serverless v2 kompatibel sind, finden Sie unter Anforderungen und Einschränkungen für Aurora Serverless v2.

   Wenn auf dem bereitgestellten Cluster eine Engine-Version ausgeführt wird, die für Aurora Serverless v2 nicht verfügbar ist, aktualisieren Sie die Engine-Version des Clusters:

   - Wenn Sie einen mit MySQL 5.7 kompatiblen bereitgestellten Cluster vorliegen haben, befolgen Sie die Upgrade-Anweisungen für Aurora MySQL Version 3. Verwenden Sie die Verfahren in Erläuterung der Durchführung eines direkten Upgrades.
   - Wenn Sie einen mit PostgreSQL kompatiblen bereitgestellten Cluster vorliegen haben, auf dem PostgreSQL Version 11 oder 12 ausgeführt wird, befolgen Sie die Upgrade-Anweisungen für Aurora PostgreSQL Version 13. Verwenden Sie die Verfahren in Durchführen eines Hauptversions-Upgrades.
2. Konfigurieren Sie alle anderen Cluster-Eigenschaften so, dass sie den Aurora Serverless v2-Anforderungen von Anforderungen und Einschränkungen für Aurora Serverless v2 entsprechen.
3. Konfigurieren Sie die Skalierungskonfiguration für den Cluster. Folgen Sie dem Verfahren unter Festlegen des Aurora Serverless v2-Kapazitätsbereichs für einen Cluster.
4. Fügen Sie dem Cluster eine oder mehrere Aurora Serverless v2-DB-Instances hinzu. Gehen Sie vor, wie im allgemeinen Verfahren unter Hinzufügen von Aurora-Replicas zu einem DB-Cluster beschrieben. Geben Sie für jede neue DB-Instance den speziellen DB-Instance-Klassennamen Serverless in der AWS-Managementkonsole oder AWS CLI oder `db.serverless` der Amazon RDS-API an.

   In einigen Fällen sind möglicherweise bereits eine oder mehrere bereitgestellte Reader-DB-Instances im Cluster vorhanden. Wenn dies der Fall ist, können Sie einen der Reader in eine Aurora Serverless v2-DB-Instance konvertieren, anstatt eine neue DB-Instance zu erstellen. Eine Schritt-für-Schritt-Anleitung hierzu finden Sie unter Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2.
5. Führen Sie einen Failover-Vorgang aus, damit eine der Aurora Serverless v2-DB-Instances zur Writer-DB-Instance für den Cluster wird.
6. (Optional) Konvertieren Sie alle bereitgestellten DB-Instances in Aurora Serverless v2 oder entfernen Sie sie aus dem Cluster. Gehen Sie vor, wie im allgemeinen Verfahren unter

Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2 oder Löschen einer DB-Instance aus einem Aurora-DB-Cluster beschrieben.

> **Tip**
>
> Das Entfernen der bereitgestellten DB-Instances ist nicht zwingend erforderlich. Sie können einen Cluster einrichten, der sowohl Aurora Serverless v2 als auch bereitgestellte DB-Instances enthält. Bevor Sie jedoch mit den Leistungs- und Skalierungsmerkmalen von Aurora Serverless v2-DB-Instances vertraut sind, empfehlen wir Ihnen, Ihre Cluster mit DB-Instances desselben Typs zu konfigurieren.

Das folgende AWS CLI Beispiel zeigt den Switchover-Prozess unter Verwendung eines bereitgestellten Clusters, auf dem Aurora MySQL Version 3.02.0 ausgeführt wird. Der Cluster heißt `mysql-80`. Der Cluster beginnt mit zwei bereitgestellten DB-Instances namens `provisioned-instance-1` und `provisioned-instance-2`, ein Writer und ein Reader. Beide verwenden die `db.r6g.large`-DB-Instance-Klasse.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2     False
provisioned-instance-1     True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1     db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2     db.r6g.large
```

Wir erstellen eine Tabelle mit einigen Daten. Auf diese Weise können wir bestätigen, dass die Daten und der Betrieb des Clusters vor und nach der Umstellung gleich sind.

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a
 provisioned writer.');
```

```
Query OK, 1 row affected (0.02 sec)
```

Zunächst fügen wir dem Cluster einen Kapazitätsbereich hinzu. Andernfalls erhalten wir einen Fehler beim Hinzufügen von Aurora Serverless v2-DB-Instances zum Cluster. Wenn wir das AWS-Managementkonsole für dieses Verfahren verwenden, erfolgt dieser Schritt automatisch, wenn wir die erste Aurora Serverless v2 DB-Instance hinzufügen.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance
 operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
 Serverless v2 DB instance.

$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster
 doesn't have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
 'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

Wir erstellen zwei Reader von Aurora Serverless v2, die an die Stelle der ursprünglichen DB-Instances treten. Dazu geben wir die `db.serverless`-DB-Instance-Klasse für die neuen DB-Instances an.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
```

```
  "DBInstanceStatus": "creating"
}

$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
 && \
  aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

Wir führen einen Failover-Vorgang aus, damit eine der Aurora Serverless v2-DB-Instances zur Writer-DB-Instance für den Cluster wird.

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
  --target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
```

```
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

Es dauert einige Sekunden, bis diese Änderung wirksam wird. Zu diesem Zeitpunkt verfügen wir über einen Aurora Serverless v2-Writer und einen Aurora Serverless v2-Reader. Daher benötigen wir keine der ursprünglich bereitgestellten DB-Instances.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
mysql-80
serverless-v2-instance-1        True
serverless-v2-instance-2        False
provisioned-instance-2          False
provisioned-instance-1          False
```

Der letzte Schritt im Umstellungsverfahren besteht darin, beide bereitgestellten DB-Instances zu löschen.

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}

$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-1",
```

```
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

Abschließend bestätigen wir zur Überprüfung, dass die ursprüngliche Tabelle von der Aurora Serverless v2-Writer-DB-Instance aus zugänglich und beschreibbar ist.

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
+--------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless
 v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

Wir stellen außerdem eine Verbindung mit der Aurora Serverless v2-Reader-DB-Instance her und bestätigen, dass die neu geschriebenen Daten dort auch verfügbar sind.

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

Aurora Serverless v2 und Aurora Serverless v1 im Vergleich

Wenn Sie Aurora Serverless v1 bereits verwenden, können Sie sich mit den Hauptunterschieden zwischen Aurora Serverless v1 und Aurora Serverless v2 vertraut machen. Die architektonischen Unterschiede, wie die Unterstützung für Reader-DB-Instances, eröffnen neue Arten von Anwendungsfällen.

Sie können die folgenden Tabellen verwenden, um die wichtigsten Unterschiede zwischen Aurora Serverless v2 und Aurora Serverless v1besser zu verstehen.

Themen

- Aurora Serverless v2- und Aurora Serverless v1-Anforderungen im Vergleich
- Skalierung und Verfügbarkeit von Aurora Serverless v2 und Aurora Serverless v1 im Vergleich
- Funktionsunterstützung von Aurora Serverless v2 und Aurora Serverless v1 im Vergleich
- Anpassen von Aurora Serverless v1-Anwendungsfälle an Aurora Serverless v2

Aurora Serverless v2- und Aurora Serverless v1-Anforderungen im Vergleich

Die folgende Tabelle fasst die verschiedenen Anforderungen zum Ausführen Ihrer Datenbank mit Aurora Serverless v2 oder Aurora Serverless v1 zusammen. Aurora Serverless v2 bietet höhere Versionen der Aurora-MySQL- und Aurora-PostgreSQL-DB-Engines als Aurora Serverless v1.

Funktion	Anforderung von Aurora Serverless v2	Anforderung von Aurora Serverless v1
DB-Engines	Aurora MySQL, Aurora PostgreSQL	Aurora MySQL, Aurora PostgreSQL
Unterstützte Aurora-MySQL-Versionen	Siehe Aurora Serverless v2 mit Aurora MySQL.	Siehe Aurora Serverless v1 mit Aurora MySQL.
Unterstützte Aurora-PostgreSQL-Versionen	Siehe Aurora Serverless v2 mit Aurora PostgreSQL.	Siehe Aurora Serverless v1 mit Aurora PostgreSQL.
Upgraden eines DB-Clusters	Ähnlich wie bei bereitgestellten DB-Clustern können Sie Upgrades manuell durchführ	Nebenversions-Upgrades werden automatisch angewendet, sobald sie

Funktion	Anforderung von Aurora Serverless v2	Anforderung von Aurora Serverless v1
	en, ohne darauf zu warten, dass Aurora den DB-Cluste r für Sie aktualisiert. Weitere Informationen finden Sie unter Ändern eines Amazon Aurora-DB-Clusters. **Note** Um ein Upgrade der Hauptversion von 13.x auf 14.x oder 15.x für einen Aurora PostgreSQ L-kompatiblen DB-Cluster durchzuführen, muss die maximale Kapazität Ihres Clusters mindestens 2 betragen. ACUs	verfügbar sind. Weitere Informationen finden Sie unter Aurora Serverless v1- und Aurora-Datenbank-Engine-Ver sionen. Sie können Hauptversions-Upgrades manuell durchführ en. Weitere Informationen finden Sie unter Ändern eines Aurora Serverless v1-DB-Clu sters.

Funktion	Anforderung von Aurora Serverless v2	Anforderung von Aurora Serverless v1
Konvertieren von bereitgestellten Clustern aus	Sie können die folgenden Methoden verwenden: • Fügen Sie einem vorhandenen bereitgestellten Cluster eine oder mehrere Aurora Serverless v2-Reader DB-Instances hinzu. Wenn Sie Aurora Serverless v2 für den Writer verwenden möchten, führen Sie ein Failover auf eine der Aurora Serverless v2-DB-Instances durch. Damit der gesamte Cluster Aurora Serverless v2 verwendet, entfernen Sie alle bereitgestellten Writer-DB-Instances, nachdem Sie die Aurora Serverless v2-DB-Instance zum Writer hochgestuft haben. • Erstellen Sie ein neues Cluster mit der entsprechenden DB-Engine und Engine-Version. Verwenden Sie eine der Standardmethoden. Stellen Sie beispielsweise einen Cluster-Snapshot wieder her oder erstellen Sie einen Klon eines vorhandenen Clusters. Wählen Sie Aurora Serverless v2 für einige	Stellen Sie den Snapshot des bereitgestellten Clusters wieder her, um einen neuen Aurora Serverless v1-Cluster zu erstellen.

Funktion	Anforderung von Aurora Serverless v2	Anforderung von Aurora Serverless v1
	oder alle DB-Instances im neuen Cluster. Wenn Sie den neuen Cluster durch Klonen erstellen, können Sie die Engine-Version nicht gleichzeitig aktualisieren. Stellen Sie sicher, dass auf dem ursprünglichen Cluster bereits eine Engine-Version ausgeführt wird, die mit Aurora Serverless v2 kompatibel ist.	
Konvertieren von einem Cluster von Aurora Serverless v1 aus	Folgen Sie dem Verfahren unter Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2.	Nicht zutreffend
Verfügbare DB-Instance-Klassen	Die DB-Instance-Sonderklasse `db.serverless` . In der ist es als Serverless AWS-Management konsole gekennzeichnet.	Nicht zutreffend. Aurora Serverless v1 verwendet den `serverless` -Engine-Modus.
Port	Jeder Port, der mit MySQL oder PostgreSQL kompatibel ist	Nur MySQL- oder PostgreSQL-Standard-Port
Öffentliche IP-Adresse zulässig?	Ja	Nein

Funktion	Anforderung von Aurora Serverless v2	Anforderung von Aurora Serverless v1
Virtual Private Cloud (VPC) erforderlich?	Ja	Ja. Jeder Aurora Serverless v1-Cluster verbraucht 2 Schnittstellen- und Gateway-Load-Balancer-Endpunkte, die Ihrer VPC zugewiesen sind.

Skalierung und Verfügbarkeit von Aurora Serverless v2 und Aurora Serverless v1 im Vergleich

In der folgenden Tabelle werden die Unterschiede zwischen Aurora Serverless v2 und Aurora Serverless v1 im Hinblick auf Skalierbarkeit und Verfügbarkeit zusammengefasst.

Die Aurora Serverless v2-Skalierung ist reaktionsschneller, stärker granular und weniger störend als die Skalierung in Aurora Serverless v1.Aurora Serverless v2 kann sowohl durch Ändern der Größe der DB-Instance als auch durch Hinzufügen weiterer DB-Instances zum DB-Cluster skaliert werden. Es kann auch skaliert werden, indem Cluster in anderen AWS-Regionen zu einer globalen Aurora-Datenbank hinzugefügt werden. Im Gegensatz dazu skaliert Aurora Serverless v1 nur durch Erhöhung oder Verringerung der Kapazität des Writers. Sämtliche Berechnungen für einen Aurora Serverless v1-Cluster erfolgen in einer einzigen Availability Zone und einer einzigen AWS-Region.

Skalierung und Funktion für Hochverfügbarkeit	Aurora Serverless v2-Verhalten	Aurora Serverless v1-Verhalten
Minimale Aurora-Kapazitätseinheiten (ACUs) (Aurora MySQL)	0,5 wenn der Cluster ausgeführt wird, 0, wenn der Cluster angehalten ist.	1 wenn der Cluster läuft, 0, wenn der Cluster angehalten wird.
Mindestanzahl ACUs (Aurora PostgreSQL)	0,5 wenn der Cluster ausgeführt wird, 0, wenn der Cluster angehalten ist.	2 wenn der Cluster läuft, 0, wenn der Cluster angehalten wird.
Maximal ACUs (Aurora MySQL)	256	256

Skalierung und Funktion für Hochverfügbarkeit	Aurora Serverless v2-Verhalten	Aurora Serverless v1-Verhalten
Maximal ACUs (Aurora PostgreSQL)	256	384
Stoppen eines Clusters	Sie können den Cluster manuell stoppen und starten, indem Sie die gleiche Cluster-Stopp-und Start-Funktion verwenden wie für bereitgestellte Cluster.	Der Cluster pausiert nach einem Timeout automatisch. Es dauert einige Zeit, bis er wieder verfügbar ist und die Aktivität fortgesetzt wird.
Skalierung von DB-Instances	Skalieren Sie mit einem Mindestabstand von 0,5 nach oben und unten. ACUs	Skalieren Sie nach oben und unten, indem Sie den verdoppeln oder halbieren. ACUs
Anzahl von DB-Instances	Entspricht einem bereitgestellten Cluster: 1 Writer-DB-Instance, bis zu 15 Reader-DB-Instances.	1 DB-Instance, die sowohl Lese- als auch Schreibvorgänge verarbeitet
Ist die Skalierung möglich, während SQL-Anweisungen ausgeführt werden?	Ja. Bei Aurora Serverless v2 entfällt das Warten auf Ruhepunkt.	Nein. Zum Beispiel wartet die Skalierung auf den Abschluss von lang andauernden Transaktionen, temporären Tabellen und Tabellensperren.
Reader-DB-Instances skalieren zusammen mit dem Writer	Optional	Nicht zutreffend
Maximaler Speicher	128 TiB	128 TiB
Puffer-Cache wird beim Skalieren beibehalten	Ja. Die Größe des Puffer-Caches wird dynamisch geändert.	Nein. Der Puffer-Cache wird nach der Skalierung wiederverwendet.

Skalierung und Funktion für Hochverfügbarkeit	Aurora Serverless v2-Verhalten	Aurora Serverless v1-Verhalten
Failover	Ja, genauso wie für bereitgestellte Cluster.	Nur bestmöglich, vorbehaltlich der Verfügbarkeit der Kapazität. Langsamer als in Aurora Serverless v2.
Multi-AZ-Fähigkeit	Ja, genauso wie für bereitgestellte Cluster. Ein Multi-AZ-Cluster benötigt eine Reader-DB-Instance in einer zweiten Availability Zone (AZ). Für einen Multi-AZ-Cluster führt Aurora im Falle eines AZ-Fehlers ein Multi-AZ-Failover durch.	Aurora Serverless v1-Cluster führen alle ihre Berechnungen in einer einzigen AZ aus. Die Wiederherstellung im Falle eines AZ-Ausfalls ist nur bestmöglich und unterliegt der Verfügbarkeit der Kapazität.
Globale Aurora-Datenbanken	Ja	Nein
Skalierung basierend auf Speicherauslastung	Ja	Nein
Skalierung basierend auf CPU-Last	Ja	Ja
Skalierung basierend auf Netzwerkverkehr	Ja, basierend auf Speicher- und CPU-Overhead des Netzwerkverkehrs. Der `max_connections` -Parameter bleibt konstant, um einen Abbruch von Verbindungen beim Herunterskalieren zu vermeiden.	Ja, basierend auf der Anzahl der Verbindungen.
Timeout-Aktion für Skalierungs-Ereignisse	Nein	Ja

Skalierung und Funktion für Hochverfügbarkeit	Aurora Serverless v2-Verhalten	Aurora Serverless v1-Verhalten
Hinzufügen neuer DB-Instances zum Cluster durch AWS Auto Scaling	Nicht zutreffend. Sie können Aurora Serverless v2-Reader DB-Instances in den Hochstufungsstufen 2 bis 15 erstellen und sie auf geringe Kapazität herunterskaliert lassen.	Nein. Reader-DB-Instances sind nicht verfügbar.

Funktionsunterstützung von Aurora Serverless v2 und Aurora Serverless v1 im Vergleich

Die folgende Tabelle enthält eine Zusammenfassung folgender Funktionen:

- Funktionen, die in Aurora Serverless v2, aber nicht in Aurora Serverless v1 verfügbar sind
- Funktionen, die in Aurora Serverless v1 und Aurora Serverless v2 unterschiedlich funktionieren
- Funktionen, die derzeit in Aurora Serverless v2 nicht verfügbar sind

Aurora Serverless v2 enthält viele Funktionen aus bereitgestellten Clustern, die für Aurora Serverless v1 nicht verfügbar sind.

Funktion	Aurora Serverless v2-Support	Aurora Serverless v1-Support
Cluster-Topologie	Aurora Serverless v2 ist eine Eigenschaft einzelner DB-Instances. Ein Cluster kann mehrere Aurora Serverless v2DB-Instances oder eine Kombination aus Aurora Serverless v2 und bereitgestellten DB-Instances enthalten.	Aurora Serverless v1-Cluster verwenden nicht den Begriff der DB-Instances. Sie können die Aurora Serverless v1-Eigenschaft nach dem Erstellen des Clusters nicht mehr ändern.

Funktion	Aurora Serverless v2-Support	Aurora Serverless v1-Support
Konfigurationsparameter	Es können fast alle Parameter geändert werden wie in bereitgestellten Clustern. Details hierzu finden Sie unter Arbeiten mit Parametergruppen für Aurora Serverless v2.	Es kann nur eine Teilmenge von Parametern geändert werden.
Parametergruppen	Cluster-Parametergruppen und DB-Parametergruppen Parameter mit dem Wert `provisioned` im Attribut `SupportedEngineModes` sind verfügbar. Dies sind deutlich mehr Parameter als in Aurora Serverless v1.	Nur Cluster-Parametergruppe. Parameter mit dem Wert `serverless` im Attribut `SupportedEngineModes` sind verfügbar.
Verschlüsselung für Cluster-Volume	Optional	Erforderlich Die Einschränkungen in Einschränkungen von Amazon Aurora-verschlüsselten DB-Clustern gelten für alle Aurora Serverless v1-Cluster.
Regionsübergreifende Snapshots	Ja	Der Snapshot muss mit Ihrem eigenen Schlüssel AWS Key Management Service (AWS KMS) verschlüsselt werden.
Automatisierte Backups werden nach dem Löschen des DB-Clusters beibehalten	Ja	Nein

Funktion	Aurora Serverless v2-Support	Aurora Serverless v1-Support
TLS/SSL	Ja. Die Unterstützung ist die gleiche wie für bereitgestellte Cluster. Weitere Informationen zur Nutzung finden Sie unter Verwenden von TLS/SSL mit Aurora Serverless v2.	Ja. Es gibt einige Unterschiede im Hinblick auf den TLS-Unterstützung für bereitgestellte Cluster. Weitere Informationen zur Nutzung finden Sie unter Verwenden von TLS/SSL mit Aurora Serverless v1.
Klonen	Nur von und zu DB-Engine-Versionen, die mit Aurora Serverless v2 kompatibel sind. Sie können das Klonen nicht verwenden, um ein Upgrade von Aurora Serverless v1 oder einer früheren Version eines bereitgestellten Clusters durchzuführen.	Nur von und zu DB-Engine-Versionen, die mit Aurora Serverless v1 kompatibel sind.
Integration in Amazon S3	Ja	Ja
Integration mit AWS Secrets Manager	Ja	Nein
Exportieren von DB-Cluster-Snapshots zu S3	Ja	Nein
Zuweisen einer IAM-Rolle	Ja	Nein
Logs auf Amazon hochladen CloudWatch	Optional. Sie wählen aus, welche Protokolle aktiviert und in welche Protokolle hochgeladen werden sollen. CloudWatch	Alle Protokolle, die aktiviert sind, werden CloudWatch automatisch hochgeladen.
Daten-API verfügbar	Ja	Ja

Funktion	Aurora Serverless v2-Support	Aurora Serverless v1-Support
Abfrage-Editor verfügbar	Ja	Ja
Performance Insights	Ja	Nein
Amazon-RDS-Proxy verfügbar	Ja	Nein
Babelfish für Aurora PostgreSQL verfügbar	Ja. Unterstützt für Aurora-PostgreSQL-Versionen, die sowohl mit Babelfish als auch Aurora Serverless v2 kompatibel sind.	Nein

Anpassen von Aurora Serverless v1-Anwendungsfälle an Aurora Serverless v2

Abhängig von Ihrem Anwendungsfall für Aurora Serverless v1 können Sie diesen Ansatz wie folgt anpassen, um die Aurora Serverless v2-Funktionen zu nutzen.

Angenommen, Sie haben einen Aurora Serverless v1-Cluster vorliegen, der leicht geladen ist, und Ihre Priorität besteht darin, die kontinuierliche Verfügbarkeit aufrechtzuerhalten und gleichzeitig die Kosten zu minimieren. Mit Aurora Serverless v2 können Sie mit 0,5 eine kleinere Mindesteinstellung für ACU konfigurieren als bei Aurora Serverless v1. Hier beträgt der Mindestwert 1 ACU. Sie können die Verfügbarkeit erhöhen, indem Sie eine Multi-AZ-Konfiguration erstellen, wobei die Reader-DB-Instance ebenfalls mindestens 0,5 ACUs hat.

Angenommen, Sie haben einen Aurora Serverless v1-Cluster vorliegen, den Sie in einem Entwicklungs- und Testszenario verwenden. In diesem Fall haben die Kosten ebenfalls eine hohe Priorität, aber der Cluster muss nicht jederzeit verfügbar sein. Aurora Serverless v2 kann jede Instance automatisch anhalten, wenn sie vollständig im Leerlauf ist. Sie tun dies, indem Sie eine Mindestkapazität von 0 ACUs für den Cluster angeben, wie unter erklärt. Skalierung auf 0 ACUs mit automatischem Anhalten und Fortsetzen für Aurora Serverless v2 Sie können den Cluster auch manuell stoppen, wenn er nicht benötigt wird, und ihn starten, wenn der nächste Test- oder Entwicklungszyklus ansteht.

Angenommen, Sie haben einen Aurora Serverless v1-Cluster mit einer hohen Workload vorliegen. Ein gleichwertiger Cluster mit Aurora Serverless v2 kann mit größerer Granularität skaliert werden. Aurora Serverless v1Skaliert beispielsweise durch Verdoppelung der Kapazität, beispielsweise von

64 auf 128 ACUs. Im Gegensatz dazu kann Ihre Aurora Serverless v2-DB-Instance in Schritten von 0,5 ACUs skaliert werden.

Angenommen, Ihre Workload erfordert eine höhere Gesamtkapazität, als in Aurora Serverless v1 verfügbar ist. Sie können mehrere Aurora Serverless v2-Reader-DB-Instances verwenden, um die leseintensiven Teile der Workload von der Writer-DB-Instance auszulagern. Sie können leseintensive Workloads auch auf mehrere Reader-DB-Instances verteilen.

Für eine schreibintensive Workload können Sie den Cluster mit einer großen bereitgestellten DB-Instance als Writer konfigurieren. Sie können diesen Vorgang zusammen mit einer oder mehreren Reader-DB-Instances von Aurora Serverless v2 ausführen.

Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2

> **Important**
>
> AWS hat den end-of-life Termin bekannt gegeben fürAurora Serverless v1: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Der Prozess zum Aktualisieren eines DB-Clusters von Aurora Serverless v1 auf Aurora Serverless v2 umfasst mehrere Schritte. Das liegt daran, dass Sie nicht direkt von Aurora Serverless v1 in Aurora Serverless v2 konvertieren können. Es gibt immer einen Zwischenschritt, bei dem der Aurora Serverless v1-DB-Cluster in einen bereitgestellten Cluster konvertiert wird.

Mit Aurora MySQL kompatible DB-Cluster

Sie können Ihren Aurora Serverless v1 DB-Cluster in einen bereitgestellten DB-Cluster konvertieren und ihn dann mithilfe einer blue/green Bereitstellung aktualisieren und in einen Aurora Serverless v2 DB-Cluster konvertieren. Wir empfehlen dieses Verfahren für Produktionsumgebungen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

Um ein blue/green Deployment für ein Upgrade eines Aurora Serverless v1 Clusters zu verwenden, auf dem Aurora MySQL Version 2 ausgeführt wird (MySQL 5.7-kompatibel)

1. Konvertieren Sie den DB-Cluster von Aurora Serverless v1 in einen bereitgestellten Cluster von Aurora MySQL Version 2. Folgen Sie dem Verfahren unter Konvertieren von Aurora Serverless v1 in bereitgestellt.
2. Erstellen Sie eine Bereitstellung. blue/green Folgen Sie dem Verfahren unter Eine blue/green Bereitstellung in — Amazon Aurora erstellen.
3. Wählen Sie eine Aurora-MySQL-Version für den grünen Cluster aus, die mit Aurora Serverless v2 kompatibel ist, z. B. 3.04.1.

   Eine Liste kompatibler Versionen finden Sie unter Aurora Serverless v2 mit Aurora MySQL.
4. Ändern Sie die Writer-DB-Instance des grünen Clusters sodass die DB-Instance-Klasse Serverless v2 (db.serverless) verwendet wird.

   Details hierzu finden Sie unter Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2.
5. Wenn Ihr aktualisierter Aurora Serverless v2-DB-Cluster verfügbar ist, wechseln Sie vom blauen Cluster zum grünen Cluster.

Mit Aurora PostgreSQL kompatible DB-Cluster

Sie können Ihren Aurora Serverless v1 DB-Cluster in einen bereitgestellten DB-Cluster konvertieren und ihn dann mithilfe einer blue/green Bereitstellung aktualisieren und in einen Aurora Serverless v2 DB-Cluster konvertieren. Wir empfehlen dieses Verfahren für Produktionsumgebungen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Blue/Green Deployments für Datenbank-Updates.

So verwenden Sie eine blue/green Bereitstellung zum Upgrade eines Aurora Serverless v1 Clusters, auf dem Aurora PostgreSQL Version 11 ausgeführt wird

1. Konvertieren Sie den DB-Cluster von Aurora Serverless v1 in einen bereitgestellten Cluster von Aurora PostgreSQL. Folgen Sie dem Verfahren unter Konvertieren von Aurora Serverless v1 in bereitgestellt.
2. Erstellen Sie eine Bereitstellung blue/green . Folgen Sie dem Verfahren unter Eine blue/green Bereitstellung in — Amazon Aurora erstellen.

3. Wählen Sie eine Aurora-PostgreSQL-Version für den grünen Cluster aus, die mit Aurora Serverless v2 kompatibel ist, z. B. 15.3.

   Eine Liste kompatibler Versionen finden Sie unter Aurora Serverless v2 mit Aurora PostgreSQL.

4. Ändern Sie die Writer-DB-Instance des grünen Clusters sodass die DB-Instance-Klasse Serverless v2 (db.serverless) verwendet wird.

   Details hierzu finden Sie unter Konvertieren eines bereitgestellten Writers oder Readers in Aurora Serverless v2.

5. Wenn Ihr aktualisierter Aurora Serverless v2-DB-Cluster verfügbar ist, wechseln Sie vom blauen Cluster zum grünen Cluster.

Sie können Ihren Aurora Serverless v1-DB-Cluster auch direkt von Aurora PostgreSQL Version 11 auf Version 13 aktualisieren, ihn in einen bereitgestellten DB-Cluster konvertieren und dann den bereitgestellten Cluster in einen Aurora Serverless v2-DB-Cluster konvertieren.

Konvertieren Sie für das Upgrade dann einen Aurora Serverless v1-Cluster, auf dem Aurora PostgreSQL Version 11 ausgeführt wird.

1. Konvertieren Sie den DB-Cluster von Aurora Serverless v1 in einen bereitgestellten Cluster von Aurora PostgreSQL. Folgen Sie dem Verfahren unter Konvertieren von Aurora Serverless v1 in bereitgestellt.
2. Führen Sie ein Upgrade des Aurora Serverless v1-Clusters auf eine Aurora-PostgreSQL-Version 13 durch, die mit Aurora Serverless v2 kompatibel ist, z. B. 13.12. Folgen Sie dem Verfahren unter Durchführen eines Upgrades der Hauptversion.

   Eine Liste kompatibler Versionen finden Sie unter Aurora Serverless v2 mit Aurora PostgreSQL.

3. Fügen Sie dem Cluster eine Aurora Serverless v2-Reader-DB-Instance hinzu. Weitere Informationen finden Sie unter Hinzufügen eines Aurora Serverless v2-Readers.
4. Führen Sie einen Failover zur Aurora Serverless v2-DB-Instance durch:

   a. Wählen Sie die Writer-DB-Instance des DB-Clusters aus.

   b. Wählen Sie für Aktionen die Option Failover aus.

   c. Klicken Sie auf der Bestätigungsseite auf Failover.

Für Aurora Serverless v1-DB-Cluster, auf denen Aurora PostgreSQL Version 13 ausgeführt wird, konvertieren Sie den Aurora Serverless v1-Cluster in einen bereitgestellten DB-Cluster und anschließend den bereitgestellten Cluster in einen Aurora Serverless v2-DB-Cluster.

So führen Sie ein Upgrade eines Clusters von Aurora Serverless v1 durch, auf dem Aurora PostgreSQL Version 13 ausgeführt wird

1. Konvertieren Sie den DB-Cluster von Aurora Serverless v1 in einen bereitgestellten Cluster von Aurora PostgreSQL. Folgen Sie dem Verfahren unter Konvertieren von Aurora Serverless v1 in bereitgestellt.
2. Fügen Sie dem Cluster eine Aurora Serverless v2-Reader-DB-Instance hinzu. Weitere Informationen finden Sie unter Hinzufügen eines Aurora Serverless v2-Readers.
3. Führen Sie einen Failover zur Aurora Serverless v2-DB-Instance durch:
    a. Wählen Sie die Writer-DB-Instance des DB-Clusters aus.
    b. Wählen Sie für Aktionen die Option Failover aus.
    c. Klicken Sie auf der Bestätigungsseite auf Failover.
4. Entfernen Sie die Reader-Instance.

Migrieren einer On-Premises-Datenbank zu Aurora Serverless v2

Sie können Ihre On-Premises-Datenbanken zu Aurora Serverless v2 migrieren, genau wie bei bereitgestellten Aurora MySQL und Aurora PostgreSQL.

- Für MySQL-Datenbanken können Sie den `mysqldump`-Befehl verwenden. Weitere Informationen finden Sie im Abschnitt Importing data to an Amazon RDS für MySQL database with reduced downtime im Benutzerhandbuch für Amazon Relational Database Service.
- Für PostgreSQL-Datenbanken können Sie die Befehle `pg_dump` und `pg_restore` verwenden. Weitere Informationen finden Sie im Blog-Beitrag Best practices for migrating PostgreSQL databases to Amazon RDS and Amazon Aurora (Bewährte Methoden für die Migration von PostgreSQL Datenbanken zu RDS und Amazon Aurora).

Verwenden von Amazon Aurora Serverless v1

> **Important**
>
> AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Amazon Aurora Serverless v1 (Amazon Aurora Serverless Version 1) ist eine bedarfsabhängige Konfiguration der automatischen Skalierung für Amazon Aurora. Ein Aurora Serverless v1-DB-Cluster ist ein DB-Cluster, der die Rechenkapazität abhängig von den Anforderungen Ihrer Anwendung skaliert. Im Gegensatz dazu wird bei von Aurora bereitgestellten DB-Clustern die Kapazität manuell verwaltet. Aurora Serverless v1 bietet eine relativ einfache, kostengünstige Option für selten oder vorübergehend auftretende oder unvorhersehbare Workloads. Dies ist kosteneffektiv, weil es basierend auf den Anforderungen Ihrer Anwendung automatisch gestartet und die Rechenkapazität skaliert wird, sodass sie mit der Nutzung durch Ihre Anwendung übereinstimmt, und heruntergefahren wird, wenn es nicht verwendet wird.

Weitere Informationen zu den Preisen finden Sie unter Serverless-Preise unter MySQL-kompatible Edition oder PostgreSQL-kompatible Edition auf der Seite Amazon Aurora pricing.

Aurora Serverless v1-Cluster verfügen über die gleiche Art von verteiltem und hochverfügbarem Speicher-Volumen mit hoher Kapazität, das von bereitgestellten DB-Clustern verwendet wird.

Bei einem Aurora Serverless v1-Cluster ist das Cluster-Volume immer verschlüsselt. Sie können den Verschlüsselungsschlüssel auswählen, aber Sie können die Verschlüsselung nicht deaktivieren. Das bedeutet, dass Sie für Aurora Serverless v1 die gleichen Vorgänge ausführen können wie für verschlüsselte Snapshots. Weitere Informationen finden Sie unter Aurora Serverless v1 und Snapshots.

Themen

- Verfügbarkeit von Regionen und Versionen für Aurora Serverless v1
- Vorteile von Aurora Serverless v1
- Anwendungsfälle für Aurora Serverless v1
- Einschränkungen von Aurora Serverless v1
- Anforderungen an die Konfiguration von Aurora Serverless v1
- Verwenden von TLS/SSL mit Aurora Serverless v1
- Funktionsweise von Aurora Serverless v1
- Erstellen eines Aurora Serverless v1-DB Clusters
- Wiederherstellen eines Aurora Serverless v1-DB-Clusters
- Ändern eines Aurora Serverless v1-DB-Clusters
- Manuelles Skalieren der Kapazität des Aurora Serverless v1-DB-Clusters
- Anzeigen von Aurora Serverless v1-DB-Clustern
- Löschen eines Aurora Serverless v1-DB-Clusters
- Aurora Serverless v1- und Aurora-Datenbank-Engine-Versionen

Verfügbarkeit von Regionen und Versionen für Aurora Serverless v1

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Aurora-Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur Verfügbarkeit von Versionen und Regionen im Zusammenhang mit Aurora und Aurora Serverless v1 finden Sie unter Aurora Serverless v1.

Vorteile von Aurora Serverless v1

Aurora Serverless v1 bietet folgende Vorteile:

- Einfacher als bereitgestellt – Aurora Serverless v1 eliminiert einen Großteil der Komplexität bei der Verwaltung von DB-Instances und -Kapazität.
- Skalierbar – Aurora Serverless v1 skaliert Rechen- und Speicherkapazität nahtlos nach Bedarf, ohne dass die Client-Verbindungen gestört werden.
- Kosteneffektiv – Wenn Sie Aurora Serverless v1 verwenden, zahlen Sie nur für die Datenbankressourcen, die Sie verbrauchen – auf Sekundenbasis.

- Hochverfügbarer Speicher – Aurora Serverless v1 verwendet dasselbe fehlertolerante verteilte Speichersystem mit sechsfacher Replikation wie Aurora, um vor Datenverlust zu schützen.

Anwendungsfälle für Aurora Serverless v1

Aurora Serverless v1 ist auf die folgenden Anwendungsfälle ausgelegt:

- Selten verwendete Anwendungen – Sie haben eine Anwendung, die mehrmals pro Tag oder Woche nur für ein paar Minuten verwendet wird, wie beispielsweise eine Blog-Website mit geringem Volumen. Mit Aurora Serverless v1 zahlen Sie nur für die Datenbankressourcen, die Sie verbrauchen – auf Sekundenbasis.
- Neue Anwendungen – Sie stellen eine neue Anwendung bereit und sind sich nicht sicher, welche Instance-Größe Sie benötigen. Mit Aurora Serverless v1 können Sie einen Datenbankendpunkt erstellen und es der Datenbank überlassen, automatisch gemäß den Kapazitätsanforderungen Ihrer Anwendung zu skalieren.
- Variable Workloads – Sie führen eine nur wenig benutzte Anwendung aus, mit Spitzen von 30 Minuten bis zu mehreren Stunden ein paarmal pro Tag oder mehrere Male pro Jahr. Beispiele sind Anwendungen für die Personalabteilung oder für die Erstellung von Budgets oder Betriebsberichten. Mit Aurora Serverless v1 müssen Sie nicht mehr Spitzen- oder durchschnittliche Kapazitäten bereitstellen.
- Unvorhersehbare Workloads – Sie führen tägliche Workloads mit plötzlichen und unvorhersehbaren Aktivitätszuwächsen aus. Ein Beispiel dafür ist eine Verkehrs-Website, für die Aktivitätsspitzen entstehen, wenn es zu regnen beginnt. Mit Aurora Serverless v1 wird Ihre Datenbank automatisch auf die Kapazität skaliert, mit der die Spitzenlast Ihrer Anwendung erfüllt werden kann, und wieder herunterskaliert, wenn die Aktivitätsspitze vorbei ist.
- Entwicklungs- und Testdatenbanken – Ihre Entwickler verwenden Datenbanken während der Arbeitszeit, benötigen sie jedoch nachts oder am Wochenende nicht. Mit Aurora Serverless v1 wird Ihre Datenbank automatisch heruntergefahren, wenn sie nicht verwendet wird.
- Anwendungen für mehrere Mandanten – Mit Aurora Serverless v1 müssen Sie die Datenbankkapazität nicht für jede Anwendung in Ihrer Flotte einzeln verwalten. Aurora Serverless v1 verwaltet die individuelle Datenbankkapazität für Sie.

Einschränkungen von Aurora Serverless v1

Die folgenden Einschränkungen gelten für Aurora Serverless v1:

- **Important**

  AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

- Aurora Serverless v1 unterstützt die folgenden -Funktionen nicht:
  - Globale Aurora-Datenbanken
  - Aurora-Replikate
  - AWS Identity and Access Management (IAM) Datenbankauthentifizierung
  - Rückverfolgung in Aurora
  - Datenbankaktivitätsstreams
  - Kerberos-Authentifizierung
  - Performance Insights
  - RDS-Proxy
  - Anzeigen von Protokollen in der AWS-Managementkonsole
- Verbindungen zu einem Aurora Serverless v1-DB-Cluster werden automatisch geschlossen, wenn sie länger als einen Tag offen gehalten werden.
- Alle Aurora Serverless v1-DB-Cluster haben die folgenden Einschränkungen:
  - Sie können keine Aurora Serverless v1-Snapshots in Amazon-S3-Buckets exportieren.
  - Sie können AWS Database Migration Service und die Erfassung von Datenänderungen (CDC) nicht mit Aurora Serverless v1-DB-Clustern verwenden. Nur bereitgestellte Aurora-DB-Cluster unterstützen CDC mit AWS DMS als Quelle.
  - Sie können Daten nicht als Textdateien in Amazon S3 speichern oder Daten aus Textdateien aus S3 in Aurora Serverless v1 laden.
  - Sie können eine IAM-Rolle nicht einem Aurora Serverless v1-DB-Cluster anfügen. Sie können jedoch Daten aus Amazon S3 in Aurora Serverless v1 laden, indem Sie die `aws_s3`-Erweiterung mit der Funktion `aws_s3.table_import_from_s3` und dem Parameter `credentials`

verwenden. Weitere Informationen finden Sie unter Importieren von Amazon S3 in einen Aurora-PostgreSQL-DB-Cluster.

- Wenn Sie den Query Editor verwenden, wird ein Secrets-Manager-Secret für die DB-Anmeldeinformationen für den Zugriff auf die Datenbank erstellt. Wenn Sie die Anmeldeinformationen aus dem Query Editor löschen, wird das zugehörige Secret auch aus Secrets Manager gelöscht. Sie können dieses Secret nach dem Löschen nicht wiederherstellen.

- Aurora-MySQL-basierte DB-Cluster, auf denen Aurora Serverless v1 ausgeführt wird, unterstützen Folgendes nicht:
  - Aufrufen von AWS Lambda-Funktionen innerhalb des Aurora-MySQL-DB-Clusters. AWS Lambda-Funktionen können jedoch Aufrufe ans Aurora Serverless v1-DB-Cluster tätigen.
  - Wiederherstellen eines Snapshots aus einer DB-Instance, die nicht Aurora MySQL oder RDS für MySQL ist
  - Replizieren von Daten mit auf binären Protokollen basierender Replikation. Diese Einschränkung gilt unabhängig davon, ob Ihr Aurora-MySQL-basierter DB-Cluster Aurora Serverless v1 die Quelle oder das Ziel der Replikation ist. Um Daten aus einer MySQL-DB-Instance außerhalb von Aurora, z. B. aus einer in Amazon EC2 ausgeführten Instance, in einen Aurora Serverless v1-DB-Cluster zu replizieren, sollten Sie die Verwendung von AWS Database Migration Service erwägen. Weitere Informationen finden Sie im AWS Database Migration Service-Benutzerhandbuch.
  - Erstellen von Benutzern mit hostbasiertem Zugriff (`'username'@'IP_address'`). Das liegt daran, dass Aurora Serverless v1 eine Router-Flotte zwischen dem Client und dem Datenbankhost für eine nahtlose Skalierung verwendet. Die IP-Adresse, die der Aurora Serverless-DB-Cluster sieht, ist die des Router-Hosts und nicht die Ihres Clients. Weitere Informationen finden Sie unter Aurora Serverless v1-Architektur.

    Verwenden Sie stattdessen den Platzhalter (`'username'@'%'`).

- Aurora-PostgreSQL-basierte DB-Cluster, auf denen Aurora Serverless v1 ausgeführt wird, haben die folgenden Einschränkungen:
  - Aurora-PostgreSQL-Abfrageplan-Verwaltung (`apg_plan_management`-Erweiterung) wird nicht unterstützt.
  - Die in Amazon RDS PostgreSQL und Aurora PostgreSQL verfügbare logische Replikationsfunktion wird nicht unterstützt.
  - Ausgehende Kommunikation, z. B. Kommunikation, die von Amazon RDS für PostgreSQL-Erweiterungen ermöglicht wird, wird nicht unterstützt. Beispielsweise können Sie mit der Erweiterung `postgres_fdw/dblink` nicht auf externe Daten zugreifen. Weitere Informationen

zu RDS PostgreSQL-Erweiterungen finden Sie unter PostgreSQL in Amazon RDS im RDS-Benutzerhandbuch.

- Derzeit werden bestimmte SQL-Abfragen und -Befehle nicht empfohlen. Dazu gehören empfohlene Sperren auf Sitzungsebene, temporäre Beziehungen, asynchrone Benachrichtigungen (`LISTEN`) und Cursors with Hold (`DECLARE` *`name`* `... CURSOR WITH HOLD FOR` *`query`*). `NOTIFY`-Befehle verhindern außerdem eine Skalierung und werden nicht empfohlen.

  Weitere Informationen finden Sie unter Automatische Skalierung für Aurora Serverless v1.

- Sie können das bevorzugte automatisierte Backup-Fenster für einen Aurora Serverless v1-DB-Cluster nicht festlegen.
- Sie können das Wartungsfenster für einen DB-Cluster von Aurora Serverless v1 einstellen. Weitere Informationen finden Sie unter Anpassen des bevorzugten DB-Cluster-Wartungsfensters.

Anforderungen an die Konfiguration von Aurora Serverless v1

Beachten Sie beim Erstellen eines Aurora Serverless v1-DB-Clusters die folgenden Anforderungen:

- Verwenden Sie diese spezifischen Portnummern für jede DB-Engine:
  - Aurora MySQL – `3306`
  - Aurora PostgreSQL – `5432`
- Erstellen Sie Ihr Aurora Serverless v1-DB-Cluster in einer Virtual Private Cloud (VPC), basierend auf dem Amazon-VPC-Service. Wenn Sie einen Aurora Serverless v1-DB-Cluster in Ihrer VPC erstellen, verbrauchen Sie zwei (2) der fünfzig (50) Interface- und Gateway Load Balancer-Endpunkte, die Ihrer VPC zugewiesen sind. Diese Endpunkte werden automatisch für Sie erstellt. Wenn Sie Ihr Kontingent erhöhen möchten, wenden Sie sich an Support. Weitere Informationen finden Sie unter Amazon VPC-Kontingente.
- Sie können einem Aurora Serverless v1-DB-Cluster keine öffentliche IP-Adresse geben. Sie können nur von einer VPC aus auf einen Aurora Serverless v1-DB-Cluster zugreifen.
- Erstellen Sie Subnetze in verschiedenen Availability Zones für die DB-Subnetzgruppe, die Sie für Ihren Aurora Serverless v1-DB-Cluster verwenden. Anders ausgedrückt: In einer Availability Zone kann sich nur ein Subnetz befinden.
- Änderungen an einer Subnetzgruppe, die von einem Aurora Serverless v1-DB-Cluster verwendet wird, werden nicht auf den Cluster angewendet.

- Sie können von Aurora Serverless v1 aus auf einen AWS Lambda-DB-Cluster zugreifen. Hierfür müssen Sie Ihre Lambda-Funktion so konfigurieren, dass sie in derselben VPC ausgeführt wird wie Ihr Aurora Serverless v1-DB-Cluster. Weitere Informationen über das Arbeiten mit AWS Lambda finden Sie unter Konfigurieren einer Lambda-Funktion für den Zugriff auf Ressourcen in einer Amazon VPC im AWS Lambda-Entwicklerhandbuch.

Verwenden von TLS/SSL mit Aurora Serverless v1

Aurora Serverless v1 verwendet standardmäßig das TLS-/SSL-Protokoll (Transport Layer Security/Secure Sockets Layer), um die Kommunikation zwischen Clients und dem Aurora Serverless v1-DB-Cluster zu verschlüsseln. Die TLS-/SSL-Versionen 1.0, 1.1 und 1.2 werden unterstützt. Sie müssen Ihren Aurora Serverless v1-DB-Cluster nicht für die Verwendung von TLS/SSL konfigurieren.

Es gelten jedoch die folgenden Einschränkungen:

- Die Unterstützung von TLS/SSL für Aurora Serverless v1-DB-Cluster ist derzeit in der AWS-Region China (Peking) nicht verfügbar.
- Wenn Sie Datenbankbenutzer für einen Aurora-MySQL-basierten Aurora Serverless v1-DB-Cluster erstellen, verwenden Sie nicht die `REQUIRE`-Klausel für SSL-Berechtigungen. Diese verhindert, dass Benutzer eine Verbindung mit der Aurora-DB-Instance herstellen können.
- Sowohl bei MySQL-Client- als auch PostgreSQL-Client-Dienstprogrammen sind Sitzungsvariablen, die Sie möglicherweise in anderen Umgebungen verwenden, wirkungslos, wenn TLS/SSL zwischen Client und verwendet wir Aurora Serverless v1.
- Beim MySQL-Client müssen Sie beim Verbinden mit dem `VERIFY_IDENTITY`-Modus von TLS/SSL derzeit den MySQL 8.0-kompatiblen `mysql`-Befehl verwenden. Weitere Informationen finden Sie unter Verbinden mit einer DB-Instance, auf der die MySQL-Datenbank-Engine ausgeführt wird.

Abhängig vom Client, den Sie für die Verbindung mit dem Aurora Serverless v1-DB-Cluster verwenden, müssen Sie möglicherweise TLS/SSL nicht angeben, um eine verschlüsselte Verbindung herzustellen. Um beispielsweise den PostgreSQL-Client zum Verbinden mit einem Aurora Serverless v1-DB-Cluster zu verwenden, auf dem die Aurora-PostgreSQL-kompatible Edition ausgeführt wird, verbinden Sie sich wie gewohnt.

```
psql -h endpoint -U user
```

Nachdem Sie Ihr Passwort eingegeben haben, zeigt der PostgreSQL-Client an, dass Sie die Verbindungsdetails sehen, einschließlich der TLS-/SSL-Version und der Verschlüsselung.

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

Important

Aurora Serverless v1 verwendet das Protokoll Transport Layer Security/Secure Sockets Layer (TLS/SSL), um Verbindungen standardmäßig zu verschlüsseln, es sei denn, SSL/TLS ist von der Clientanwendung deaktiviert. Die TLS/SSL-Verbindung endet bei der Router-Flotte. Die Kommunikation zwischen der Router-Flotte und Ihrem Aurora Serverless v1-DB-Cluster erfolgt innerhalb der internen Netzwerkgrenze des Services.
Sie können den Status der Clientverbindung überprüfen, um zu prüfen, ob die Verbindung mit Aurora Serverless v1 TLS/SSL-verschlüsselt ist. Die PostgreSQL-Tabellen `pg_stat_ssl` und `pg_stat_activity` und die `ssl_is_used`-Funktion zeigen den TLS/SSL-Status für die Kommunikation zwischen der Clientanwendung und Aurora Serverless v1 nicht an. In ähnlicher Weise kann der TLS/SSL-Status nicht von der `status` MySQL-Anweisung abgeleitet werden.
Die Aurora-Clusterparameter `force_ssl` für PostgreSQL und `require_secure_transport` für MySQL wurden für Aurora Serverless v1 früher nicht unterstützt. Diese Parameter sind jetzt für Aurora Serverless v1 verfügbar. Rufen Sie die API-Operation DescribeEngineDefaultClusterParameters auf, um eine vollständige Liste der von Aurora Serverless v1 unterstützten Parameter zu erhalten. Weitere Informationen zu Parametergruppen und Aurora Serverless v1 finden Sie unter Parametergruppen für Aurora Serverless v1.

Um den MySQL-Client zum Verbinden mit einem Aurora Serverless v1-DB-Cluster zu verwenden, auf dem die Aurora-MySQL-kompatible Edition ausgeführt wird, geben Sie in Ihrer Anforderung TLS/SSL an. Das folgende Beispiel umfasst den Amazon Root CA 1 Trust Store, der von Amazon Trust Services heruntergeladen wurde und für diese Verbindung erforderlich ist.

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

Geben Sie bei der Aufforderung Ihr Passwort ein. Kurz darauf öffnet sich der MySQL-Monitor. Sie können sich mit dem `status`-Befehl vergewissern, dass die Sitzung verschlüsselt ist.

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:           ***@*******
SSL:                    Cipher in use is ECDHE-RSA-AES256-SHA
...
```

Weitere Informationen zum Verbinden mit der Aurora MySQL-Datenbank über den MySQL-Client finden Sie unter Verbinden mit einer DB-Instance, auf der die MySQL-Datenbank-Engine ausgeführt wird.

Aurora Serverless v1 unterstützt alle für den MySQL-Client (`mysql`) und PostgreSQL-Client (`psql`) verfügbaren TLS-/SSL-Modi, einschließlich der in der folgenden Tabelle aufgeführten.

Beschreibung des TLS-/SSL-Modus	mysql-	psql
Verbinden ohne Verwendung von TLS/SSL	DISABLED	disable
Verbindung zuerst mit TLS/SSL probieren, bei Bedarf aber auf Nicht-SSL zurückgreifen	PREFERRED	bevorzugen (Standard)
Verwendung von TLS/SSL erzwingen	REQUIRED	require
TLS/SSL erzwingen und CA überprüfen	VERIFY_CA	verify-ca
TLS/SSL erzwingen, CA überprüfen und CA-Hostnamen überprüfen	VERIFY_IDENTITY	verify-full

Aurora Serverless v1 nutzt Platzhalter-Zertifikate. Wenn Sie bei Verwendung von TLS/SSL die Option "CA überprüfen" oder "CA und CA-Hostnamen überprüfen" angeben, laden Sie zuerst den Amazon Root CA 1 Trust Store von Amazon Trust Services herunter. Danach können Sie diese PEM-formatierte Datei in Ihrem Client-Befehl identifizieren. Beim PostgreSQL-Client gehen Sie so vor:

Für Linux, macOS oder Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Weitere Informationen zum Arbeiten mit der Aurora PostgreSQL-Datenbank unter Verwendung des Postgres-Clients finden Sie unter Herstellen einer Verbindung zu einer DB-Instance, in der die PostgreSQL-Datenbank-Engine ausgeführt wird.

Weitere Informationen zum Verbinden mit Aurora-DB-Clustern im Allgemeinen finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

Unterstützte Verschlüsselungssammlungen für Verbindungen mit Aurora Serverless v1-DB-Clustern

Durch die Verwendung von konfigurierbaren Chiffrier-Suiten können Sie mehr Kontrolle über die Sicherheit Ihrer Datenbankverbindungen haben. Sie können eine Liste von Verschlüsselungssammlungen angeben, die Sie zum Sichern von TLS/SSL-Verbindungen zu Ihrer Datenbank des Clients zulassen möchten. Mit konfigurierbaren Chiffrier-Suiten können Sie die Verbindungsverschlüsselung steuern, die Ihr Datenbankserver akzeptiert. Dadurch wird die Verwendung von Verschlüsselungsverfahren verhindert, die nicht sicher sind oder nicht mehr verwendet werden.

Aurora Serverless v1-DB-Cluster, die auf Aurora MySQL basieren, unterstützen dieselben Verschlüsselungssammlungen wie von Aurora MySQL bereitgestellte DB-Cluster. Weitere Informationen über diese Verschlüsselungssammlungen finden Sie unter Konfigurieren von Cipher-Suites für Verbindungen mit Aurora-MySQL-DB-Clustern.

Aurora Serverless v1-DB-Cluster, die auf Aurora PostgreSQL basieren, unterstützen keine Chiffrier-Suiten.

Funktionsweise von Aurora Serverless v1

> **Important**
>
> AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

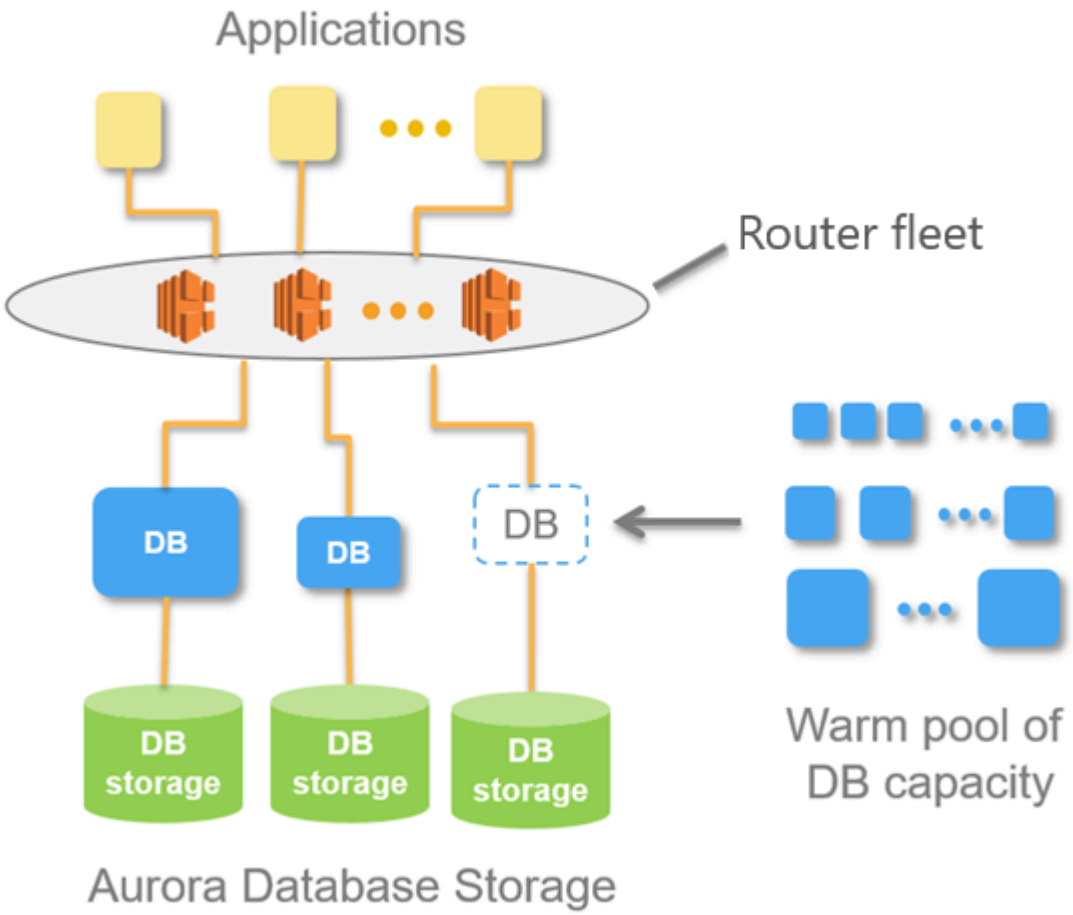
Im Folgenden erfahren Sie, wie Aurora Serverless v1 funktioniert.

Themen

- Aurora Serverless v1-Architektur
- Automatische Skalierung für Aurora Serverless v1
- Timeout-Aktion für Kapazitätsänderungen
- Pausieren und Fortsetzen für Aurora Serverless v1
- Bestimmen der maximalen Anzahl von Datenbankverbindungen für Aurora Serverless v1
- Parametergruppen für Aurora Serverless v1
- Protokollierung für Aurora Serverless v1
- Aurora Serverless v1 und Wartung
- Aurora Serverless v1 und Failover
- Aurora Serverless v1 und Snapshots

Aurora Serverless v1-Architektur

Die folgende Abbildung zeigt einen Überblick über die Aurora Serverless v1-Architektur.



Anstelle der Bereitstellung und Verwaltung von Datenbankservern geben Sie Aurora Capacity Units (ACUs) an. Jede ACU ist eine Kombination aus etwa 2 Gigabyte (GB) Arbeitsspeicher, entsprechender CPU und Netzwerkleistung. Der Datenbankspeicher wird automatisch von 10 Gibibyte (GiB) auf 128 tebibytes (TiB) skaliert, genau wie Speicher in einem standardmäßigen Aurora-DB-Cluster.

Sie können die minimale und maximale ACU angeben. Die Mindestkapazitätseinheit von Aurora ist die kleinste ACU, auf die das DB-Cluster herabskaliert werden kann. Die maximale Kapazitätseinheit von Aurora ist die größte ACU, auf die das DB-Cluster hinaufskaliert werden kann. Basierend auf Ihren Einstellungen erstellt Aurora Serverless v1 automatisch Skalierungsregeln in Bezug auf Schwellenwerte für CPU-Auslastung, Verbindungen und verfügbaren Arbeitsspeicher.

Aurora Serverless v1 verwaltet den warmen Ressourcenpool in einer AWS-Region, um die Skalierungszeit zu minimieren. Wenn Aurora Serverless v1 dem Aurora-DB-Cluster neue Ressourcen hinzufügt, verwendet es die Routerflotte, um aktive Client-Verbindungen auf die neuen Ressourcen umzustellen. Zu jedem Zeitpunkt werden Ihnen nur die ACUs in Rechnung gestellt, die in Ihrem Aurora DB-Cluster aktiv genutzt werden.

Automatische Skalierung für Aurora Serverless v1

Die Ihrem Aurora Serverless v1-DB-Cluster zugewiesene Kapazität wird basierend auf der von Ihrer Client-Anwendung generierten Last nahtlos auf- und abwärts skaliert. Hier ist die Last die CPU-Auslastung und die Anzahl der Verbindungen. Wenn die Kapazität durch eine dieser beiden Faktoren eingeschränkt wird, wird Aurora Serverless v1 skaliert. Aurora Serverless v1 skaliert auch, wenn Leistungsprobleme erkannt werden, die dadurch gelöst werden können.

Sie können Skalierungsereignisse für Ihre Aurora Serverless v1-Cluster in AWS-Managementkonsole einsehen. Während der automatischen Skalierung setzt Aurora Serverless v1 die `EngineUptime`-Metrik zurück. Der Wert des zurückgesetzten Metrikwertes bedeutet weder, dass die nahtlose Skalierung Probleme hatte, noch, dass Aurora Serverless v1 Verbindungen unterbrochen hat. Er ist einfach der Ausgangspunkt für die Betriebszeit bei der neuen Kapazität. Weitere Informationen über Metriken finden Sie unter Überwachung von Metriken in einem Amazon-Aurora-Cluster.

Wenn Ihr Aurora Serverless v1-DB-Cluster keine aktiven Verbindungen hat, kann er auf Kapazität Null skaliert werden (0 ACUs). Weitere Informationen hierzu finden Sie unter Pausieren und Fortsetzen für Aurora Serverless v1.

Wenn er einen Skalierungsvorgang durchführen muss, versucht Aurora Serverless v1 zuerst einen Skalierungspunkt zu identifizieren, ein Moment, in dem keine Anfragen bearbeitet werden. Aurora Serverless v1 kann aus folgenden Gründen möglicherweise keinen Skalierungspunkt finden:

- Lang laufende Anfragen
- Transaktionen in Bearbeitung
- Temporäre Tabellen oder Tabellensperren

Um die Erfolgsrate Ihres Aurora Serverless v1-DB-Clusters bei der Suche nach einem Skalierungspunkt zu erhöhen, empfehlen wir, lang andauernde Abfragen und lang andauernde Transaktionen zu vermeiden. Weitere Informationen zu Operationen zum Blockieren von Skalierungen und wie Sie diese vermeiden können, finden Sie unter Bewährte Methoden für die Arbeit mit Aurora Serverless v1.

Standardmäßig versucht Aurora Serverless v1, einen Skalierungspunkt für 5 Minuten (300 Sekunden) zu finden. Sie können bei der Erstellung oder Änderung des Clusters eine andere Zeitspanne festlegen. Die Timeout-Zeit kann zwischen 60 Sekunden und 10 Minuten (600 Sekunden) liegen. Wenn Aurora Serverless v1 innerhalb des angegebenen Zeitraums keinen Skalierungspunkt finden kann, wird der Autoskalierungsvorgang abgebrochen.

Standardmäßig, wenn automatischen Skalierung vor dem Timeout keinen Skalierungspunkt findet, hält Aurora Serverless v1 den Cluster auf der aktuellen Kapazität. Sie können dieses Standardverhalten bei der Erstellung oder Änderung Ihres Aurora Serverless v1 DB-Clusters ändern, indem Sie die Option Kapazitätsänderung erzwingen auswählen. Weitere Informationen finden Sie unter Timeout-Aktion für Kapazitätsänderungen.

Timeout-Aktion für Kapazitätsänderungen

Wenn die automatische Skalierung abbricht, ohne einen Skalierungspunkt zu finden, behält Aurora standardmäßig die aktuelle Kapazität bei. Sie können festlegen, dass Aurora die Änderung erzwingt, indem Sie die Option Force the capacity change (Kapazitätsänderung erzwingen) aktivieren. Diese Option ist im Abschnitt Autoscaling timeout and action (Autoscaling-Timeout und -Aktion) auf der Seite Create database (Datenbank erstellen) verfügbar, wenn Sie den Cluster erstellen.

Standardmäßig ist die Option Force the capacity change (Kapazitätsänderung erzwingen) deaktiviert. Lassen Sie diese Option deaktiviert, damit die Kapazität Ihres Aurora Serverless v1-DB-Clusters unverändert bleibt, wenn der Skalierungsvorgang ausläuft, ohne einen Skalierungspunkt zu finden.

Wenn Sie diese Option aktivieren, erwzingt Ihr Aurora Serverless v1-DB-Cluster die Kapazitätsänderung auch ohne Skalierungspunkt. Bevor Sie diese Option auswählen, sollten Sie sich der Konsequenzen dieser Auswahl bewusst sein:

- Alle In-Process-Transaktionen werden unterbrochen, und die folgende Fehlermeldung wird angezeigt.

  Aurora MySQL Version 2 – ERROR 1105 (HY000): Die letzte Transaktion wurde aufgrund nahtloser Skalierung abgebrochen. Bitte versuchen Sie es erneut.

  Sie können die Transaktion erneut übermitteln, sobald Ihr Aurora Serverless v1-DB-Cluster verfügbar ist.
- Verbindungen zu temporären Tabellen und Sperren werden gelöscht.

  Wir empfehlen, dass Sie die Option Force the capacity change (Kapazitätsänderung erzwingen) nur dann auswählen, wenn Ihre Anwendung nach unterbrochenen Verbindungen oder unvollständigen Transaktionen wiederhergestellt werden kann.

Die Entscheidungen, die Sie in der AWS-Managementkonsole treffen, wenn Sie einen Aurora Serverless v1-DB-Cluster erstellen, werden im Objekt `ScalingConfigurationInfo` in den Eigenschaften `SecondsBeforeTimeout` und `TimeoutAction` gespeichert. Der Wert der

`TimeoutAction`-Eigenschaft wird bei der Erstellung Ihres Clusters auf einen der folgenden Werte festgelegt:

- `RollbackCapacityChange` – Dieser Wert wird festgelegt, wenn Sie die Option Roll back the capacity change (Rückgängig machen der Kapazitätsänderung) auswählen. Dies ist das Standardverhalten.
- `ForceApplyCapacityChange` – Dieser Wert wird festgelegt, wenn Sie die Option Force the capacity change (Force the capacity change) auswählen.

Sie können den Wert dieser Eigenschaft von einem bestehenden Aurora Serverless v1-DB-Cluster unter Verwendung des AWS CLI-Befehls describe-db-clusters abrufen, wie im Folgenden gezeigt.

Für Linux, macOS oder Unix:

```
aws rds describe-db-clusters --region region \
  --db-cluster-identifier your-cluster-name \
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

Für Windows:

```
aws rds describe-db-clusters --region region ^
  --db-cluster-identifier your-cluster-name ^
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

Das folgende Beispiel zeigt die Abfrage und die Antwort für einen Aurora Serverless v1 DB-Cluster mit der Bezeichnung `west-coast-sles` in der Region US-West (Nordkalifornien).

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-
sles
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'

[
    {
        "ScalingConfigurationInfo": {
            "MinCapacity": 1,
            "MaxCapacity": 64,
            "AutoPause": false,
            "SecondsBeforeTimeout": 300,
            "SecondsUntilAutoPause": 300,
            "TimeoutAction": "RollbackCapacityChange"
```

```
        }
    }
]
```

Wie die Antwort zeigt, verwendet dieser Aurora Serverless v1-DB-Cluster die Standardeinstellung.

Weitere Informationen finden Sie unter Erstellen eines Aurora Serverless v1-DB Clusters. Nach Erstellung Ihres Aurora Serverless v1, können Sie die Timeout-Aktion und andere Kapazitätseinstellungen jederzeit ändern. Um zu erfahren wie, siehe Ändern eines Aurora Serverless v1-DB-Clusters.

Pausieren und Fortsetzen für Aurora Serverless v1

Sie können festlegen, Ihren Aurora Serverless v1-DB-Cluster nach einer bestimmten Zeit ohne Aktivität zu pausieren. Sie geben die Zeitspanne ohne Aktivität an, bevor der DB-Cluster angehalten wird. Wenn Sie diese Option auswählen, beträgt die standardmäßige Inaktivitätszeit fünf Minuten. Sie können diesen Wert jedoch ändern. Dies ist eine optionale Einstellung.

Wenn der DB-Cluster pausiert wird, findet keine Rechen- oder Speicheraktivität statt, und es wird Ihnen nur die Speicherung in Rechnung gestellt. Wenn beim Anhalten eines Aurora Serverless v1 DB-Clusters Datenbankverbindungen angefordert werden, nimmt der DB-Cluster die Verbindungsanforderungen automatisch wieder auf und bedient sie.

Wenn der DB-Cluster die Aktivität fortsetzt, hat er die gleiche Kapazität wie beim Anhalten des Clusters durch Aurora. Die Anzahl der ACUs hängt davon ab, wie stark Aurora den Cluster nach oben oder unten skaliert hat, bevor er angehalten wurde.

> **Note**
>
> Wenn ein DB-Cluster für mehr als sieben Tage pausiert wird, könnte der DB-Cluster mit einem Snapshot gesichert werden. In diesem Fall stellt Aurora den DB-Cluster aus dem Snapshot wieder her, wenn eine Verbindungsanforderung vorliegt.

Bestimmen der maximalen Anzahl von Datenbankverbindungen für Aurora Serverless v1

Im Folgenden sind einige Beispiele für einen DB-Cluster von Aurora Serverless v1 aufgeführt, der mit MySQL 5.7 kompatibel ist. Sie können einen MySQL-Client oder den Abfrage-Editor verwenden,

wenn Sie den Zugriff darauf konfiguriert haben. Weitere Informationen finden Sie unter Ausführen von Abfragen im Abfrage-Editor.

So ermitteln Sie die maximale Anzahl von Datenbankverbindungen

1. Ermitteln Sie den Kapazitätsbereich für Ihren DB-Cluster von Aurora Serverless v1 mithilfe der AWS CLI.

   ```
   aws rds describe-db-clusters \
       --db-cluster-identifier my-serverless-57-cluster \
       --query 'DBClusters[*].ScalingConfigurationInfo|[0]'
   ```

   Das Ergebnis zeigt, dass der Kapazitätsbereich 1 bis 4 ACU beträgt.

   ```
   {
       "MinCapacity": 1,
       "AutoPause": true,
       "MaxCapacity": 4,
       "TimeoutAction": "RollbackCapacityChange",
       "SecondsUntilAutoPause": 3600
   }
   ```

2. Führen Sie die folgende SQL-Abfrage aus, um die maximale Anzahl von Verbindungen zu ermitteln.

   ```
   select @@max_connections;
   ```

   Das angezeigte Ergebnis gilt für die Mindestkapazität des Clusters, 1 ACU.

   ```
   @@max_connections
   90
   ```

3. Skalieren Sie den Cluster auf 8–32 ACU.

   Weitere Informationen zur Skalierung finden Sie unter Ändern eines Aurora Serverless v1-DB-Clusters.

4. Bestätigen Sie den Kapazitätsbereich.

   ```
   {
       "MinCapacity": 8,
   ```

```
    "AutoPause": true,
    "MaxCapacity": 32,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

5. Suchen Sie die maximale Anzahl von Verbindungen.

```
select @@max_connections;
```

Das angezeigte Ergebnis gilt für die Mindestkapazität des Clusters, 8 ACU.

```
@@max_connections
1000
```

6. Skalieren Sie den Cluster auf die maximal mögliche Kapazität, 256–256 ACU.
7. Bestätigen Sie den Kapazitätsbereich.

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

8. Suchen Sie die maximale Anzahl von Verbindungen.

```
select @@max_connections;
```

Das angezeigte Ergebnis gilt für 256 ACU.

```
@@max_connections
6000
```

> **Note**
>
> Der Wert `max_connections` skaliert nicht linear mit der Anzahl der ACU.

9. Skalieren Sie den Cluster wieder herunter auf 1–4 ACU.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

Dieses Mal gilt der Wert `max_connections` für 4 ACU.

```
@@max_connections
270
```

10. Lassen Sie den Cluster auf 2 ACU herunterskalieren.

```
@@max_connections
180
```

Wenn Sie den Cluster so konfiguriert haben, dass er nach einer gewissen Zeit im Leerlauf pausiert, skaliert er auf 0 ACU herunter. Allerdings fällt `max_connections` nicht unter den Wert für 1 ACU.

```
@@max_connections
90
```

Parametergruppen für Aurora Serverless v1

Wenn Sie Ihren Aurora Serverless v1-DB-Cluster erstellen, wählen Sie eine bestimmte Aurora-DB-Engine und eine zugehörige DB-Cluster-Parametergruppe. Im Gegensatz zu bereitgestellten Aurora-DB-Clustern, hat ein Aurora Serverless v1-DB-Cluster eine einzige DB-Instance mit Lese-/Schreibzugriff, die nur mit einer DB-Cluster-Parametergruppe — konfiguriert ist. Er hat keine separate DB-Parametergruppe. Während der automatischen Skalierung, muss Aurora Serverless v1 in der Lage sein, Parameter zu ändern, damit der Cluster optimal für die erhöhte oder verringerte Kapazität funktioniert. Das heißt, dass bei einem Aurora Serverless v1-DB-Cluster, einige der Änderungen, die Sie an Parametern für einen bestimmten DB-Engine-Typ vornehmen, möglicherweise nicht gelten.

Zum Beispiel kann ein Aurora-PostgreSQL–basierter Aurora Serverless v1-DB-Cluster `apg_plan_mgmt.capture_plan_baselines` und andere Parameter nicht verwenden, die auf bereitgestellten Aurora PostgreSQL-DB-Clustern für die Verwaltung von Abfrageplänen verwendet werden könnten.

Sie können eine Liste der Standardwerte für die Standardparametergruppen für die verschiedenen Aurora-DB-Engines unter Verwendung des CLI-Befehls describe-engine-default-cluster-parameters und der Abfrage der AWS-Region erhalten. Die folgenden Werte können Sie für die `--db-parameter-group-family`-Option verwenden.

Aurora-MySQL-Version 2	`aurora-mysql5.7`
Aurora-PostgreSQL-Version 11	`aurora-postgresql11`
Aurora PostgreSQL Version 13	`aurora-postgresql13`

Wir empfehlen Ihnen, Ihre AWS CLI mit Ihrer AWS-Zugriffsschlüssel-ID und Ihrem geheimen AWS-Zugriffsschlüssel zu konfigurieren, und Ihre AWS-Region vor der Verwendung der AWS CLI-Befehle einzustellen. Wenn Sie die Region Ihrer CLI-Konfiguration angeben, müssen Sie die `--region`-Parameter beim Ausführen von Befehlen nicht eingeben. Weitere Informationen über die Konfiguration von AWS CLI finden Sie unter Grundlagen der Konfiguration im AWS Command Line Interface-Benutzerhandbuch.

Im folgenden Beispiel wird eine Liste von Parametern aus der Standard-DB-Cluster-Gruppe für Aurora-MySQL-Version 2 gezeigt.

Für Linux, macOS oder Unix:

```
aws rds describe-engine-default-cluster-parameters \
  --db-parameter-group-family aurora-mysql5.7 --query \
  'EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
  --output text
```

Für Windows:

```
aws rds describe-engine-default-cluster-parameters ^
   --db-parameter-group-family aurora-mysql5.7 --query ^
```

```
   "EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == `true`] | [*].{param:ParameterName}" ^
   --output text
```

Änderung von Parameterwerten für Aurora Serverless v1

Wie in Parametergruppen für Amazon Aurora erläutert, können Sie Werte in einer Standardparametergruppe unabhängig von ihrem Typ (DB-Cluster-Parametergruppe, DB-Parametergruppe) nicht direkt ändern. Stattdessen erstellen Sie eine benutzerdefinierte Parametergruppe basierend auf der standardmäßigen DB-Cluster-Parametergruppe für Ihre Aurora-DB-Engine und ändern die Einstellungen nach Bedarf für diese Parametergruppe. Zum Beispiel möchten Sie vielleicht einige der Einstellungen ändern, damit Ihre Aurora Serverless v1-DB-Cluster Abfragen protokollieren oder DB-Engine-spezifische Protokolle auf Amazon CloudWatch hochladen.

So erstellen Sie eine benutzerdefinierte DB-Cluster-Parametergruppe

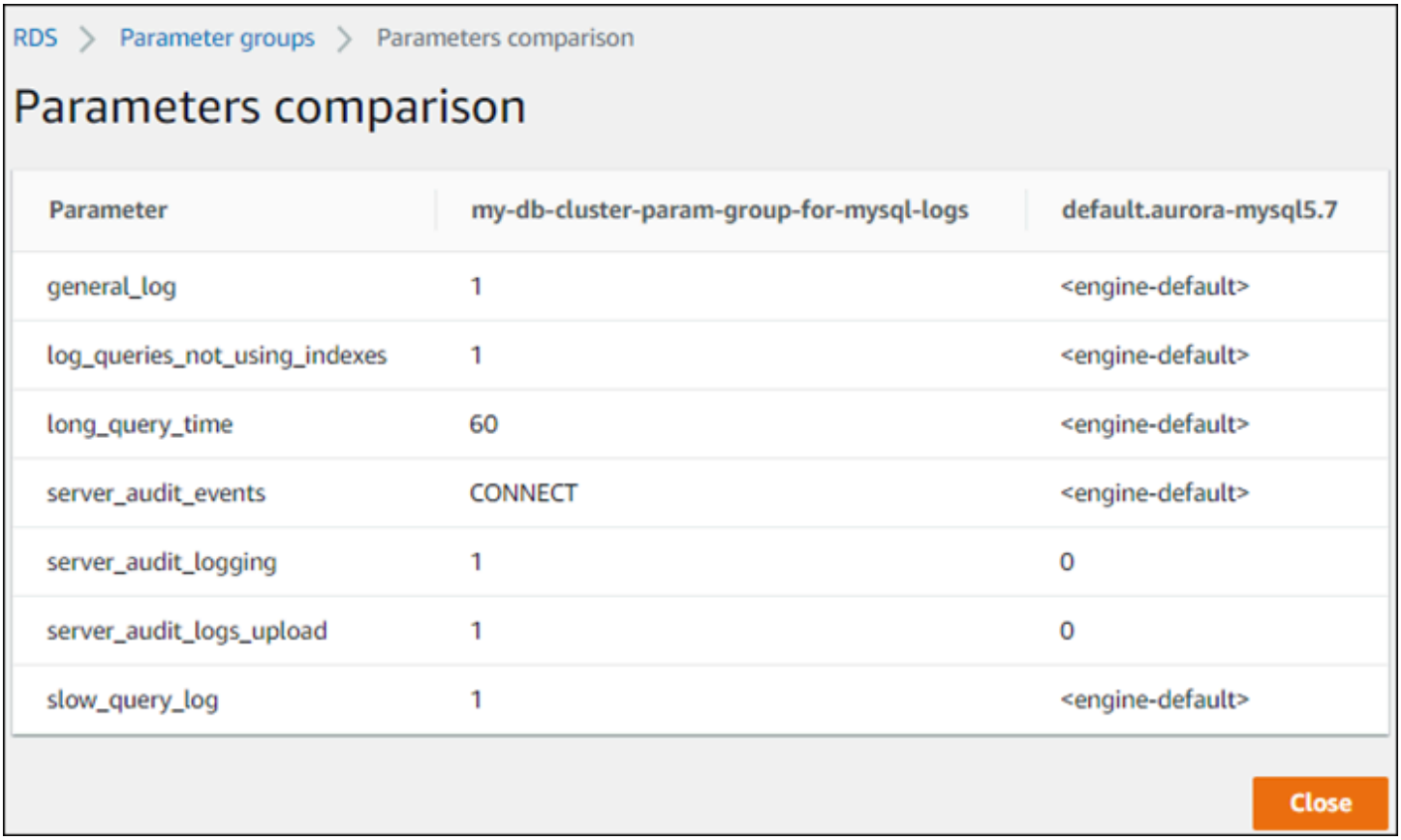
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Parameter groups (Parametergruppen).
3. Wählen Sie Parametergruppe erstellen aus, um den Parametergruppe-Detailbereich zu öffnen.
4. Wählen Sie die entsprechende Standard-DB-Cluster-Gruppe für die DB-Engine aus, die Sie für Ihren Aurora Serverless v1-DB-Cluster verwenden möchten. Wählen Sie dabei unbedingt die folgenden Optionen aus:

    a. Wählen Sie unter Parametergruppenfamilie die entsprechende Familie für die von Ihnen gewählte DB-Engine aus. Achten Sie darauf, dass Ihre Auswahl das Präfix `aurora-` im Namen enthält.
    b. Wählen Sie für Typ die Option DB-Cluster-Parametergruppe.
    c. Geben Sie für Gruppenname und Beschreibung aussagekräftige Namen für Sie oder andere ein, die möglicherweise mit Ihrem Aurora Serverless v1-DB-Cluster und dessen Parametern arbeiten müssen.
    d. Wählen Sie Erstellen aus.

Ihre benutzerdefinierte DB-Cluster-Parametergruppe wird der Liste der Parametergruppen hinzugefügt, die in Ihrer AWS-Region verfügbar sind. Sie können Ihre benutzerdefinierte DB-Cluster-Parametergruppe verwenden, wenn Sie neue Aurora Serverless v1-DB-Cluster erstellen.

Sie können auch einen vorhandenen Aurora Serverless v1-DB-Cluster anpassen, sodass ser Ihre benutzerdefinierte DB-Cluster-Parametergruppe verwendet. Nachdem Ihr Aurora Serverless v1-DB-Cluster mit der Verwendung Ihrer benutzerdefinierten DB-Cluster-Parametergruppe begonnen hat, können Sie die Werte für dynamische Parameter mithilfe von AWS-Managementkonsole oder AWS CLI ändern.

Sie können die Konsole auch verwenden, um einen Vergleich der Werte in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe im Vergleich zur Standardparametergruppe des DB-Clusters anzuzeigen, wie im folgenden Screenshot gezeigt.



Wenn Sie Parameterwerte in einem aktiven DB-Cluster ändern, startet Aurora Serverless v1 eine nahtlose Skalierung, um die Parameteränderungen anzuwenden. Wenn Ihr Aurora Serverless v1-DB-Cluster sich in einem pausierten Zustand befindet, wird er fortgesetzt und beginnt mit der Skalierung, damit er die Änderung vornehmen kann. Der Skalierungsvorgang für eine Parametergruppe ändert immer Erzwingen einer Kapazitätsänderung. Beachten Sie daher, dass das Ändern von Parametern zu unterbrochenen Verbindungen führen kann, wenn während der Skalierungsperiode kein Skalierungspunkt gefunden werden kann.

Protokollierung für Aurora Serverless v1

Standardmäßig werden Fehlerprotokolle für Aurora Serverless v1 aktiviert und automatisch auf Amazon CloudWatch hochgeladen. Sie können Aurora-Datenbank-Engine-spezifische Protokolle von Ihrem Aurora Serverless v1-DB-Cluster auf CloudWatch hochladen lassen. Aktivieren Sie dazu die Konfigurationsparameter in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe. Ihr Aurora Serverless v1-DB-Cluster lädt dann alle verfügbaren Protokolle auf Amazon CloudWatch hoch. An diesem Punkt können Sie mit CloudWatch Logs Analysen der Protokolldaten durchführen, Alarme erstellen und Metriken anzeigen lassen.

Für Aurora MySQL zeigt die folgende Tabelle die Protokolle, die Sie aktivieren können. Wenn sie aktiviert sind, werden sie automatisch von Ihrem Aurora Serverless v1-DB-Cluster zu Amazon CloudWatch hochgeladen.

Aurora-MySQL-Protokoll	Beschreibung
`general_log`	Erstellt das allgemeine Protokoll. Stellen Sie zum Einschalten auf 1. Die Standardeinstellung ist aus (0).
`log_queries_not_using_indexes`	Protokolliert alle Abfragen im Slow-Query-Protokoll, das keinen Index verwendet. Die Standardeinstellung ist aus (0). Stellen Sie auf 1 ein, um dieses Protokoll zu aktivieren.
`long_query_time`	Verhindert, dass schnell ablaufende Abfragen im Protokoll für langsame Abfragen protokoll iert werden. Kann auf einen Gleitkommawert zwischen 0 und 31 536 000 festgelegt werden. Die Standardeinstellung ist 0 (nicht aktiv).
`server_audit_events`	Die Liste der Ereignisse, die in den Protokollen erfasst werden sollen. Unterstützte Werte sind `CONNECT`, `QUERY`, `QUERY_DCL` , `QUERY_DDL` , `QUERY_DML` und `TABLE`.
`server_audit_logging`	Setzen Sie den Parameter auf 1, um die Serverprüfungsprotokollierung zu aktivieren.

Aurora-MySQL-Protokoll	Beschreibung
	Wenn Sie diesen Parameter aktivieren, können Sie angeben, welche Prüfungsereignisse an CloudWatch gesendet werden sollen, indem Sie sie im `server_audit_events` -Parameter auflisten.
`slow_query_log`	Erstellt ein Slow-Query-Protokoll. Auf 1 setzen, um das Slow-Query-Protokoll zu aktivieren. Die Standardeinstellung ist aus (0).

Weitere Informationen finden Sie unter Verwenden von Advanced Auditing in einem Amazon Aurora MySQL DB-Cluster.

Für Aurora PostgreSQL zeigt die folgende Tabelle die Protokolle, die Sie aktivieren können. Wenn sie aktiviert sind, werden sie automatisch von Ihrem Aurora Serverless v1-DB-Cluster mit den regulären Fehlerprotokollen zu Amazon CloudWatch hochgeladen.

Aurora-PostgreSQL-Protokoll	Beschreibung
`log_connections`	Standardmäßig aktiviert und kann nicht geändert werden. Protokolliert Details für alle neuen Client-Verbindungen.
`log_disconnections`	Standardmäßig aktiviert und kann nicht geändert werden. Protokolliert alle Client-Verbindungstrennungen.
`log_hostname`	Standardmäßig deaktiviert und kann nicht geändert werden. Hostnamen werden nicht protokolliert.
`log_lock_waits`	Der Standardwert ist 0 (aus). Stellen Sie auf 1 ein, um die Wartezeiten für die Sperrung zu protokollieren.

Aurora-PostgreSQL-Protokoll	Beschreibung
`log_min_duration_statement`	Die Mindestdauer (in Millisekunden), die eine Anweisung vor der Protokollierung ausgeführt wird.
`log_min_messages`	Legt die Nachrichtenebenen fest, die protokolliert werden. Unterstützte Werte sind , , , , , , , , , , und `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`. Zum Protokollieren von Leistungsdaten im `postgres`-Protokoll, setzen Sie den Wert auf `debug1` .
`log_temp_files`	Protokolliert die Verwendung von temporären Dateien, die über den angegebenen Kilobyte (kB) liegen.
`log_statement`	Steuert die spezifischen SQL-Anweisungen, die protokolliert werden. Unterstützte Werte sind `none`, `ddl`, `mod` und `all`. Der Standardwert ist `none`.

Nachdem Sie die Protokolle für Aurora MySQL oder Aurora PostgreSQL für Ihren Aurora Serverless v1-DB-Cluster aktiviert haben, können Sie die Protokolle in CloudWatch einsehen.

Anzeigen von Aurora Serverless v1-Protokollen mit Amazon CloudWatch

Aurora Serverless v1 lädt automatisch („veröffentlicht“) auf Amazon CloudWatch alle Protokolle hoch, die in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe aktiviert sind. Sie müssen die Protokolltypen nicht auswählen oder angeben. Das Hochladen von Protokollen beginnt, sobald Sie den Protokollkonfigurationsparameter aktivieren. Wenn Sie den Protokoll-Parameter später deaktivieren, werden weitere Uploads angehalten. Alle Protokolle, die bereits in CloudWatch veröffentlicht wurden, sind weiter vorhanden, bis Sie sie löschen.

Weitere Informationen zur Verwendung von CloudWatch mit Aurora MySQL-Protokollen finden Sie unter Überwachung von Protokollereignissen in Amazon CloudWatch.

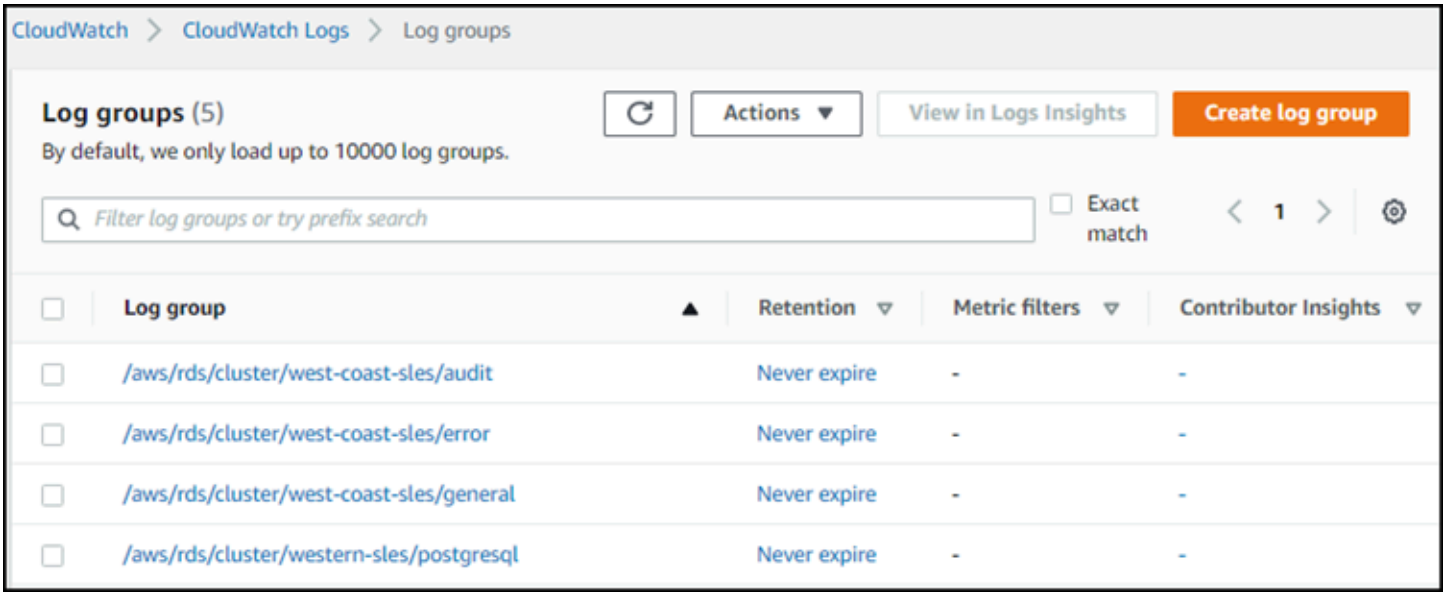
Weitere Informationen zu CloudWatch und Aurora PostgreSQL finden Sie unter Veröffentlichen von Aurora-PostgreSQL-Protokollen in Amazon CloudWatch Logs.

So zeigen Sie Protokolle für Ihren Aurora Serverless v1-DB-Cluster an:

1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie Ihre AWS-Region.
3. Wählen Sie Protokollgruppen.
4. Wählen Sie Ihr DB-Cluster-Protokoll für Aurora Serverless v1 in der Liste aus. Bei Fehlerprotokollen ist das Benennungsmuster wie folgt:

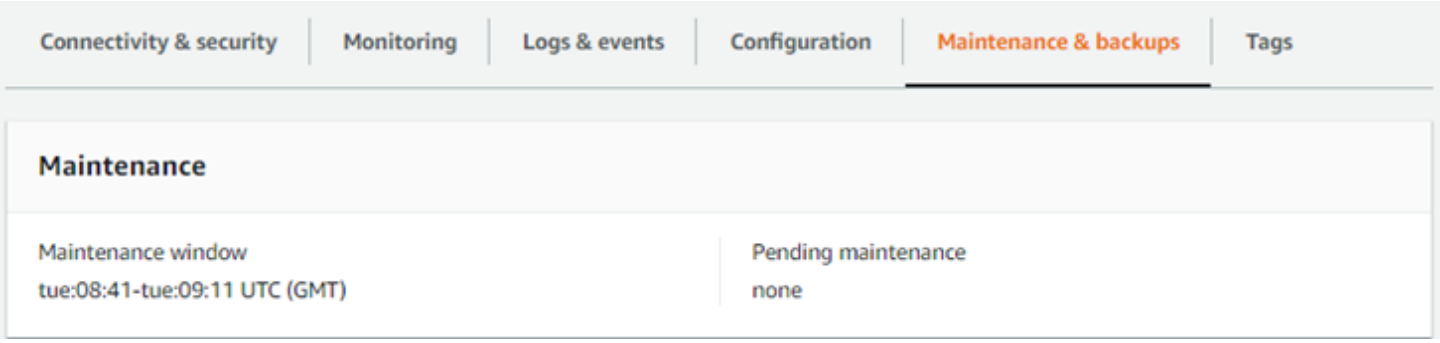
```
/aws/rds/cluster/cluster-name/error
```

Im folgenden Screenshot finden Sie beispielsweise Verzeichnisse für Protokolle, die für einen Aurora PostgreSQL Aurora Serverless v1-DB-Cluster namens `western-sles` veröffentlicht wurden. Sie können auch mehrere Verzeichnisse für Aurora MySQL Aurora Serverless v1-DB-Cluster `west-coast-sles` finden. Wählen Sie das gewünschte Protokoll aus, um sich den Inhalt anzusehen.



Aurora Serverless v1 und Wartung

Die Wartung für Aurora Serverless v1-DB-Cluster, wie die Anwendung der neuesten Funktionen, Bugfixes und Sicherheitsupdates, wird automatisch für Sie durchgeführt. Aurora Serverless v1verfügt über ein Wartungsfenster, das Sie in der AWS-Managementkonsole unter Wartung und Sicherungen) für Ihren Aurora Serverless v1-DB-Cluster anzeigen können. Sie finden dort das Datum und die Uhrzeit für die mögliche Durchführung der Wartung, und ob eine Wartung für Ihren Aurora Serverless v1-DB-Cluster aussteht, wie in der folgenden Abbildung dargestellt.



Sie können das Wartungsfenster festlegen, wenn Sie den Aurora Serverless v1-DB-Cluster erstellen, und Sie können das Fenster später ändern. Weitere Informationen finden Sie unter Anpassen des bevorzugten DB-Cluster-Wartungsfensters.

Wartungsfenster werden für geplante Hauptversions-Upgrades verwendet. Nebenversions-Upgrades und Patches werden sofort während der Skalierung angewendet. Die Skalierung erfolgt entsprechend Ihrer Einstellung für `TimeoutAction`:

- `ForceApplyCapacityChange`: Die Änderung wird sofort angewendet.
- `RollbackCapacityChange`: Aurora erzwingt die Aktualisierung des Clusters 3 Tage nach dem ersten Patchversuch.

Wie bei jeder Änderung, die ohne einen geeigneten Skalierungspunkt erzwungen wird, kann Ihr Workload dadurch unterbrochen werden.

Wann immer möglich, führt Aurora Serverless v1 die Wartung unterbrechungsfrei durch. Wenn eine Wartung erforderlich ist, skaliert Ihr Aurora Serverless v1-DB-Cluster seine Kapazität, damit die erforderlichen Vorgänge durchgeführt werden können. Vor dem Skalieren sucht Aurora Serverless v1 nach einem Skalierungspunkt. Dies geschieht bei Bedarf bis zu drei Tage lang.

Am Ende jedes Tages, an dem Aurora Serverless v1 keinen Skalierungspunkt finden kann, erstellt es ein Cluster-Ereignis. Dieses Ereignis informiert Sie über die ausstehende Wartung und die

Notwendigkeit einer Skalierung für die Durchführung der Wartung. Die Benachrichtigung enthält auch das Datum, an dem Aurora Serverless v1 die Skalierung des DB-Clusters erzwingen kann.

Weitere Informationen finden Sie unter Timeout-Aktion für Kapazitätsänderungen.

Aurora Serverless v1 und Failover

Wenn die DB-Instance für einen Aurora Serverless v1-DB-Cluster nicht mehr verfügbar ist oder die Availability Zone (AZ), in der sie sich befindet, ausfällt, erstellt Aurora die DB-Instance in einer anderen AZ neu. Allerdings ist der Aurora Serverless v1-Cluster kein Multi-AZ-Cluster. Das liegt daran, dass er aus einer einzigen DB-Instance in einer einzigen AZ besteht. Dieser Failover-Mechanismus benötigt mehr Zeit als bei einem Aurora-Cluster mit bereitgestellten oder Aurora Serverless v2-Instances. Die Failover-Zeit für Aurora Serverless v1 ist nicht definiert, da sie von der Nachfrage sowie der Kapazität abhängt, die in anderen AZs innerhalb der angegebenen AWS-Region verfügbar ist.

Da Aurora die Datenverarbeitungskapazität und den Speicher voneinander trennt, ist das Speichervolumen für den Cluster auf mehrere AZs verteilt. Ihre Daten bleiben auch bei Ausfällen der DB-Instance oder der zugehörigen AZ verfügbar.

Aurora Serverless v1 und Snapshots

Das Cluster-Volume für einen Aurora Serverless v1-Cluster ist immer verschlüsselt. Sie können den Verschlüsselungsschlüssel auswählen, aber Sie können die Verschlüsselung nicht deaktivieren. Zum Kopieren oder Freigeben eines Snapshots eines Aurora Serverless v1-Clusters verschlüsseln Sie den Snapshot mit Ihrem eigenen AWS KMS key. Weitere Informationen finden Sie unter Kopieren eines DB-Cluster-Snapshots. Weitere Informationen zur Verschlüsselung und zu Amazon Aurora finden Sie unter Verschlüsseln eines Amazon-Aurora-DB-Clusters.

Erstellen eines Aurora Serverless v1-DB Clusters

Important

AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei

Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Mit dem folgenden Verfahren wird ein Aurora Serverless v1-Cluster ohne eines Ihrer Schemaobjekte oder Daten erstellt. Wenn Sie einen Aurora Serverless v1-Cluster erstellen möchten, der ein Duplikat eines vorhandenen bereitgestellten oder Aurora Serverless v1-Clusters ist, können Sie stattdessen eine Snapshot-Wiederherstellung oder einen Klonvorgang durchführen. Entsprechende Details finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot und Klonen eines Volumes für einen Amazon-Aurora-DB-Cluster. Sie können einen vorhandenen bereitgestellten Cluster nicht in Aurora Serverless v1 konvertieren. Sie können einen vorhandenen Aurora Serverless v1-Cluster auch nicht wieder in einen bereitgestellten Cluster konvertieren.

Wenn Sie einen Aurora Serverless v1-DB-Cluster erstellen, können Sie die Mindest- und Höchstkapazität für den Cluster festlegen. Eine Kapazitätseinheit entspricht einer bestimmten Rechen- und Arbeitsspeicherkonfiguration. Aurora Serverless v1 erstellt Skalierungsregeln in Bezug auf Schwellenwerte für die CPU-Auslastung, Verbindungen und den verfügbaren Arbeitsspeicher. Außerdem erfolgt eine nahtlose Skalierung auf einen Bereich von Kapazitätseinheiten, wie für Ihre Anwendungen erforderlich. Weitere Informationen finden Sie unter Aurora Serverless v1-Architektur.

Sie können die folgenden spezifischen Werte für Ihren Aurora Serverless v1-DB-Cluster festlegen:

- Minimale Aurora Capacity Unit – Aurora Serverless v1 kann die Kapazität bis zu dieser Kapazitätseinheit reduzieren.
- Maximale Aurora Capacity Unit – Aurora Serverless v1 kann die Kapazität bis zu dieser Kapazitätseinheit erhöhen.

Sie können auch die folgenden optionalen Skalierungskonfigurationsoptionen auswählen:

- Bei Zeitüberschreitung Skalierung der Kapazität auf die angegebenen Werte durchsetzen – Sie können diese Einstellung auswählen, wenn Sie möchten, dass Aurora Serverless v1 die Skalierung für Aurora Serverless v1 durchsetzt, auch wenn kein Skalierungspunkt gefunden wird, bevor die Zeitüberschreitung eintritt. Wenn Sie möchten, dass Aurora Serverless v1 Kapazitätsänderungen abbricht, wenn es keinen Skalierungspunkt findet, wählen Sie diese Einstellung nicht. Weitere Informationen finden Sie unter Timeout-Aktion für Kapazitätsänderungen.
- Rechenkapazität nach aufeinanderfolgenden Minuten von Inaktivität anhalten – Sie können diese Einstellung auswählen, wenn Sie möchten, dass Aurora Serverless v1 auf Null skaliert wird, wenn

es für einen von Ihnen angegebenen Zeitraum keine Aktivität auf dem DB-Cluster gibt. Wenn diese Einstellung aktiviert ist, setzt der Aurora Serverless v1-DB-Cluster die Verarbeitung automatisch fort und wird auf die notwendige Kapazität skaliert, um den Workload zu verarbeiten, wenn der Datenbankdatenverkehr fortgesetzt wird. Weitere Informationen hierzu finden Sie unter Pausieren und Fortsetzen für Aurora Serverless v1.

Bevor Sie einen Aurora Serverless v1-DB-Cluster erstellen können, benötigen Sie ein AWS-Konto. Sie müssen darüber hinaus die Einrichtungsaufgaben für die Arbeit mit Amazon Aurora abgeschlossen haben. Weitere Informationen finden Sie unter Einrichten Ihrer Umgebung für Amazon Aurora. Sie müssen auch andere vorab erforderliche Schritte zum Erstellen eines Aurora-DB-Clusters ausführen. Weitere Informationen hierzu finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Aurora Serverless v1 ist in bestimmten AWS-Regionen und nur für bestimmte Aurora-MySQL- und Aurora-PostgreSQL-Versionen verfügbar. Weitere Informationen finden Sie unter Aurora Serverless v1.

> **Note**
>
> Das Cluster-Volume für einen Aurora Serverless v1-Cluster ist immer verschlüsselt. Wenn Sie Ihren Aurora Serverless v1-DB-Cluster erstellen, können Sie die Verschlüsselung nicht deaktivieren, aber Sie können Ihren eigenen Verschlüsselungsschlüssel verwenden. Bei Aurora Serverless v2 können Sie auswählen, ob das Cluster-Volume verschlüsselt werden soll.

Sie können einen Aurora Serverless v1-DB-Cluster über die AWS CLI oder die RDS-API erstellen.

> **Note**
>
> Wenn beim Versuch, den Cluster zu erstellen, die folgende Fehlermeldung angezeigt wird, benötigt Ihr Konto zusätzliche Berechtigungen.
> `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.`
> Weitere Informationen finden Sie unter Verwenden von serviceverknüpften Rollen für Amazon Aurora.

Sie können keine direkte Verbindung zur DB-Instance in Ihrem Aurora Serverless v1-DB-Cluster herstellen. Um eine Verbindung zum Aurora Serverless v1-DB-Cluster herzustellen, verwenden Sie den Datenbankendpunkt. Sie finden den Endpunkt für Ihren Aurora Serverless v1-DB-Cluster auf der Registerkarte Connectivity & security (Konnektivität und Sicherheit) für Ihren Cluster in der AWS-Managementkonsole. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

AWS CLI

Um einen neuen Aurora Serverless v1-DB-Cluster über die AWS CLI zu erstellen, führen Sie den Befehl create-db-cluster aus und geben `serverless` für die Option `--engine-mode` an.

Optional können Sie die Option `--scaling-configuration` angeben, um die minimale Kapazität, die maximale Kapazität und die automatische Pause zu konfigurieren, wenn es keine Verbindungen gibt.

Die folgenden Befehlsbeispiele erstellen einen neuen Serverless DB-Cluster, indem Sie die `--engine-mode`-Option auf `serverless` festlegen. Die Beispiele definieren auch Werte für die `--scaling-configuration`-Option.

Beispiel für Aurora MySQL

Mit den folgenden Befehlen werden neue Aurora MySQL-kompatible Serverless-DB-Cluster erstellt. Gültige Kapazitätswerte für Aurora MySQL sind 1, 2, 4, 8, 16, 32, 64, 128 und 256.

Linux, macOS oder Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Für Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
```

```
    --master-username username --master-user-password password
```

Beispiel für Aurora PostgreSQL

Mit dem folgenden Befehl wird ein neuer, mit PostgreSQL 13.9 kompatibler Serverless-DB-Cluster erstellt. Gültige Kapazitätswerte für Aurora PostgreSQL sind 2, 4, 8, 16, 32, 64, 192 und 384.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-postgresql --engine-version 13.9 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Für Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql --engine-version 13.9 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

RDS-API

Zum Erstellen eines neuen Aurora Serverless v1-DB-Clusters über die RDS-API führen Sie die CreateDBCluster-Operation durch und geben `serverless` als `EngineMode`-Parameter an.

Optional können Sie den Parameter `ScalingConfiguration` angeben, um die minimale Kapazität, die maximale Kapazität und die automatische Pause zu konfigurieren, wenn es keine Verbindungen gibt. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128 und 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 und 384.

Wiederherstellen eines Aurora Serverless v1-DB-Clusters

Important

AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Sie können einen Aurora Serverless v1-DB-Cluster konfigurieren, wenn Sie einen bereitgestellten DB-Cluster-Snapshot über die AWS CLI oder die RDS-API wiederherstellen.

Wenn Sie einen Snapshot in einem Aurora Serverless v1-DB-Cluster wiederherstellen, können Sie die folgenden Werte festlegen:

- Minimale Aurora Capacity Unit – Aurora Serverless v1 kann die Kapazität bis zu dieser Kapazitätseinheit reduzieren.
- Maximale Aurora Capacity Unit – Aurora Serverless v1 kann die Kapazität bis zu dieser Kapazitätseinheit erhöhen.
- Zeitüberschreitungsaktion – Die Aktion, die ausgeführt werden soll, wenn für eine Kapazitätsänderung eine Zeitüberschreitung eintritt, da kein Skalierungspunkt gefunden werden kann.Aurora Serverless v1 Der -DB-Cluster kann die neuen Kapazitätseinstellungen für Ihren DB-Cluster durchsetzen, wenn Sie die Option Bei Zeitüberschreitung Skalierung der Kapazität auf die angegebenen Werte durchsetzen auswählen. Er kann auch einen Rollback für die Kapazitätsänderung ausführen, um sie zu stornieren, wenn Sie die Option nicht auswählen. Weitere Informationen finden Sie unter Timeout-Aktion für Kapazitätsänderungen.
- Pause after inactivity (Nach Inaktivität pausieren): Die Zeitdauer, die ohne Datenbankverkehr verstreichen muss, bis auf eine Verarbeitungskapazität von null skaliert wird. Wenn der Datenbankverkehr wieder aufgenommen wird, nimmt Aurora automatisch die Verarbeitungskapazität wieder auf und skaliert sie in Übereinstimmung mit dem Datenverkehr.

Allgemeine Informationen zum Wiederherstellen eines DB-Clusters aus einem Snapshot finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.

AWS CLI

Sie können einen Aurora Serverless-DB-Cluster konfigurieren, wenn Sie einen bereitgestellten DB-Cluster-Snapshot über die AWS-Managementkonsole, die AWS CLI oder die RDS-API wiederherstellen.

Wenn Sie einen Snapshot in einem Aurora Serverless-DB-Cluster wiederherstellen, können Sie die folgenden Werte festlegen:

- Minimale Aurora Capacity Unit – Aurora Serverless kann die Kapazität bis zu dieser Kapazitätseinheit reduzieren.
- Maximale Aurora Capacity Unit – Aurora Serverless kann die Kapazität bis zu dieser Kapazitätseinheit erhöhen.
- Zeitüberschreitungsaktion – Die Aktion, die ausgeführt werden soll, wenn für eine Kapazitätsänderung eine Zeitüberschreitung eintritt, da kein Skalierungspunkt gefunden werden kann.Aurora Serverless v1 Der -DB-Cluster kann die neuen Kapazitätseinstellungen für Ihren DB-Cluster durchsetzen, wenn Sie die Option Bei Zeitüberschreitung Skalierung der Kapazität auf die angegebenen Werte durchsetzen auswählen. Er kann auch einen Rollback für die Kapazitätsänderung ausführen, um sie zu stornieren, wenn Sie die Option nicht auswählen. Weitere Informationen finden Sie unter Timeout-Aktion für Kapazitätsänderungen.
- Pause after inactivity (Nach Inaktivität pausieren): Die Zeitdauer, die ohne Datenbankverkehr verstreichen muss, bis auf eine Verarbeitungskapazität von null skaliert wird. Wenn der Datenbankverkehr wieder aufgenommen wird, nimmt Aurora automatisch die Verarbeitungskapazität wieder auf und skaliert sie in Übereinstimmung mit dem Datenverkehr.

> **Note**
>
> Die Version des DB-Cluster-Snapshots muss mit Aurora Serverless v1 kompatibel sein. Eine Liste der unterstützten Versionen finden Sie unter Aurora Serverless v1.

Zum Wiederherstellen eines Snapshots in einem Aurora Serverless v1-Cluster mit MySQL 5.7-Kompatibilität fügen Sie die folgenden Parameter hinzu:

- `--engine` **`aurora-mysql`**

- `--engine-version 5.7`

Mit den Parametern `--engine` und `--engine-version` können Sie einen MySQL 5.7-kompatiblen Aurora Serverless v1-Cluster aus einem MySQL 5.6-kompatiblen Aurora- oder Aurora Serverless v1-Snapshot erstellen. Im folgenden Beispiel wird ein Snapshot aus einem MySQL 5.6-kompatiblen Cluster namens *mydbclustersnapshot* in einem MySQL 5.7-kompatiblen Aurora Serverless v1-Cluster namens *mynewdbcluster*wiederhergestellt.

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless \
    --engine aurora-mysql \
    --engine-version 5.7
```

Für Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql ^
    --engine-version 5.7
```

Optional können Sie die Option `--scaling-configuration` angeben, um die minimale Kapazität, die maximale Kapazität und die automatische Pause zu konfigurieren, wenn es keine Verbindungen gibt. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128 und 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 und 384.

Im folgenden Beispiel können Sie von einem zuvor erstellten DB-Cluster-Snapshot namens *mydbclustersnapshot* in einem neuen DB-Cluster namens *mynewdbcluster*wiederherstellen. Sie legen `--scaling-configuration` so fest, dass der neue Aurora Serverless v1-DB-Cluster bei Bedarf von 8 ACUs auf 64 ACUs (Aurora-Kapazitätseinheiten) skaliert werden kann, um die Workload zu verarbeiten. Nach Abschluss der Verarbeitung und nach 1000 Sekunden ohne zu unterstützende

Verbindungen wird der Cluster heruntergefahren, bis die Verbindungsanforderung zum Neustart auffordert.

Für Linux, macOS oder Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

Für Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine-mode serverless --scaling-configuration
 MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

RDS-API

Um einen Aurora Serverless v1-DB-Cluster zu konfigurieren, wenn Sie eine Wiederherstellung von einem DB-Cluster über die RDS-API durchführen, führen Sie die Operation RestoreDBClusterFromSnapshot durch und geben `serverless` als `EngineMode`-Parameter an.

Optional können Sie den Parameter `ScalingConfiguration` angeben, um die minimale Kapazität, die maximale Kapazität und die automatische Pause zu konfigurieren, wenn es keine Verbindungen gibt. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128 und 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 und 384.

Ändern eines Aurora Serverless v1-DB-Clusters

> **Important**
>
> AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das

Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Nachdem Sie einen DB-Cluster von Aurora Serverless v1 konfiguriert haben, können Sie bestimmte Eigenschaften über die AWS-Managementkonsole, die AWS CLI oder die RDS-API ändern. Die meisten Eigenschaften, die Sie ändern können, sind die gleichen wie für andere Arten von Aurora-Clustern.

Die für Aurora Serverless v1 wichtigsten Änderungen sind folgende:

- Ändern der Skalierungskonfiguration
- Durchführen eines Upgrades der Hauptversion
- Konvertieren von Aurora Serverless v1 in bereitgestellt

Ändern der Skalierungskonfiguration eines DB-Clusters von Aurora Serverless v1

Sie können die minimale und maximale Kapazität für den DB-Cluster festlegen. Jede Kapazitätseinheit entspricht einer bestimmten Rechen- und Arbeitsspeicherkonfiguration. Aurora Serverless erstellt automatisch Skalierungsregeln in Bezug auf Schwellenwerte für die CPU-Auslastung, Verbindungen und den verfügbaren Arbeitsspeicher. Sie können auch festlegen, ob Aurora Serverless die Datenbank anhalten soll, wenn es keine Aktivitäten gibt, und weiterlaufen lassen soll, wenn wieder Aktivitäten beginnen.

Sie können die folgenden spezifischen Werte für die Skalierungskonfiguration festlegen:

- Minimale Aurora Capacity Unit – Aurora Serverless kann die Kapazität bis zu dieser Kapazitätseinheit reduzieren.
- Maximale Aurora Capacity Unit – Aurora Serverless kann die Kapazität bis zu dieser Kapazitätseinheit erhöhen.
- Zeitüberschreitung und Aktion für automatische Skalierung - Dieser Abschnitt gibt an, wie langeAurora Serverless wartet, um einen Skalierungspunkt zu finden, bevor die Zeitüberschreitung

eintritt. Sie legt auch fest, was zu tun ist, wenn eine Kapazitätsänderung nicht mehr funktioniert, weil sie keinen Skalierungspunkt findet. Aurora kann die Kapazitätsänderung erzwingen, um die Kapazität so schnell wie möglich auf den angegebenen Wert zu setzen. Die Kapazitätsänderung kann auch zurückgesetzt und somit abgebrochen werden. Weitere Informationen finden Sie unter Timeout-Aktion für Kapazitätsänderungen.

- Pause nach Inaktivität – Verwenden Sie die optionale Einstellung Kapazität auf 0 ACUs skalieren, wenn der Cluster inaktiv ist, um die Datenbank auf eine Verarbeitungskapazität von null zu skalieren, während sie inaktiv ist. Wenn der Datenbankverkehr wieder aufgenommen wird, nimmt Aurora automatisch die Verarbeitungskapazität wieder auf und skaliert sie in Übereinstimmung mit dem Datenverkehr.

> **Note**
>
> Wenn Sie den Kapazitätsbereich für einen DB-Cluster von Aurora Serverless ändern, erfolgt die Änderung sofort, unabhängig davon, ob Sie sie sofort oder während des nächsten geplanten Wartungsfensters anwenden möchten.

Konsole

Sie können die Skalierungskonfiguration eines Aurora-DB-Clusters über die AWS-Managementkonsole ändern.

So ändern Sie einen Aurora Serverless v1-DB-Cluster:

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Aurora Serverless v1DB-Cluster, den Sie ändern möchten.
4. Wählen Sie für Aktionen die Option Cluster ändern.
5. Ändern Sie im Abschnitt Capacity settings (Kapazitätseinstellungen) die Skalierungskonfiguration.
6. Klicken Sie auf Weiter.
7. Überprüfen Sie Ihre Änderungen auf der Seite DB-Cluster ändern und wählen Sie aus, wann diese angewendet werden sollen.
8. Wählen Sie Cluster bearbeiten aus.

AWS CLI

Um die Skalierungskonfiguration eines Aurora Serverless v1-DB-Clusters über die AWS CLI zu ändern, führen Sie den AWS CLI-Befehl modify-db-cluster aus. Geben Sie die Option `--scaling-configuration` an, um die minimale Kapazität, die maximale Kapazität und die automatische Pause zu konfigurieren, wenn es keine Verbindungen gibt. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128 und 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 und 384.

In diesem Beispiel ändern Sie die Skalierungskonfiguration eines Aurora Serverless v1-DB-Clusters namens *`sample-cluster`*.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

RDS-API

Sie können die Skalierungskonfiguration eines Aurora- DB-Clusters über die API-Operation ModifyDBCluster ändern. Geben Sie den Parameter `ScalingConfiguration` an, um die minimale Kapazität, die maximale Kapazität und die automatische Pause zu konfigurieren, wenn es keine Verbindungen gibt. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128 und 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 und 384.

Durchführen eines Upgrades der Hauptversion eines DB-Clusters von Aurora Serverless v1

> **Important**
>
> AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Sie können die Hauptversion für einen Aurora Serverless v1-DB-Cluster, der mit PostgreSQL 11 kompatibel ist, auf eine entsprechende mit PostgreSQL 13 kompatible Version aktualisieren.

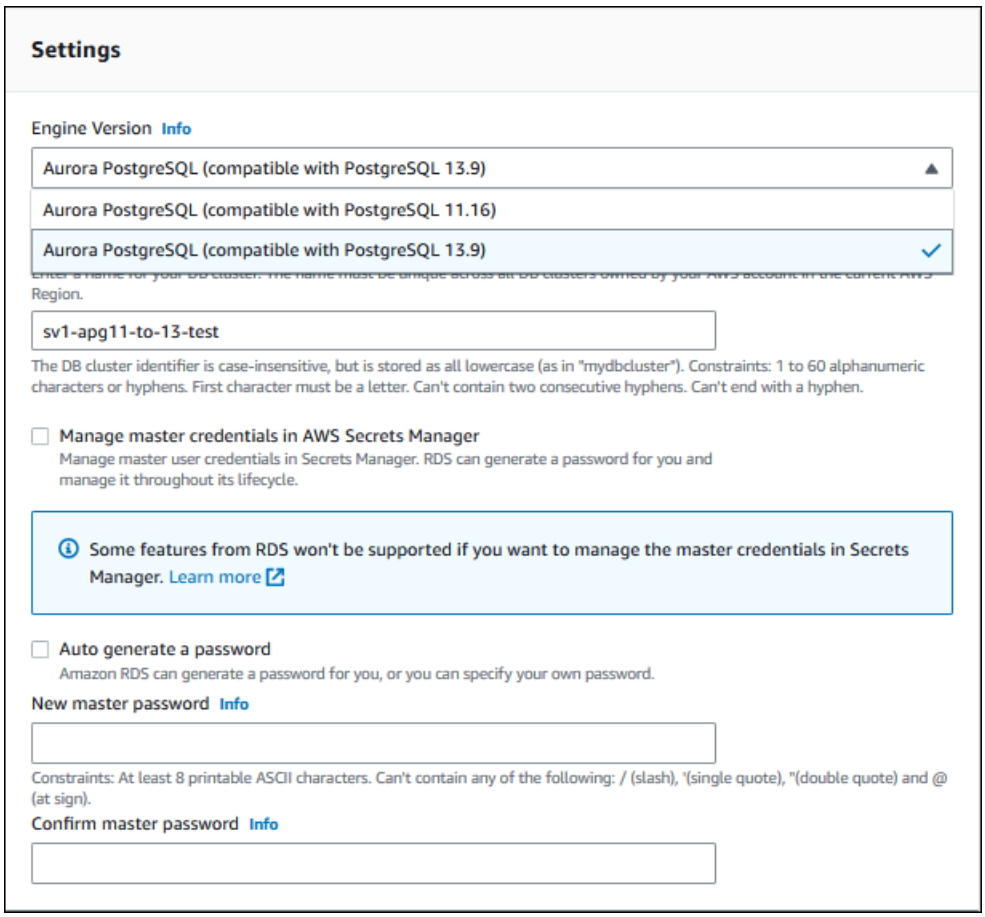
Konsole

Sie können ein direktes Upgrade des DB-Clusters von Aurora Serverless v1 mit der AWS-Managementkonsole durchführen.

So führen Sie ein Upgrade eines DB-Clusters von Aurora Serverless v1 durch

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den DB-Cluster von Aurora Serverless v1 aus, für den Sie ein Upgrade durchführen möchten.
4. Wählen Sie für Aktionen die Option Cluster ändern.
5. Wählen Sie für Version eine Versionsnummer von Aurora PostgreSQL Version 13.

   Das folgende Beispiel zeigt ein direktes Upgrade von Aurora PostgreSQL 11.16 auf 13.9.



Wenn Sie ein Hauptversions-Upgrade durchführen, lassen Sie alle anderen Eigenschaften unverändert. Wenn Sie eine der anderen Eigenschaften ändern möchten, führen Sie einen anderen Vorgang Ändern nach Abschluss des Upgrades aus.

6. Klicken Sie auf Weiter.
7. Überprüfen Sie Ihre Änderungen auf der Seite DB-Cluster ändern und wählen Sie aus, wann diese angewendet werden sollen.
8. Wählen Sie Cluster bearbeiten aus.

AWS CLI

Wenn Sie ein direktes Upgrade von einem mit PostgreSQL 11 kompatiblen DB-Cluster von Aurora Serverless v1 auf einen mit PostgreSQL 13 kompatiblen Cluster durchführen möchten, geben Sie den Parameter `--engine-version` mit einer Versionsnummer von Aurora PostgreSQL Version 13 an, die mit Aurora Serverless v1 kompatibel ist. Nehmen Sie außerdem den Parameter `--allow-major-version-upgrade` mit auf.

In diesem Beispiel ändern Sie die Hauptversion eines mit PostgreSQL 11 kompatiblen Aurora Serverless v1-DB-Clusters namens `sample-cluster`. Dadurch wird ein direktes Upgrade auf einen mit PostgreSQL 13 kompatiblen DB-Cluster von Aurora Serverless v1 durchgeführt.

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-version 13.serverless_12 \
    --allow-major-version-upgrade
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 13.serverless_12 ^
    --allow-major-version-upgrade
```

RDS-API

Wenn Sie ein direktes Upgrade von einem mit PostgreSQL 11 kompatiblen DB-Cluster von Aurora Serverless v1 auf einen mit PostgreSQL 13 kompatiblen Cluster durchführen möchten, geben Sie den Parameter `EngineVersion` mit einer Versionsnummer von Aurora PostgreSQL Version 13 an, die mit Aurora Serverless v1 kompatibel ist. Nehmen Sie außerdem den Parameter `AllowMajorVersionUpgrade` mit auf.

Konvertieren eines DB-Clusters von Aurora Serverless v1 zu einem bereitgestellten Cluster

Sie können einen DB-Cluster von Aurora Serverless v1 zu einem bereitgestellten DB-Cluster konvertieren. Um die Konvertierung durchzuführen, ändern Sie mit der AWS-CLI oder der Amazon-RDS-API die DB-Instance-Klasse in Bereitgestellt. Gehen Sie wie folgt vor, um Ihre DB-Instance-Klasse zu ändern.

Das folgende Beispiel zeigt, wie Sie mit der AWS-CLI einen Aurora Serverless v1-DB-Cluster in einen bereitgestellten Cluster konvertieren.

AWS CLI

Führen Sie den AWS CLI-Befehl modify-db-cluster aus, um einen DB-Cluster von Aurora Serverless v1 in einen bereitgestellten Cluster zu konvertieren.

Die folgenden Parameter sind erforderlich:

- `--db-cluster-identifier` – Der DB-Cluster von Aurora Serverless v1, den Sie in einen bereitgestellten Cluster konvertieren.
- `--engine-mode` – Verwenden Sie den Wert `provisioned`.
- `--allow-engine-mode-change`
- `--db-cluster-instance-class` – Wählen Sie die DB-Instance-Klasse für den bereitgestellten DB-Cluster auf der Grundlage der Kapazität des DB-Clusters von Aurora Serverless v1.

In diesem Beispiel konvertieren Sie einen DB-Cluster von Aurora Serverless v1 mit dem Namen `sample-cluster` und verwenden die DB-Instance-Klasse `db.r5.xlarge`.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--engine-mode provisioned \
--allow-engine-mode-change \
--db-cluster-instance-class db.r5.xlarge
```

Für Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--engine-mode provisioned ^
--allow-engine-mode-change ^
--db-cluster-instance-class db.r5.xlarge
```

Das folgende Beispiel zeigt, wie Sie mit der Amazon-RDS-API einen Aurora Serverless v1-DB-Cluster in einen bereitgestellten Cluster konvertieren.

RDS-API

Verwenden Sie die API-Operation ModifyDBCluster, um einen DB-Cluster von Aurora Serverless v1 in einen bereitgestellten Cluster zu konvertieren.

Die folgenden Parameter sind erforderlich:

- `DBClusterIdentifier` – Der DB-Cluster von Aurora Serverless v1, den Sie in einen bereitgestellten Cluster konvertieren.

- `EngineMode` – Verwenden Sie den Wert `provisioned`.
- `AllowEngineModeChange`
- `DBClusterInstanceClass` – Wählen Sie die DB-Instance-Klasse für den bereitgestellten DB-Cluster auf der Grundlage der Kapazität des DB-Clusters von Aurora Serverless v1.

Überlegungen bei der Konvertierung von einem Aurora Serverless v1-DB-Cluster zu einem bereitgestellten Cluster

Bei der Konvertierung von einem Aurora Serverless v1-DB-Cluster zu einem bereitgestellten Cluster sollten Sie Folgendes berücksichtigen:

- Sie können diese Konvertierung im Rahmen des Upgrades Ihres DB-Clusters von Aurora Serverless v1 auf Aurora Serverless v2 verwenden. Weitere Informationen finden Sie unter Aktualisieren eines Aurora Serverless v1-Clusters auf Aurora Serverless v2.
- Der Konvertierungsprozess erstellt eine Reader-DB-Instance im DB-Cluster, stuft die Reader-Instance zu einer Writer-Instance hoch und löscht dann die ursprüngliche Instance von Aurora Serverless v1. Wenn Sie den DB-Cluster konvertieren, können Sie keine anderen Änderungen gleichzeitig vornehmen, z. B. die DB-Engine-Version oder die DB-Cluster-Parametergruppe ändern. Der Konvertierungsvorgang wird sofort angewendet und kann nicht rückgängig gemacht werden.
- Bei der Konvertierung wird ein Backup-DB-Cluster-Snapshot des DB-Clusters für den Fall erstellt, dass ein Fehler auftritt. Der Bezeichner des DB-Cluster-Snapshots hat das Format `pre-modify-engine-mode-`*`DB_cluster_identifier`*`-`*`timestamp`*.
- Aurora verwendet die aktuelle Standard-DB-Engine-Nebenversion für den bereitgestellten DB-Cluster.
- Wenn Sie keine DB-Instance-Klasse für Ihren konvertierten DB-Cluster angeben, empfiehlt Aurora eine Klasse auf Basis der maximalen Kapazität des ursprünglichen DB-Clusters von Aurora Serverless v1. Die empfohlenen Kapazitäts- und Instance-Klassenzuordnungen sind in der folgenden Tabelle aufgeführt.

Maximale Serverless Kapazität (ACUs)	Bereitgestellte DB-Instance-Klasse
1	db.t3.small
2	db.t3.medium

Maximale Serverless Kapazität (ACUs)	Bereitgestellte DB-Instance-Klasse
4	db.t3.large
8	db.r5.large
16	db.r5.xlarge
32	db.r5.2xlarge
64	db.r5.4xlarge
128	db.r5.8xlarge
192	db.r5.12xlarge
256	db.r5.16xlarge
384	db.r5.24xlarge

Note

Abhängig von der ausgewählten DB-Instance-Klasse und Ihrer Datenbanknutzung fallen für einen bereitgestellten DB-Cluster im Vergleich zu Aurora Serverless v1 möglicherweise andere Kosten an.
Wenn Sie Ihren DB-Cluster von Aurora Serverless v1 in eine burstfähige DB-Instance-Klasse (db.t*) konvertieren, können zusätzliche Kosten für die Verwendung des DB-Clusters anfallen. Weitere Informationen finden Sie unter DB-Instance-Klassenarten.

Manuelles Skalieren der Kapazität des Aurora Serverless v1-DB-Clusters

Important

AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das

Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

In der Regel werden Aurora Serverless v1-DB-Cluster basierend auf dem Workload nahtlos skaliert. Die Kapazität kann jedoch nicht immer schnell genug skaliert werden, um plötzliche Extreme wie einen exponentiellen Anstieg der Transaktionen zu meistern. In solchen Fällen können Sie den Skalierungsvorgang manuell initiieren, indem Sie einen neuen Kapazitätswert festlegen. Nachdem Sie die Kapazität explizit festgelegt haben, skaliert Aurora Serverless v1 den DB-Cluster automatisch. Der Vorgang basiert auf der Ruhepause für die Herunterskalierung.

Sie können die Kapazität eines Aurora Serverless v1-DB-Clusters über die AWS-Managementkonsole, die AWS CLI oder die RDS-API explizit auf einen bestimmten Wert festlegen.

Konsole

Sie können die Kapazität eines Aurora-DB-Clusters über die AWS-Managementkonsole festlegen.

So ändern Sie einen Aurora Serverless v1-DB-Cluster:

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Aurora Serverless v1-DB-Cluster aus, den Sie ändern möchten.
4. Wählen Sie für Actions (Aktionen) die Option Set capacity (Kapazität festlegen) aus.
5. Wählen Sie im Fenster Datenbankkapazität skalieren Folgendes aus:

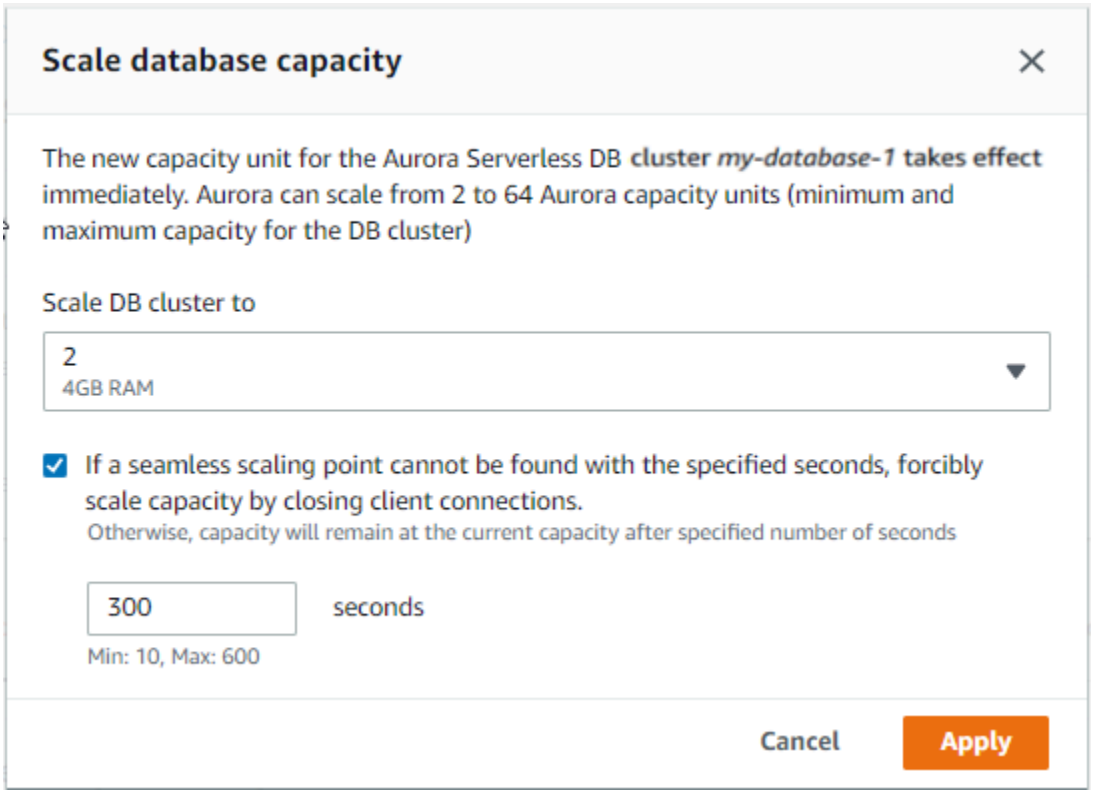
    a. Wählen Sie bei der Dropdown-Auswahl für Scale DB-Cluster to (DB-Cluster skalieren auf) die neue Kapazität aus, die Sie für Ihren DB-Cluster verwenden möchten.

    b. Wählen Sie beim Kontrollkästchen If a seamless scaling point cannot be found (Wenn kein nahtloser Skalierungspunkt gefunden werden kann) das gewünschte Verhalten für die `TimeoutAction`-Einstellung des Aurora Serverless v1-DB-Clusters aus:

    - Deaktivieren Sie diese Option, wenn Sie möchten, dass die Kapazität unverändert bleibt, wenn Aurora Serverless v1 vor dem Timeout keinen Skalierungspunkt finden kann.

- Aktivieren Sie diese Option, wenn Sie erzwingen möchten, dass der Aurora Serverless v1-DB-Cluster seine Kapazität ändert, auch wenn er vor dem Timeout keinen Skalierungspunkt finden kann. Diese Option kann dazu führen, dass Aurora Serverless v1 Verbindungen unterbricht, die verhindern, dass es einen Skalierungspunkt findet.

c. Geben Sie im Feld seconds (Sekunden) die Zeit ein, die der Aurora Serverless v1-DB-Cluster vor dem Timeout nach einem Skalierungspunkt suchen soll. Sie können einen Wert zwischen 10 Sekunden und 600 Sekunden (10 Minuten) angeben. Der Standardwert beträgt fünf Minuten (300 Sekunden). Im folgenden Beispiel wird erzwungen, dass der Aurora Serverless v1-DB-Cluster auf 2 ACUs herunterskaliert, auch wenn er innerhalb von fünf Minuten keinen Skalierungspunkt finden kann.



6. Wählen Sie Apply (Anwenden) aus.

Weitere Informationen zu Skalierungspunkten, `TimeoutAction` und Ruhephasen finden Sie unter Automatische Skalierung für Aurora Serverless v1.

AWS CLI

Um die Kapazität eines Aurora Serverless v1-DB-Clusters über die AWS CLI festzulegen, führen Sie den AWS CLI-Befehl modify-current-db-cluster-capacity aus und geben die Option `--capacity` an. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128 und 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 und 384.

In diesem Beispiel legen Sie die Kapazität eines Aurora Serverless v1-DB-Clusters namens *`sample-cluster`* auf *`64`* fest.

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --
capacity 64
```

RDS-API

Sie können die Kapazität eines Aurora-DB-Clusters über die API-Operation ModifyCurrentDBClusterCapacity festlegen. Geben Sie den Parameter `Capacity` an. Zu den gültigen Kapazitätswerten gehören die folgenden:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128 und 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192 und 384.

Anzeigen von Aurora Serverless v1-DB-Clustern

Important

AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei

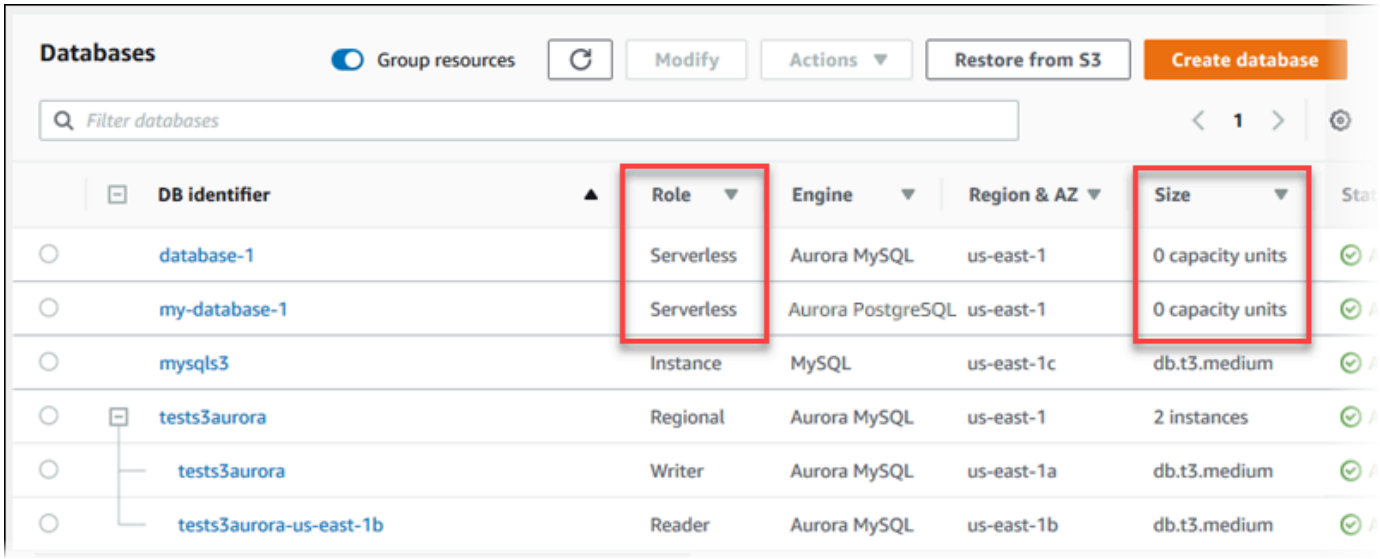
Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Nachdem Sie einen oder mehrere Aurora Serverless v1-DB-Cluster erstellt haben, können Sie anzeigen, welche DB-Cluster den Typ Serverless (Serverlos) und welche den Typ Instance haben. Sie können auch die aktuelle Anzahl von Aurora Capacity Units (ACUs) einsehen, die jeder Aurora Serverless v1-DB-Cluster verwendet. Jede ACU ist eine Kombination aus Rechenkapazität (CPU) und Arbeitsspeicherkapazität (RAM).

So zeigen Sie Ihre Aurora Serverless v1-DB-Cluster an:

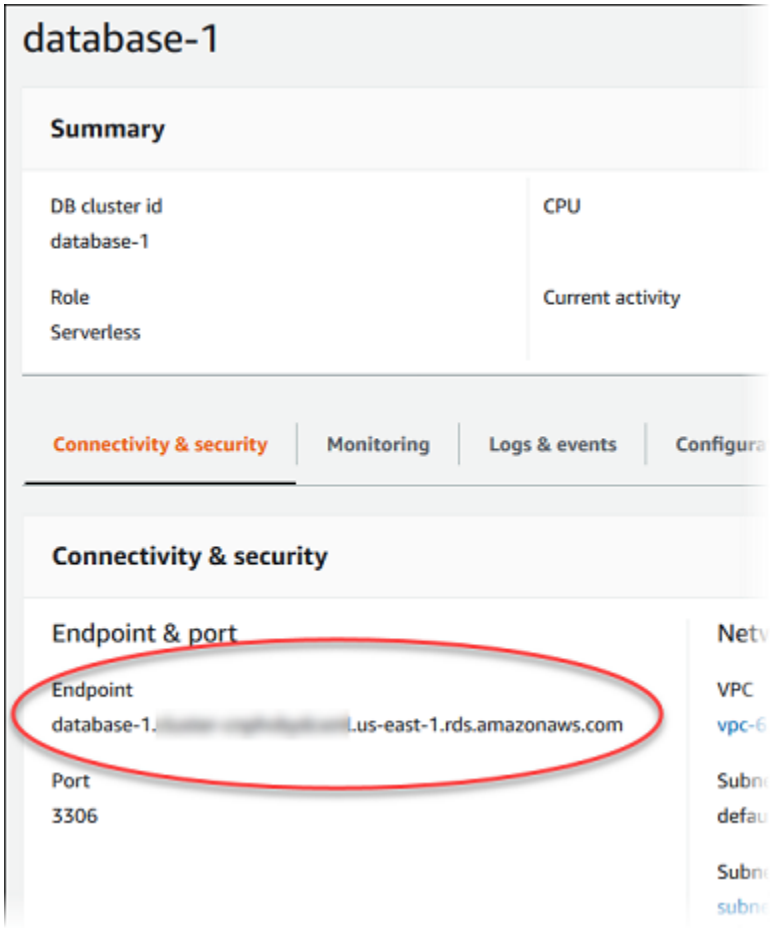
1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie oben rechts in der AWS-Managementkonsole die AWS-Region aus, in der Sie die Aurora Serverless v1-DB-Cluster erstellt haben.
3. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.

   Der DB-Cluster-Typ für die einzelnen DB-Cluster wird unter Role (Rolle) angezeigt. Bei den Aurora Serverless v1-DB-Clustern wird Serverless (Serverlos) als Typ angezeigt. Sie können die aktuelle Kapazität eines Aurora Serverless v1-DB-Clusters unter Size (Größe) anzeigen.

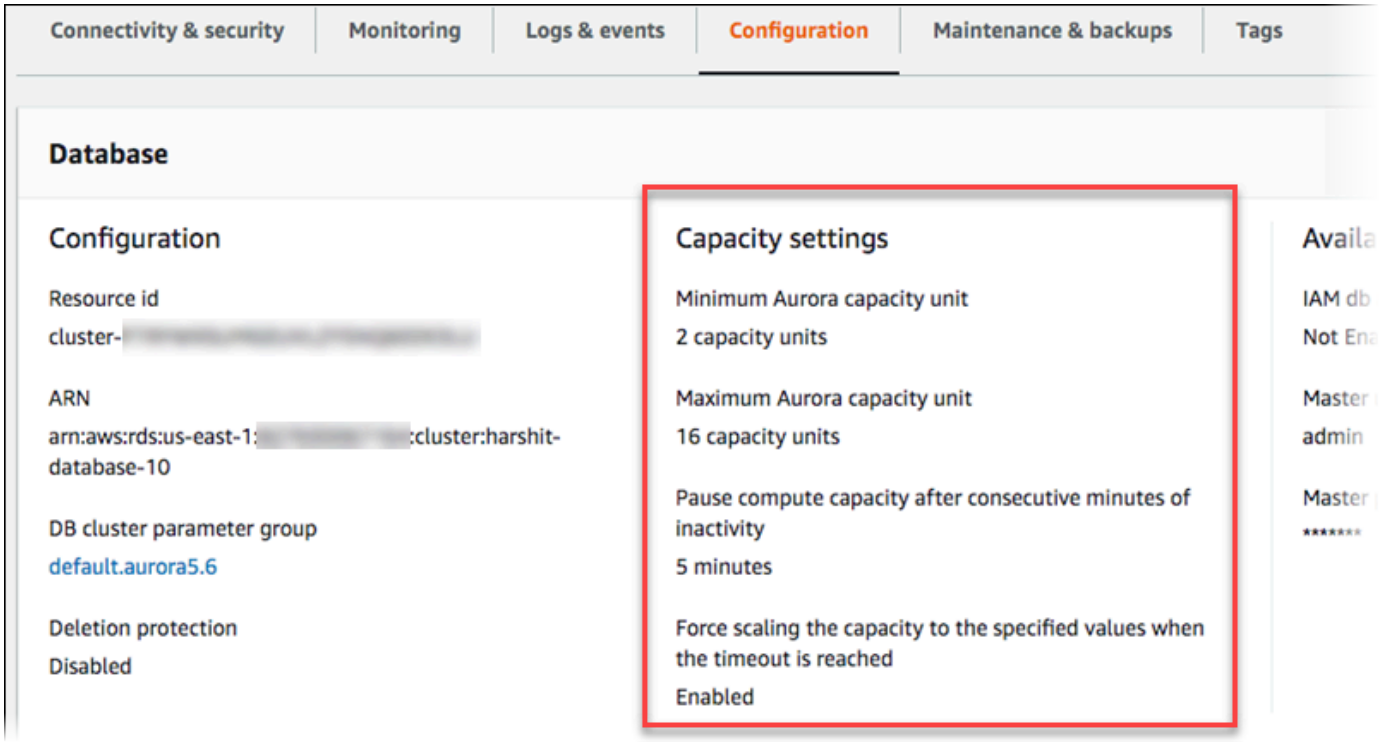


4. Wählen Sie den Namen eines Aurora Serverless v1-DB-Clusters aus, um dessen Details anzuzeigen.

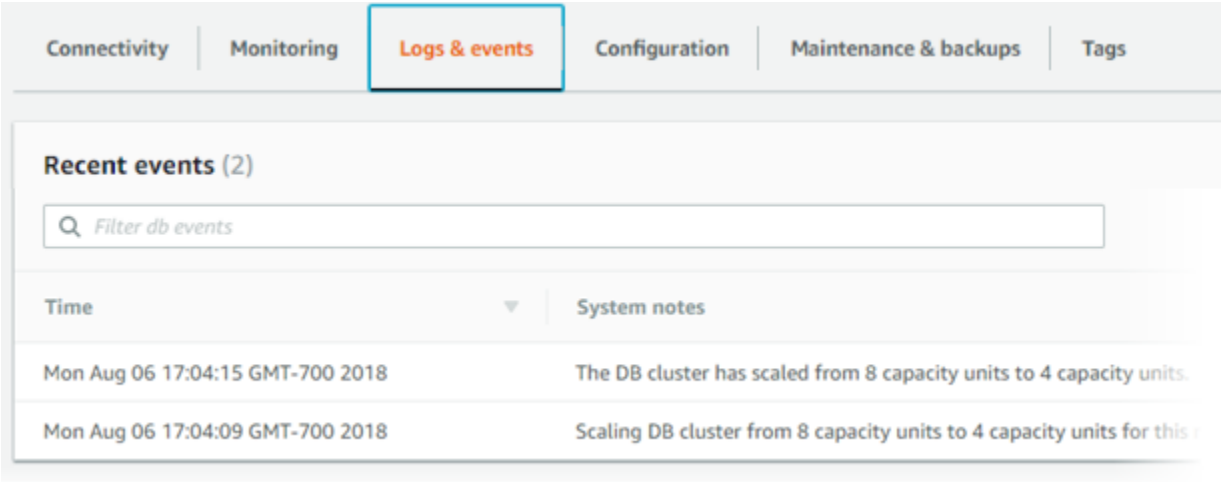
Beachten Sie auf der Registerkarte Connectivity & security (Konnektivität und Sicherheit) den Datenbankendpunkt. Verwenden Sie diesen Endpunkt, um eine Verbindung zu Ihrem Aurora Serverless v1-DB-Cluster herzustellen.



Wählen Sie die Registerkarte Configuration (Konfiguration) aus, um die Kapazitätseinstellungen anzuzeigen.



Wenn das DB-Cluster nach unten oder oben skaliert, pausiert oder fortgesetzt wird, wird ein Skalierungsereignis generiert. Wählen Sie die Registerkarte Logs & events (Protokolle und Ereignisse) aus, um aktuelle Ereignisse zu sehen. Die folgende Abbildung zeigt Beispiele für diese Ereignisse.



Überwachen der Kapazität und Skalieren von Ereignissen für den Aurora Serverless v1-DB-Cluster

Sie können Ihren Aurora Serverless v1-DB-Cluster in CloudWatch anzeigen, um die dem DB-Cluster zugewiesene Kapazität anhand der Metrik `ServerlessDatabaseCapacity` zu überwachen. Zusätzlich können Sie alle Aurora CloudWatch-Standardmetriken überwachen, wie z. B. `CPUUtilization`, `DatabaseConnections`, `Queries` usw.

Sie können veranlassen, dass Aurora einige oder alle Datenbankprotokolle in CloudWatch veröffentlicht. Sie bestimmen, welche Protokolle veröffentlicht werden sollen, indem Sie in der mit dem `general_log`-Cluster verknüpften DB-Cluster-Parametergruppe Konfigurationsparameter wie `slow_query_log` und Aurora Serverless v1 aktivieren. Anders als bei bereitgestellten Clustern müssen Sie bei Aurora Serverless v1-Clustern in den DB-Cluster-Einstellungen nicht angeben, welche Protokolltypen in CloudWatch hochgeladen werden sollen. Aurora Serverless v1-Cluster laden automatisch alle verfügbaren Protokolle hoch. Wenn Sie einen Protokollkonfigurationsparameter deaktivieren, stoppt die Veröffentlichung des Protokolls in CloudWatch. Sie können die Protokolle auch in CloudWatch löschen, wenn sie nicht mehr benötigt werden.

Erste Schritte mit Amazon CloudWatch für Ihren Aurora Serverless v1-DB-Cluster finden Sie unter Anzeigen von Aurora Serverless v1-Protokollen mit Amazon CloudWatch. Weitere Informationen zur Überwachung von Aurora-DB-Clustern über CloudWatch finden Sie unter Überwachung von Protokollereignissen in Amazon CloudWatch.

Zum Verbinden eines Aurora Serverless v1-DB-Clusters verwenden Sie den Datenbankendpunkt. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

> **Note**
>
> Sie können in Aurora Serverless v1-DB-Clustern keine direkte Verbindung zu bestimmten DB-Instances herstellen.

Löschen eines Aurora Serverless v1-DB-Clusters

> **Important**
>
> AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version. Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Je nachdem, wie Sie einen Aurora Serverless v1-DB-Cluster erstellen, ist der Löschschutz möglicherweise standardmäßig aktiviert. Sie können einen Aurora Serverless v1-DB-Cluster, für den Löschschutz aktiviert ist, nicht sofort löschen. Um Aurora Serverless v1-DB-Cluster mit Löschschutz über die AWS-Managementkonsole zu löschen, bearbeiten Sie zuerst den Cluster, um diesen Schutz zu entfernen. Informationen zur Verwendung der AWS CLI für diese Aufgabe finden Sie unter AWS CLI.

So deaktivieren Sie den Löschschutz mithilfe der AWS-Managementkonsole

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Klicken Sie im Navigationsbereich auf DB clusters (DB-Cluster).
3. Wählen Sie Ihren Aurora Serverless v1-DB-Cluster in der Liste aus.
4. Wählen Sie Modify (Ändern), um die Konfiguration des DB-Clusters zu öffnen. Auf der Seite "DB-Cluster ändern" werden die Einstellungen, Kapazitätseinstellungen und andere Konfigurationsdetails für Ihren Aurora Serverless v1-DB-Cluster geöffnet. Den Löschschutz finden Sie im Bereich Additional configuration (Zusätzliche Konfiguration).
5. Deaktivieren Sie das Kontrollkästchen Enable deletion protection (Löschschutz aktivieren) auf der Eigenschaftenkarte Additional configuration (Zusätzliche Konfiguration).
6. Klicken Sie auf Weiter. Summary of modifications (Zusammenfassung der Änderungen) wird angezeigt.

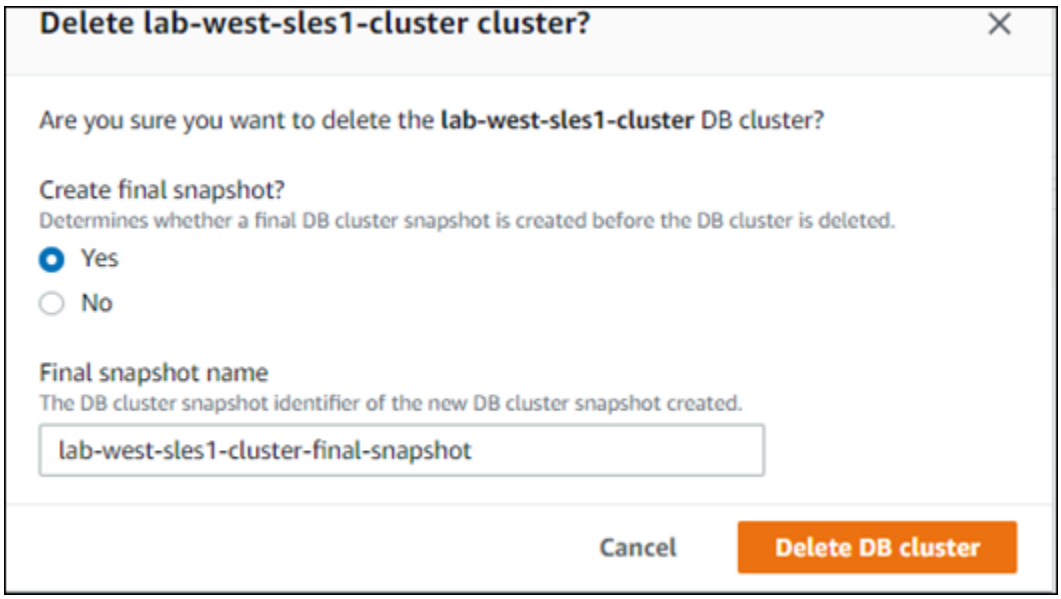
7. Wählen Sie Modify cluster (Cluster ändern), um die Zusammenfassung der Änderungen zu akzeptieren. Sie können auch Back (Zurück) wählen, um Ihre Änderungen zu bearbeiten, oder Cancel (Abbrechen), um Ihre Änderungen zu verwerfen.

Nachdem der Löschschutz nicht mehr aktiv ist, können Sie Ihren Aurora Serverless v1-DB-Cluster über die AWS-Managementkonsole löschen.

Konsole

So löschen Sie einen Aurora Serverless v1-DB-Cluster:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Bereich Resources (Ressourcen) die Option DB Clusters (DB-Cluster) aus.
3. Wählen Sie den Aurora Serverless v1-DB-Cluster aus, den Sie löschen möchten.
4. Klicken Sie bei Actions auf Delete. Sie werden aufgefordert, zu bestätigen, dass Sie den Aurora Serverless v1-DB-Cluster löschen möchten.
5. Sie sollten die vorausgewählten Optionen beibehalten:
   - Yes (Ja) bei Create final snapshot? (Finalen Snapshot erstellen?)
   - Ihr Aurora Serverless v1-DB-Cluster-Name sowie `-final-snapshot` für Final snapshot name (Name des finalen Snapshots). Sie können jedoch den Namen für Ihren finalen Snapshot in diesem Feld ändern.



Wenn Sie bei Create final snapshot? (Finalen Snapshot erstellen?) No (Nein) wählen, können Sie Ihren DB-Cluster nicht mit Snapshots oder zeitpunktbezogener Wiederherstellung wiederherstellen.

6. Wählen Sie Delete DB cluster (DB-Cluster löschen) aus.

Aurora Serverless v1 löscht Ihren DB-Cluster. Wenn Sie sich für einen finalen Snapshot entschieden haben, wird der Status Ihres Aurora Serverless v1-DB-Clusters zu "Backing-up" (Wird gesichert) geändert, bevor er gelöscht wird und nicht mehr in der Liste erscheint.

AWS CLI

Bevor Sie beginnen, konfigurieren Sie Ihre AWS CLI mit Ihrer AWS-Zugriffsschlüssel-ID, dem geheimen AWS-Zugriffsschlüssel und der AWS-Region, in der sich Ihr Aurora Serverless v1-DB-Cluster befindet. Weitere Informationen finden Sie unter Konfigurationsgrundlagen im AWS Command Line Interface-Benutzerhandbuch.

Sie können einen Aurora Serverless v1-DB-Cluster erst löschen, nachdem Sie für Cluster, die mit dieser Option konfiguriert sind, den Löschschutz deaktiviert haben. Wenn Sie versuchen, einen Cluster zu löschen, bei dem diese Schutzoption aktiviert ist, wird die folgende Fehlermeldung angezeigt.

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
```

```
  operation: Cannot delete protected Cluster, please disable deletion protection and
 try again.
```

Sie können die Löschschutzeinstellung Ihres Aurora Serverless v1-DB-Clusters ändern, indem Sie den AWS CLI-Befehl modify-db-cluster verwenden, wie im Folgenden dargestellt:

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

Dieser Befehl liefert die überarbeiteten Eigenschaften für den angegebenen DB-Cluster. Jetzt können Sie den Aurora Serverless v1-DB-Cluster löschen.

Es empfiehlt sich, immer einen finalen Snapshot zu erstellen, wenn Sie einen Aurora Serverless v1-DB-Cluster löschen. Im folgenden Beispiel zur Verwendung der AWS CLI mit delete-db-cluster wird die Vorgehensweise gezeigt. Sie geben den Namen Ihres DB-Clusters und einen Namen für den Snapshot an.

Für Linux, macOS oder Unix:

```
aws rds delete-db-cluster --db-cluster-identifier \
  your-cluster-name --no-skip-final-snapshot \
  --final-db-snapshot-identifier name-your-snapshot
```

Für Windows:

```
aws rds delete-db-cluster --db-cluster-identifier ^
  your-cluster-name --no-skip-final-snapshot ^
  --final-db-snapshot-identifier name-your-snapshot
```

Aurora Serverless v1- und Aurora-Datenbank-Engine-Versionen

Important

AWS hat das Ende der Nutzungsdauer für Aurora Serverless v1 bekannt gegeben: 31. März 2025. Alle Aurora Serverless v1-Cluster, die bis zum 31. März 2025 nicht migriert wurden, werden während des Wartungsfensters zu Aurora Serverless v2 migriert. Wenn das Upgrade fehlschlägt, konvertiert Amazon Aurora den Serverless-v1-Cluster während des Wartungsfensters in einen bereitgestellten Cluster mit der entsprechenden Engine-Version.

Falls zutreffend, registriert Amazon Aurora den konvertierten bereitgestellten Cluster bei Amazon RDS Extended Support. Weitere Informationen finden Sie unter RDS Extended Support.

Aurora Serverless v1 ist in bestimmten AWS-Regionen und nur für bestimmte Aurora-MySQL- und Aurora-PostgreSQL-Versionen verfügbar. Die aktuelle Liste der AWS-Regionen, die Aurora Serverless v1 unterstützen, sowie die spezifischen Versionen von Aurora MySQL und Aurora PostgreSQL, die in jeder Region verfügbar sind, finden Sie unter Aurora Serverless v1.

Aurora Serverless v1 verwendet die zugehörige Aurora-Datenbank-Engine, um spezifische unterstützte Versionen für jede unterstützte Datenbank-Engine zu ermitteln:

- Aurora MySQL Serverless
- Aurora PostgreSQL Serverless

Wenn Nebenversionen der Datenbank-Engines für Aurora Serverless v1 verfügbar werden, werden sie automatisch in den verschiedenen AWS-Regionen angewendet, in denen Aurora Serverless v1 verfügbar ist. Mit anderen Worten, Sie müssen Ihren Aurora Serverless v1-DB-Cluster nicht aktualisieren, um eine neue Nebenversion für die DB-Engine Ihres Clusters zu erhalten, wenn sie für Aurora Serverless v1 verfügbar ist.

Aurora MySQL Serverless

Aurora Serverless v1 ist in bestimmten AWS-Regionen und nur für bestimmte Aurora-MySQL-Versionen verfügbar. Die aktuelle Liste der AWS-Regionen, die Aurora Serverless v1 unterstützen, sowie die spezifischen Versionen von Aurora MySQL, die in jeder Region verfügbar sind, finden Sie unter Aurora Serverless v1 mit Aurora MySQL.

Weitere Informationen zu Verbesserungen und Bugfixes für Aurora MySQL Version 2 finden Sie unter Aktualisierungen der Datenbank-Engine für Amazon Aurora MySQL Version 2 in den Versionshinweisen für Aurora MySQL.

Um eine neuere Version von Aurora MySQL zu verwenden, können Sie Aurora Serverless v2 verwenden. Informationen zu den AWS-Regionen und Aurora-MySQL-Versionen, die Sie mit Aurora Serverless v2 verwenden können, finden Sie unter Aurora Serverless v2 mit Aurora MySQL. Weitere Informationen zur Verwendung von Aurora Serverless v2 finden Sie unter Verwenden von Aurora Serverless v2.

Aurora PostgreSQL Serverless

Aurora Serverless v1 ist in bestimmten AWS-Regionen und nur für bestimmte Aurora-PostgreSQL-Versionen verfügbar. Die aktuelle Liste der AWS-Regionen, die Aurora Serverless v1 unterstützen, sowie die spezifischen Versionen von Aurora PostgreSQL, die in jeder Region verfügbar sind, finden Sie unter Aurora Serverless v1 mit Aurora PostgreSQL.

Wenn Sie Aurora PostgreSQL für Ihren Aurora Serverless v1-DB-Cluster nutzen möchten, können Sie die mit Aurora PostgreSQL 13 kompatiblen Versionen verwenden. Nebenversionen für Aurora PostgreSQL-kompatible Edition enthalten nur Änderungen, die abwärtskompatibel sind. Ihr DB-Cluster von Aurora Serverless v1 wird transparent aktualisiert, wenn eine Aurora-PostgreSQL-Nebenversion für Aurora Serverless v1 in Ihrer AWS-Region verfügbar wird.

Um eine neuere Version von Aurora PostgreSQL zu nutzen, können Sie Aurora Serverless v2 verwenden. Informationen zu den AWS-Regionen und Aurora-PostgreSQL-Versionen, die Sie mit Aurora Serverless v2 verwenden können, finden Sie unter Aurora Serverless v2 mit Aurora PostgreSQL. Weitere Informationen zur Verwendung von Aurora Serverless v2 finden Sie unter Verwenden von Aurora Serverless v2.

Automatische Unterversion-Upgrades für Aurora Serverless v1

Wenn Nebenversionen der Datenbank-Engines für Aurora Serverless v1 verfügbar werden, werden sie automatisch in den verschiedenen AWS-Regionen angewendet, in denen Aurora Serverless v1 verfügbar ist. Mit anderen Worten, Sie müssen Ihren Aurora Serverless v1-DB-Cluster nicht aktualisieren, um eine neue Nebenversion für die DB-Engine Ihres Clusters zu erhalten, wenn sie für Aurora Serverless v1 verfügbar ist.

Verwenden der Daten-API von Amazon RDS

Wenn Sie die RDS-Daten-API (Daten-API) verwenden, können Sie mit einer Webservices-Schnittstelle zu Ihrem Aurora-DB-Cluster arbeiten. Die Daten-API erfordert keine beständige Verbindung zum DB-Cluster. Stattdessen bietet es einen sicheren HTTP-Endpunkt und eine Integration mit AWS SDKs. Über den Endpunkt können Sie SQL-Anweisungen ausführen, ohne Verbindungen zu verwalten.

Benutzer müssen bei Aufrufen an die Daten-API keine Anmeldeinformationen weitergeben, da die Daten-API Datenbankanmeldeinformationen verwendet, die in gespeichert sind AWS Secrets Manager. Um Anmeldeinformationen in Secrets Manager speichern zu können, müssen Benutzer die entsprechenden Berechtigungen zur Verwendung von Secrets Manager und der Daten-API erhalten. Weitere Informationen zum Autorisieren von Benutzern finden Sie unter Autorisieren des Zugriffs auf die Amazon-RDS-Daten-API.

Sie können die Daten-API auch verwenden, um Amazon Aurora in andere AWS Anwendungen wie AWS Lambda AWS AppSync, und zu integrieren AWS Cloud9. Die Daten-API bietet auch eine sicherere Möglichkeit für die Verwendung von AWS Lambda. Sie können damit auf Ihr DB-Cluster zugreifen, ohne eine Lambda-Funktion für den Zugriff auf Ressourcen in einer Virtual Private Cloud (VPC) konfigurieren zu müssen. Weitere Informationen finden Sie unter AWS Lambda, AWS AppSync und AWS Cloud9.

Sie können die Daten-API aktivieren, wenn Sie den Aurora-DB-Cluster erstellen. Sie können die Konfiguration später auch ändern. Weitere Informationen finden Sie unter Aktivieren der Amazon-RDS-Daten-API.

Nach Aktivierung der Daten-API können Sie auch den Abfrage-Editor verwenden, um Ad-hoc-Abfragen auszuführen, ohne ein Abfrage-Werkzeug für den Zugriff auf Aurora in einer VPC zu konfigurieren. Weitere Informationen finden Sie unter Arbeiten mit dem Aurora-Abfrage-Editor.

Themen

- Verfügbarkeit von Regionen und Versionen für die Amazon-RDS-Daten-API
- Verwendung IPv6 mit der Amazon RDS-Daten-API
- Einschränkungen der Amazon-RDS-Daten-API
- Vergleich zum Verhalten der Daten-API von Amazon RDS – bei Aurora Serverless v2 und bereitgestellten Clustern und bei Aurora Serverless v1-Clustern
- Autorisieren des Zugriffs auf die Amazon-RDS-Daten-API

- Aktivieren der Amazon-RDS-Daten-API
- Erstellen eines Amazon-VPC-Endpunkts für die Amazon-RDS-Daten-API (AWS PrivateLink)
- Aufrufen der Daten-API von Amazon RDS
- Verwenden der Java-Client-Bibliothek für die RDS-Daten-API
- Verarbeiten von Abfrageergebnissen der Amazon-RDS-Data-API im JSON-Format
- Fehlerbehebung bei Problemen mit der Amazon-RDS-Daten-API
- Protokollieren von Aufrufen der Amazon-RDS-Daten-API mit AWS CloudTrail
- Überwachen von RDS-Daten-API-Abfragen mit Performance Insights

Verfügbarkeit von Regionen und Versionen für die Amazon-RDS-Daten-API

Informationen zu den Regionen und Engine-Versionen, die für die Daten-API verfügbar sind, finden Sie in folgenden Abschnitten.

Cluster-Typ	Verfügbarkeit von Regionen und Versionen	
Aurora PostgreSQL, bereitgestellt und Serverless v2	Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellt	
Aurora MySQL, bereitgestellt und Serverless v2	Daten-API mit Aurora MySQL Serverless v2 und bereitgestellt	
Aurora PostgreSQL Serverless v1	Daten-API mit Aurora PostgreSQL Serverless v1	
Aurora MySQL Serverless v1	Daten-API mit Aurora MySQL Serverless v1	

Wenn Sie kryptografische Module benötigen, die von FIPS 140-2 validiert werden, wenn Sie über eine Befehlszeilenschnittstelle oder eine API auf die Daten-API zugreifen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-2.

Verwendung IPv6 mit der Amazon RDS-Daten-API

Die Amazon RDS Data API unterstützt IPv6 Konnektivität über Dual-Stack-Endpunkte. Auf diese Weise können Sie mithilfe von IPv6 Adressen eine Verbindung zur Daten-API herstellen und gleichzeitig die Abwärtskompatibilität mit beibehalten. IPv4

IPv6 Endpunkt-Unterstützung

Die Daten-API bietet Dual-Stack-Endpunkte, die IPv4 sowohl IPv6 Verbindungen als auch unterstützen. Diese Endpunkte verwenden die `.aws`-Domain anstelle der herkömmlichen `.amazonaws.com`-Domain.

Verfügbare Endpunkttypen

Öffentliche Dual-Stack-Endpunkte

- Format: `rds-data.`*`region`*`.api.aws`
- Beispiel: `rds-data.us-east-1.api.aws`

FIPS-Dual-Stack-Endpunkte

- Format: `rds-data-fips.`*`region`*`.api.aws`
- Beispiel: `rds-data-fips.us-east-1.api.aws`

PrivateLink IPv6 Endpunkte

- Verfügbar über VPC-Endpunkte mit Support IPv6
- Ermöglicht private IPv6 Konnektivität innerhalb Ihrer VPC

Endpunkte, die nur aus IPv4 Legacy-Versionen bestehen

Die vorhandenen `.amazonaws.com` Endpunkte unterstützen IPv4 weiterhin reine Verbindungen:

- `rds-data.`*`region`*`.amazonaws.com`
- `rds-data-fips.`*`region`*`.amazonaws.com`

> **Note**
>
> Ältere Endpunkte bleiben unverändert, um die Abwärtskompatibilität mit bestehenden Anwendungen zu gewährleisten.

Endpunkte verwenden IPv6

Um die Anwendung IPv6 mit der Daten-API zu verwenden, aktualisieren Sie Ihre Anwendung so, dass sie die neuen Dual-Stack-Endpunkte verwendet. Ihre Anwendung wird automatisch verwendet, IPv6 sofern verfügbar, oder es wird darauf zurückgegriffen. IPv4

Allgemeine Hinweise zur Einrichtung IPv6 in Ihrer VPC finden Sie unter Migration zu IPv6 im Amazon VPC-Benutzerhandbuch.

Sie können IPv6 Endpoints auf zwei Arten konfigurieren:

- Umgebungsvariable verwenden: `AWS_USE_DUALSTACK_ENDPOINT=true` In Ihrer IPv6 Umgebung festgelegt. Das AWS CLI und erstellt AWS SDKs automatisch die entsprechenden `api.aws` Endpunkte, ohne dass Sie den Endpunkt URLs manuell angeben müssen.
- Verwenden eines expliziten Endpunkts URLs: Geben Sie die Dual-Stack-Endpunkt-URL direkt in Ihren AWS CLI Befehlen oder Ihrer SDK-Konfiguration an, wie in den folgenden Beispielen gezeigt.

AWS CLI Konfiguration

Konfigurieren Sie die AWS CLI für die Verwendung von IPv6 Endpunkten, indem Sie die Endpunkt-URL angeben:

Für Linux, macOS oder Unix:

```
aws rds-data execute-statement \
    --endpoint-url https://rds-data.us-east-1.api.aws \
    --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:my-cluster" \
    --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret" \
    --database "mydb" \
    --sql "SELECT * FROM users LIMIT 10"
```

Für Windows:

```
aws rds-data execute-statement ^
```

```
 --endpoint-url https://rds-data.us-east-1.api.aws ^
 --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:my-cluster" ^
 --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret" ^
 --database "mydb" ^
 --sql "SELECT * FROM users LIMIT 10"
```

AWS SDK-Konfiguration

Für AWS SDKs die Verwendung von Dual-Stack-Endpunkten konfigurieren:

Python

```
import boto3

# Create RDS Data API client with IPv6 dual-stack endpoint
client = boto3.client(
    'rds-data',
    endpoint_url='https://rds-data.us-east-1.api.aws'
)

# Execute a SQL statement
response = client.execute_statement(
    resourceArn='arn:aws:rds:us-east-1:123456789012:cluster:my-cluster',
    secretArn='arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret',
    database='mydb',
    sql='SELECT * FROM users LIMIT 10'
)

print(response['records'])
```

Java

```
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import java.net.URI;

// Create RDS Data API client with IPv6 dual-stack endpoint
RdsDataClient client = RdsDataClient.builder()
    .endpointOverride(URI.create("https://rds-data.us-east-1.api.aws"))
    .build();

// Execute a SQL statement
```

```
ExecuteStatementRequest request = ExecuteStatementRequest.builder()
    .resourceArn("arn:aws:rds:us-east-1:123456789012:cluster:my-cluster")
    .secretArn("arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret")
    .database("mydb")
    .sql("SELECT * FROM users LIMIT 10")
    .build();

ExecuteStatementResponse response = client.executeStatement(request);
System.out.println(response.records());
```

JavaScript

```
const { RDSDataClient, ExecuteStatementCommand } = require("@aws-sdk/client-rds-
data");

// Create RDS Data API client with IPv6 dual-stack endpoint
const client = new RDSDataClient({
    endpoint: "https://rds-data.us-east-1.api.aws"
});

// Execute a SQL statement
const command = new ExecuteStatementCommand({
    resourceArn: "arn:aws:rds:us-east-1:123456789012:cluster:my-cluster",
    secretArn: "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret",
    database: "mydb",
    sql: "SELECT * FROM users LIMIT 10"
});

const response = await client.send(command);
console.log(response.records);
```

Verwenden mit AWS PrivateLink IPv6

Sie können VPC-Endpunkte für die Daten-API erstellen, die IPv6 Konnektivität innerhalb Ihrer VPC unterstützen. Ausführliche Anleitungen zum Erstellen von VPC-Endpunkten für die Daten-API finden Sie unter Erstellen eines Amazon-VPC-Endpunkts für die Amazon-RDS-Daten-API (AWS PrivateLink).

Stellen Sie beim Erstellen eines VPC-Endpunkts für den IPv6 Support sicher, dass:

- Ihre VPC und Subnetze sind für die Unterstützung konfiguriert IPv6

- Sicherheitsgruppen ermöglichen den IPv6 Datenverkehr an den erforderlichen Ports (normalerweise 443 für HTTPS)
- Netzwerke ACLs sind so konfiguriert, dass sie IPv6 Datenverkehr zulassen

Überlegungen zur Migration

Beachten Sie bei der Migration zu IPv6 Endpunkten Folgendes:

- Schrittweise Migration: Sie können Anwendungen schrittweise migrieren, indem Sie URLs jeweils eine Anwendung auf dem Endpunkt aktualisieren.
- Netzwerkkompatibilität: Stellen Sie IPv6 vor der Migration sicher, dass Ihre Netzwerkinfrastruktur dies unterstützt.
- Sicherheitsrichtlinien: Aktualisieren Sie die Sicherheitsgruppenregeln und das Netzwerk ACLs , um bei Bedarf IPv6 Datenverkehr zuzulassen.
- Überwachung: Aktualisieren Sie die Überwachungs- und Protokollierungskonfigurationen, um IPv6 Adressen zu verwalten.

> **Note**
>
> Datenbankverbindungsadressen: Wenn IPv6 Endpunkte für die Daten-API verwendet werden, werden in den zugrunde liegenden Datenbankverbindungen und Datenbankprotokollen weiterhin IPv4 Adressen angezeigt. Dieses Verhalten ist zu erwarten und beeinträchtigt nicht die Funktionalität Ihrer IPv6 -fähigen Anwendungen.

Fehlerbehebung bei IPv6 der Konnektivität

Wenn Sie Probleme mit der IPv6 Konnektivität haben, überprüfen Sie Folgendes:

Netzwerkkonfiguration

: Stellen Sie sicher, dass Ihr Netzwerk IPv6 Routing unterstützt IPv6 und ob es richtig konfiguriert ist.

DNS-Auflösung

: Stellen Sie sicher, dass Ihr DNS-Resolver AAAA-Datensätze für die Dual-Stack-Endpunkte auflösen kann.

Sicherheitsgruppen

Aktualisieren Sie die Sicherheitsgruppenregeln, um IPv6 Datenverkehr an den erforderlichen Ports zuzulassen (normalerweise 443 für HTTPS).

Client-Bibliotheken

Stellen Sie sicher, dass Ihre HTTP-Clientbibliotheken Dual-Stack-Konnektivität unterstützen IPv6 .

Einschränkungen der Amazon-RDS-Daten-API

Für die RDS-Daten-API gelten die folgenden Beschränkungen:

- Sie können Daten-API-Abfragen nur auf Writer-Instances in einem DB-Cluster ausführen. Writer-Instances akzeptieren jedoch sowohl Schreib- als auch Leseanfragen.
- Bei globalen Aurora-Datenbanken können Sie die Daten-API sowohl auf dem primären als auch auf dem sekundären DB-Cluster aktivieren. Ein sekundärer Cluster hat jedoch erst dann eine Writer-Instance, wenn er zum primären Cluster hochgestuft wurde. Die Daten-API benötigt Zugriff auf die Writer-Instance, um Abfragen verarbeiten zu können. Das gilt sogar für Leseabfragen. Aus diesem Grund schlagen Lese- und Schreibanfragen fehl, die an den sekundären Cluster gesendet werden, solange ihm eine Writer-Instance fehlt. Sobald ein sekundärer Cluster hochgestuft wurde und eine Writer-Instance verfügbar ist, werden Daten-API-Abfragen auf dieser DB-Instance erfolgreich verarbeitet.
- Die Daten-API wird in T-DB-Instance-Klassen nicht unterstützt.
- Für Aurora Serverless v2- und bereitgestellte DB-Cluster unterstützt die RDS-Daten-API einige Datentypen nicht. Die Liste der unterstützten Typen finden Sie unter the section called “Daten-API mit Aurora Serverless v2 oder mit Aurora Serverless v1 – Vergleich”.
- Für Datenbanken mit Aurora PostgreSQL Version 14 und höher unterstützt die Daten-API nur `scram-sha-256` für die Passwortverschlüsselung.
- Das Limit für Antwortgrößen beträgt 1 MiB. Wenn der Aufruf mehr als 1 MiB an Antwortdaten zurückgibt, wird der Aufruf beendet.
- Für Aurora Serverless v1 beträgt die maximale Anzahl der Anforderungen pro Sekunde 1 000. Für alle anderen unterstützten Datenbanken gibt es kein Limit.
- Die Größenbegrenzung der Data-API beträgt 64 KB pro Zeile in der von der Datenbank zurückgegebenen Ergebnismenge. Stellen Sie sicher, dass jede Zeile in einem Ergebnissatz höchstens 64 KB groß ist.

Vergleich zum Verhalten der Daten-API von Amazon RDS – bei Aurora Serverless v2 und bereitgestellten Clustern und bei Aurora Serverless v1-Clustern

Die neuesten Verbesserungen der Daten-APIs von Amazon RDS machen Daten-APIs für Cluster verfügbar, die aktuelle Versionen von PostgreSQL- oder MySQL-Engines ausführen. Diese Cluster können so konfiguriert werden, dass sie Aurora Serverless v2 verwenden, oder bereitgestellte Instance-Klassen wie `db.r6g` oder `db.r6i`.

In den folgenden Abschnitten werden die Unterschiede beschrieben, die hinsichtlich der Daten-API von Amazon RDS zwischen Aurora Serverless v2 und bereitgestellten DB-Clustern auf der einen Seite und Aurora Serverless v1-DB-Clustern auf der anderen Seite bestehen. Aurora Serverless v1 DB-Cluster verwenden den `serverless`-Engine-Modus. Bereitgestellte DB-Cluster verwenden den `provisioned`-Engine-Modus. Ein Aurora Serverless v2-DB-Cluster verwendet auch den `provisioned`-Engine-Modus und enthält eine oder mehrere Aurora Serverless v2-DB-Instances mit der `db.serverless`-Instance-Klasse.

Maximale Anzahl von Anforderungen pro Sekunde

Aurora Serverless v1

Daten-APIs können bis zu 1 000 Anforderungen pro Sekunde stellen.

Aurora Serverless v2

Daten-APIs können eine unbegrenzte Anzahl von Anforderungen pro Sekunde stellen.

Aktivieren oder Deaktivieren der Daten-API von Amazon RDS in einer vorhandenen Datenbank

Aurora Serverless v1

- Mit der Amazon-RDS-API: Verwenden Sie den Vorgang `ModifyCluster` und legen Sie für den Parameter `EnableHttpEndpoint` entweder `True` oder `False` fest.
- Mit der AWS CLI: Verwenden Sie die Operation `modify-db-cluster` mit der Option `--enable-http-endpoint` oder `--no-enable-http-endpoint`.

Aurora Serverless v2

- Mit der Amazon-RDS-API: Verwenden Sie die Operationen `EnableHttpEndpoint` und `DisableHttpEndpoint`.
- Mit der AWS CLI: Verwenden Sie die Operationen `enable-http-endpoint` und `disable-http-endpoint`.

CloudTrail-Ereignisse

Aurora Serverless v1

Ereignisse aus Daten-API-Aufrufen sind Verwaltungsereignisse. Diese Ereignisse werden in einem Trail standardmäßig automatisch eingeschlossen. Weitere Informationen finden Sie unter the section called “Ausschließen von Daten-API-Ereignissen aus einem CloudTrail-Trail (nur Aurora Serverless v1)”.

Aurora Serverless v2

Ereignisse aus Daten-API-Aufrufen sind Datenereignisse. Diese Ereignisse werden in einem Trail standardmäßig automatisch ausgeschlossen. Weitere Informationen finden Sie unter the section called “Einschließen von Daten-API-Ereignissen in einen Cloudtrail-Trail”.

Unterstützung für mehrere Anweisungen

Aurora Serverless v1

- Mehrere Anweisungen werden für Aurora MySQL nicht unterstützt.
- Mehrere Anweisungen geben für Aurora PostgreSQL nur die erste Abfrageantwort zurück.

Aurora Serverless v2

Mehrere Anweisungen werden nicht unterstützt. Wenn Sie versuchen, mehrere Anweisungen in einem einzigen API-Aufruf auszuführen, wird der Fehler “`An error occurred (ValidationException) when calling the ExecuteStatement operation: Multistatements aren't supported.`” zurückgegeben. Um mehrere Anweisungen auszuführen, nutzen Sie separate `ExecuteStatement`-API-Aufrufe oder `BatchExecuteStatement` für die Stapelverarbeitung.

Das folgende Beispiel zeigt die Fehlermeldung, die ein API-Aufruf zurückgibt, wenn Sie versuchen, mehrere Anweisungen darin auszuführen.

```
 aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM your_table; Select * FROM next_table;

                                "An error occurred (ValidationException) when calling
 the ExecuteStatement operation: Multistatements aren't supported.
```

Im folgenden Beispiel werden mehrere Anweisungen mit separaten `ExecuteStatement`-API-Aufrufen ausgeführt.

```
aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM your_table;"

aws rds-data execute-statement \
    --resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
    --secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
    --database "your_database" \
    --sql "SELECT * FROM next_table;"
```

Gleichzeitige Anforderungen für dieselbe Transaktions-ID

Aurora Serverless v1

Nachfolgende Anforderungen warten, bis die aktuelle Anforderung abgeschlossen ist. Ihre Anwendung muss Zeitüberschreitungsfehler beheben, wenn die Wartezeit zu lang ist.

Aurora Serverless v2

Wenn die Daten-API mehrere Anforderungen mit derselben Transaktions-ID erhält, gibt sie sofort diesen Fehler zurück:

`DatabaseErrorException: Transaction is still running a query`

Dieser Fehler tritt in zwei Situationen auf:

- Ihre Anwendung stellt asynchrone Anforderungen (wie JavaScript-Versprechen) mit derselben Transaktions-ID.
- Eine frühere Anforderung mit dieser Transaktions-ID wird noch verarbeitet.

Das folgende Beispiel zeigt alle Anforderungen, die parallel zu `promise.all()` ausgeführt werden.

```
const api_calls = [];
for (let i = 0; i < 10; i++) {
api_calls.push(
    client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    )
);
}
await Promise.all(api_calls);
```

Um diesen Fehler zu beheben, warten Sie, bis die aktuelle Anforderung abgeschlossen ist, bevor Sie eine weitere Anforderung mit derselben Transaktions-ID senden, oder entfernen Sie die Transaktions-ID, um parallele Anforderungen zu ermöglichen.

Das folgende Beispiel zeigt einen API-Aufruf, der die sequentielle Ausführung mit derselben Transaktions-ID verwendet.

```
 for (let i = 0; i < 10; i++) {
    await client.send(
    new ExecuteStatementCommand({
        ...params,
        sql: `insert into table_name values (i);`,
        transactionId
    })
    ).promise()
);
}
```

Wie sich BatchExecuteStatement verhält

Weitere Informationen zu `BatchExecuteStatement` finden Sie unter BatchExecuteStatement.

Aurora Serverless v1

Das Objekt für generierte Felder im Aktualisierungsergebnis enthält eingefügte Werte.

Aurora Serverless v2

- Für Aurora MySQL enthält das Objekt für generierte Felder im Aktualisierungsergebnis eingefügte Werte.
- Für Aurora PostgreSQL bleibt das Objekt für generierte Felder leer.

Wie sich ExecuteSQL verhält

Weitere Informationen zu `ExecuteSQL` finden Sie unter ExecuteSQL.

Aurora Serverless v1

Der Vorgang `ExecuteSQL` ist veraltet.

Aurora Serverless v2

Der Vorgang `ExecuteSQL` wird nicht unterstützt.

Wie sich ExecuteStatement verhält

Weitere Informationen zu `ExecuteStatement` finden Sie unter ExecuteStatement.

Aurora Serverless v1

Der Parameter `ExecuteStatement` unterstützt das Abrufen von multidimensionalen Array-Spalten und allen erweiterten Datentypen.

Aurora Serverless v2

Der Parameter `ExecuteStatement` unterstützt keine multidimensionalen Array-Spalten. Er unterstützt auch bestimmte PostgreSQL-Datentypen nicht, einschließlich geometrischer und monetärer Typen. Wenn eine Daten-API auf einen nicht unterstützten Datentyp stößt, gibt sie diesen Fehler zurück: `UnsupportedResultException: The result contains the unsupported data type data_type`.

Um dieses Problem zu umgehen, konvertieren Sie den nicht unterstützten Datentyp in `TEXT`. Im folgenden Beispiel wird ein nicht unterstützter Datentyp in `TEXT` konvertiert.

```
SELECT custom_type::TEXT FROM my_table;--
ORSELECT CAST(custom_type AS TEXT) FROM my_table;
```

Eine Liste der unterstützten Datentypen für jede Aurora-Datenbank-Engine finden Sie in der Referenz zu Daten-API-Vorgängen.

Wie sich der Parameter Schema verhält

Aurora Serverless v1

Der Parameter `Schema` wird nicht unterstützt. Wenn Sie den Parameter `Schema` in einen API-Aufruf aufnehmen, ignoriert die Daten-API den Parameter.

Aurora Serverless v2

Der Parameter `Schema` ist veraltet. Wenn Sie den Parameter `Schema` in einen API-Aufruf aufnehmen, gibt die Daten-API diesen Fehler zurück: `ValidationException: The schema parameter isn't supported`. Im folgenden Beispiel wird ein Daten-API-Aufruf angezeigt, der den Fehler `ValidationException` zurückgibt.

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--schema "your_schema" \
--sql "SELECT * FROM your_table LIMIT 10"
```

Um dieses Problem zu lösen, entfernen Sie den Parameter `Schema` aus Ihrem API-Aufruf.

Das folgende Beispiel zeigt einen Daten-API-Aufruf, bei dem der Parameter `Schema` entfernt wurde.

```
aws rds-data execute-statement \
--resource-arn "arn:aws:rds:region:account:cluster:cluster-name" \
--secret-arn "arn:aws:secretsmanager:region:account:secret:secret-name" \
--database "your_database" \
--sql "SELECT * FROM your_table LIMIT 10"
```

Autorisieren des Zugriffs auf die Amazon-RDS-Daten-API

Benutzer können die Amazon-RDS-Daten-API (Daten-API) nur aufrufen, wenn sie dazu autorisiert sind. Sie können einem Benutzer die Erlaubnis zur Verwendung der Daten-API erteilen, indem Sie eine AWS Identity and Access Management (IAM-) Richtlinie anhängen, die seine Rechte definiert. Sie können die Richtlinie auch einer Rolle anfügen, wenn Sie IAM-Rollen verwenden. Eine AWS verwaltete Richtlinie umfasst Berechtigungen für die Daten-API. `AmazonRDSDataFullAccess`

Die `AmazonRDSDataFullAccess` Richtlinie umfasst auch Berechtigungen, aus denen der Benutzer den Wert eines Geheimnisses abrufen kann AWS Secrets Manager. Benutzer müssen Secrets Manager zum Speichern von Geheimnissen verwenden, die sie in ihren Aufrufen der Daten-API verwenden können. Die Verwendung von Geheimnissen bedeutet, dass Benutzer keine Datenbank-Anmeldeinformationen für die Ressourcen einschließen müssen, die das Ziel ihrer Aufrufe an die Daten-API sind. Die Daten-API ruft Secrets Manager transparent auf, wodurch die Anforderung des Geheimnisses durch den Benutzer zugelassen (oder abgelehnt) wird. Informationen zum Einrichten von Geheimnissen zur Verwendung mit der Daten-API finden Sie unter Speichern von Datenbankanmeldedaten in AWS Secrets Manager.

Die Richtlinie `AmazonRDSDataFullAccess` ermöglicht den vollständigen Zugriff (über die Daten-API) auf Ressourcen. Sie können den Geltungsbereich einschränken, indem Sie eigene Richtlinien definieren, die den Amazon-Ressourcennamen (ARN) einer Ressource angeben.

Die folgende Richtlinie zeigt beispielsweise ein Beispiel für die Mindestberechtigungen, die ein Benutzer für den Zugriff auf die Daten-API für den durch den ARN identifizierten DB-Cluster benötigt. Die Richtlinie enthält die nötigen Berechtigungen, um auf Secrets Manager zuzugreifen und die Autorisierung für die DB-Instance für den Benutzer zu erhalten.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
```

```
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"
        }
    ]
}
```

Sie sollten für das Element „Ressourcen“ in Ihren Richtlinienanweisungen (wie im Beispiel gezeigt) einen spezifischen ARN anstelle eines Platzhalters (*) verwenden.

Arbeiten mit der Tag-basierten Autorisierung

Die RDS-Daten-API (Daten-API) und Secrets Manager unterstützen beide die Tag-basierte Autorisierung. Tags sind Schlüssel-Wert-Paare, die eine Ressource, z. B. einen RDS-Cluster, mit einem zusätzlichen Zeichenfolgenwert kennzeichnen, z. B.

- `environment:production`
- `environment:development`

Sie können auf Ihre Ressourcen Tags zum Zweck der Kostenzuweisung, der Ausführungsunterstützung, der Zugriffskontrolle und zu vielen weiteren Zwecken anwenden. (Wenn Sie noch keine Tags für Ihre Ressourcen anwenden und Tags anwenden möchten, finden Sie weitere Informationen unter Markieren von Amazon-RDS-Ressourcen.) Sie können die Tags in Ihren Richtlinienanweisungen verwenden, um den Zugriff auf die RDS-Cluster einzuschränken, die mit diesen Tags gekennzeichnet sind. Ein Aurora-DB-Cluster könnte beispielsweise Tags besitzen, die die Umgebung als Produktions- oder Entwicklungsumgebung kennzeichnen.

Das folgende Beispiel zeigt, wie Sie in Ihren Richtlinienanweisungen Tags verwenden können. Diese Anweisung erfordert, dass sowohl der Cluster als auch der in der Daten-API-Anforderung übergebene Schlüssel das Tag `environment:production` enthalten.

So wird die Richtlinie angewendet: Wenn ein Benutzer einen Aufruf über die Daten-API ausführt, wird die Anforderung an den Service gesendet. Die Daten-API überprüft zunächst, ob der in der Anforderung übergebene Cluster-ARN mit `environment:production` markiert ist. Anschließend wird Secrets Manager aufgerufen, um den Wert des Geheimnisses des Benutzers in der Anforderung abzurufen. Secrets Manager überprüft auch, ob das Geheimnis des Benutzers mit `environment:production` markiert ist. Wenn ja, verwendet die Daten-API den abgerufenen Wert für das DB-Passwort des Benutzers. Wenn dieser ebenfalls korrekt ist, wird die Daten-API-Anforderung für den Benutzer erfolgreich aufgerufen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                 "secretsmanager:GetSecretValue"
               ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                  "rds-data:*"
               ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
```

```
                }
            }
        }
    ]
}
```

Das Beispiel zeigt separate Aktionen für `rds-data` und `secretsmanager` für die Daten-API und Secrets Manager. Sie können jedoch Aktionen kombinieren und Tag-Bedingungen auf viele verschiedene Arten definieren, um Ihre spezifischen Anwendungsfälle zu unterstützen. Weitere Informationen finden Sie unter Verwenden identitätsbasierter Richtlinien (IAM-Richtlinien) für Secrets Manager.

Im Element „Bedingung“ der Richtlinie können Sie Tag-Schlüssel aus den folgenden Optionen auswählen:

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

Weitere Informationen zu Ressourcen-Tags und deren Verwendung `aws:TagKeys` finden Sie unter Steuern des Zugriffs auf AWS Ressourcen mithilfe von Ressourcen-Tags.

> **Note**
>
> Sowohl Daten-API als auch AWS Secrets Manager autorisierte Benutzer. Wenn Sie nicht für alle in einer Richtlinie definierten Aktionen Berechtigungen besitzen, erhalten Sie den Fehler `AccessDeniedException`.

Speichern von Datenbankanmeldedaten in AWS Secrets Manager

Wenn Sie die Daten-API von Amazon RDS (Daten-API) aufrufen, können Sie die Anmeldeinformationen für die Datenbank unter Verwendung eines Geheimnisses in Secrets Manager übergeben. Zum Übermitteln der Anmeldeinformationen auf diese Weise geben Sie den Namen des Secrets oder den Amazon-Ressourcennamen (ARN) des Secrets an.

So speichern Sie DB-Cluster-Anmeldeinformationen in einem Secret

1. Sie können in Secrets Manager einen geheimen Schlüssel erstellen, der Anmeldeinformationen für den Aurora-DB-Cluster enthält.

   Anweisungen finden Sie unter Erstellen eines Datenbank-Secrets im AWS Secrets Manager - Benutzerhandbuch.

2. Verwenden Sie die Secrets Manager Manager-Konsole, um die Details für das von Ihnen erstellte Geheimnis anzuzeigen, oder führen Sie den `aws secretsmanager describe-secret` AWS CLI Befehl aus.

   Notieren Sie sich den Namen und den ARN des Secrets. Sie können sie in Aufrufen an die Daten-API verwenden.

Weitere Informationen zur Verwendung von Secrets Manager finden Sie im AWS Secrets-Manager-Benutzerhandbuch.

Informationen dazu, wie Amazon Aurora die Identitäts- und Zugriffsverwaltung steuert, finden Sie unter So funktioniert Amazon Aurora mit IAM.

Informationen zum Erstellen einer IAM-Richtlinie finden Sie unter Erstellen von IAM-Richtlinien im IAM-Benutzerhandbuch. Informationen zum Hinzufügen einer IAM-Richtlinie zu einem Benutzer finden Sie im Abschnitt Hinzufügen und Entfernen von IAM-Identitätsberechtigungen im IAM-Benutzerhandbuch.

Aktivieren der Amazon-RDS-Daten-API

Um die Amazon-RDS-Daten-API (Daten-API) nutzen zu können, müssen Sie sie für Ihren Aurora-DB-Cluster aktivieren. Sie können die Daten-API aktivieren, wenn Sie den DB-Cluster erstellen oder ändern.

> **Note**
>
> Ob die Daten-API für Ihren Cluster verfügbar ist, hängt von Ihrer Aurora-Version, Datenbank-Engine und AWS-Region ab. Für ältere Aurora-Versionen funktioniert die Daten-API nur mit Aurora Serverless v1-Clustern. Bei neueren Aurora-Versionen funktioniert die Daten-API mit Clustern, die sowohl bereitgestellte als auch Aurora Serverless v2-Instances

verwenden. Informationen dazu, ob Ihr Cluster die Daten-API verwenden kann, finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für RDS-Daten-API.

Themen

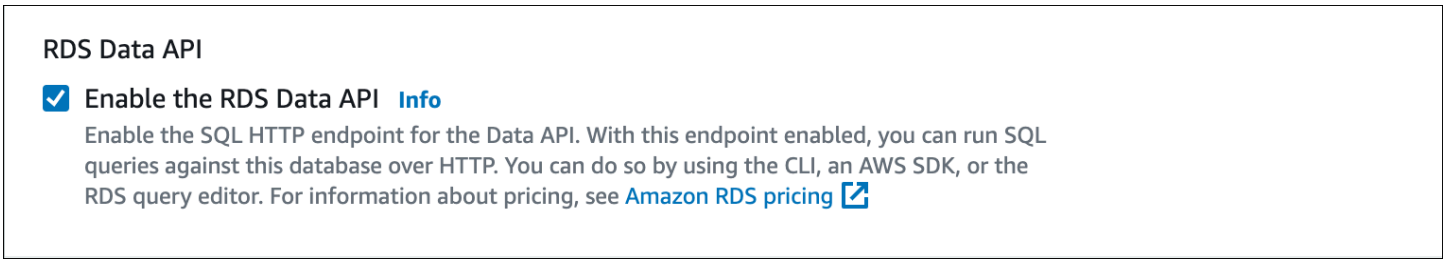
- Aktivieren der RDS-Daten-API beim Erstellen einer Datenbank
- Aktivieren der RDS-Daten-API in einer vorhandenen Datenbank

Aktivieren der RDS-Daten-API beim Erstellen einer Datenbank

Während Sie eine Datenbank erstellen, die die RDS-Daten-API (Daten-API) unterstützt, können Sie dieses Feature aktivieren. In den folgenden Verfahren wird beschrieben, wie Sie vorgehen müssen, wenn Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API verwenden.

Konsole

Um die Daten-API bei der Erstellung eines DB-Clusters zu aktivieren, wählen Sie das Kontrollkästchen RDS-Daten-API aktivieren im Bereich Konnektivität der Seite Datenbank erstellen aus, wie im folgenden Screenshot gezeigt.



Anleitungen zum Erstellen eines Aurora-DB-Clusters, der die RDS-Daten-API nutzen kann, finden Sie unten:

- Für Aurora Serverless v2 und bereitgestellte Cluster – Erstellen eines Amazon Aurora-DB Clusters
- Für Aurora Serverless v1 – Erstellen eines Aurora Serverless v1-DB Clusters

AWS CLI

Um die Daten-API zu aktivieren, während Sie einen Aurora-DB-Cluster erstellen, führen Sie den AWS CLI-Befehl create-db-cluster mit der Option `--enable-http-endpoint` aus.

Im folgenden Beispiel wird ein DB-Cluster von Aurora PostgreSQL mit aktivierter Daten-API erstellt.

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my_pg_cluster \
    --engine aurora-postgresql \
    --enable-http-endpoint
```

Für Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my_pg_cluster ^
    --engine aurora-postgresql ^
    --enable-http-endpoint
```

RDS-API

Um die Daten-API zu aktivieren, während Sie einen Aurora-DB-Cluster erstellen, verwenden Sie den Vorgang CreateDBCluster, wobei der Wert des Parameters `EnableHttpEndpoint` auf `true` gesetzt sein muss.

Aktivieren der RDS-Daten-API in einer vorhandenen Datenbank

Sie können einen DB-Cluster ändern, der die RDS-Daten-API (Daten-API) unterstützt, um dieses Feature zu aktivieren oder zu deaktivieren.

Themen

- Aktivieren oder Deaktivieren der Daten-API (Aurora Serverless v2 und bereitgestellt)
- Aktivieren oder Deaktivieren der Daten-API (nur Aurora Serverless v1)

Aktivieren oder Deaktivieren der Daten-API (Aurora Serverless v2 und bereitgestellt)

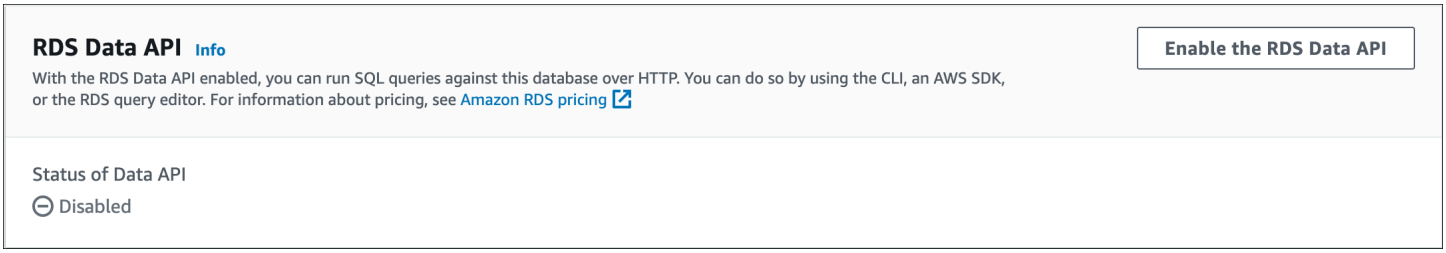
Verwenden Sie die folgenden Verfahren, um die Daten-API in Aurora Serverless v2 und bereitgestellten Datenbanken zu aktivieren oder zu deaktivieren. Verwenden Sie die Verfahren unter the section called “Aktivieren oder Deaktivieren der Daten-API (nur Aurora Serverless v1)”, um die Daten-API in Aurora Serverless v1-Datenbanken zu aktivieren oder zu deaktivieren.

Konsole

Sie können die Daten-API aktivieren oder deaktivieren, indem Sie die RDS-Konsole für einen DB-Cluster verwenden, der dieses Feature unterstützt. Öffnen Sie dazu die Cluster-Detailseite der

Datenbank, für die Sie die Daten-API aktivieren oder deaktivieren möchten, und wechseln Sie auf der Registerkarte Konnektivität und Sicherheit zum Abschnitt RDS-Daten-API. In diesem Abschnitt wird der Status der Daten-API angezeigt und Sie können sie aktivieren oder deaktivieren.

Der folgende Screenshot zeigt, dass die RDS-Daten-API nicht aktiviert ist.



AWS CLI

Um die Daten-API in einer vorhandenen Datenbank zu aktivieren oder zu deaktivieren, führen Sie den AWS CLI-Befehl enable-http-endpoint oder disable-http-endpoint aus und geben Sie den ARN Ihres DB-Clusters an.

Das folgende Beispiel aktiviert die Daten-API.

Für Linux, macOS oder Unix:

```
aws rds enable-http-endpoint \
    --resource-arn cluster_arn
```

Für Windows:

```
aws rds enable-http-endpoint ^
    --resource-arn cluster_arn
```

RDS-API

Verwenden Sie die Operationen EnableHttpEndpoint und DisableHttpEndpoint, um die Daten-API in einer vorhandenen Datenbank zu aktivieren oder zu deaktivieren.

Aktivieren oder Deaktivieren der Daten-API (nur Aurora Serverless v1)

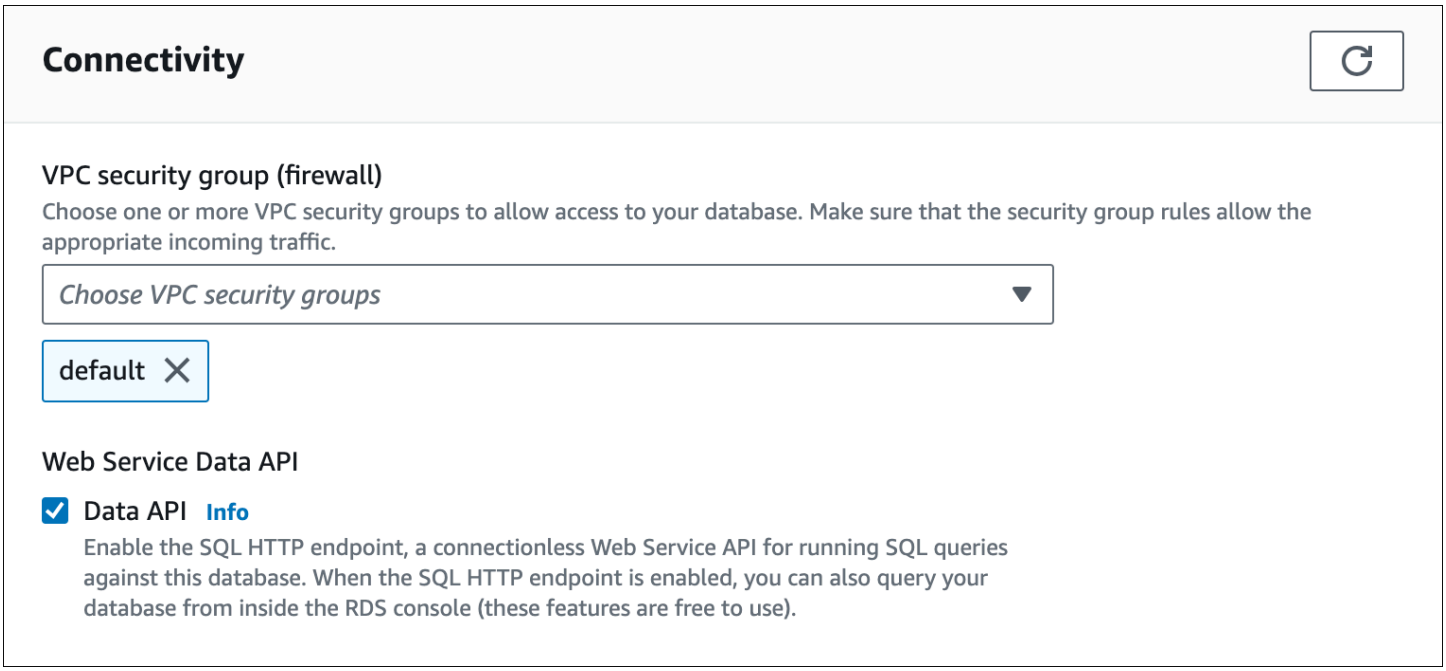
Verwenden Sie die folgenden Verfahren, um die Daten-API in vorhandenen Aurora Serverless v1-Datenbanken zu aktivieren oder zu deaktivieren. Verwenden Sie die Verfahren unter the section

called “Aktivieren oder Deaktivieren der Daten-API”, um die Daten-API für Aurora Serverless v2- und bereitgestellte Datenbanken zu aktivieren oder zu deaktivieren.

Konsole

Wenn Sie einen Aurora Serverless v1-DB-Cluster bearbeiten, aktivieren Sie die Daten-API im Abschnitt Konnektivität der RDS-Konsole.

Der folgende Screenshot zeigt die aktivierte Daten-API beim Ändern eines Aurora-DB-Clusters.



Anleitungen zum Ändern eines Aurora Serverless v1-DB-Clusters finden Sie unter Ändern eines Aurora Serverless v1-DB-Clusters.

AWS CLI

Um die Daten-API zu aktivieren oder zu deaktivieren, führen Sie den AWS CLI-Befehl modify-db-cluster mit `--enable-http-endpoint` oder `--no-enable-http-endpoint` aus.

Im folgenden Beispiel wird die Daten-API in einem `sample-cluster` aktiviert.

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --enable-http-endpoint
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --enable-http-endpoint
```

RDS-API

Um die Daten-API zu aktivieren, verwenden Sie den Vorgang ModifyDBCluster und setzen Sie den Wert von `EnableHttpEndpoint` je nach Bedarf auf `true` oder `false`.

Erstellen eines Amazon-VPC-Endpunkts für die Amazon-RDS-Daten-API (AWS PrivateLink)

Amazon VPC ermöglicht Ihnen das Starten von AWS-Ressourcen wie Aurora-DB-Clustern und Anwendungen in einer virtuellen privaten Cloud (VPC). AWS PrivateLink stellt eine hoch sichere, private Konnektivität zwischen VPCs und AWS-Services im Amazon-Netzwerk bereit. Mit AWS PrivateLink können Sie Amazon-VPC-Endpunkte erstellen, mit denen Sie eine Verbindung zu Services über verschiedene Konten und VPCs basierend auf Amazon VPC herstellen können. Weitere Informationen über AWS PrivateLink finden Sie unter VPC-Endpunktservices (AWS PrivateLink) im Amazon-Virtual-Private-Cloud-Benutzerhandbuch.

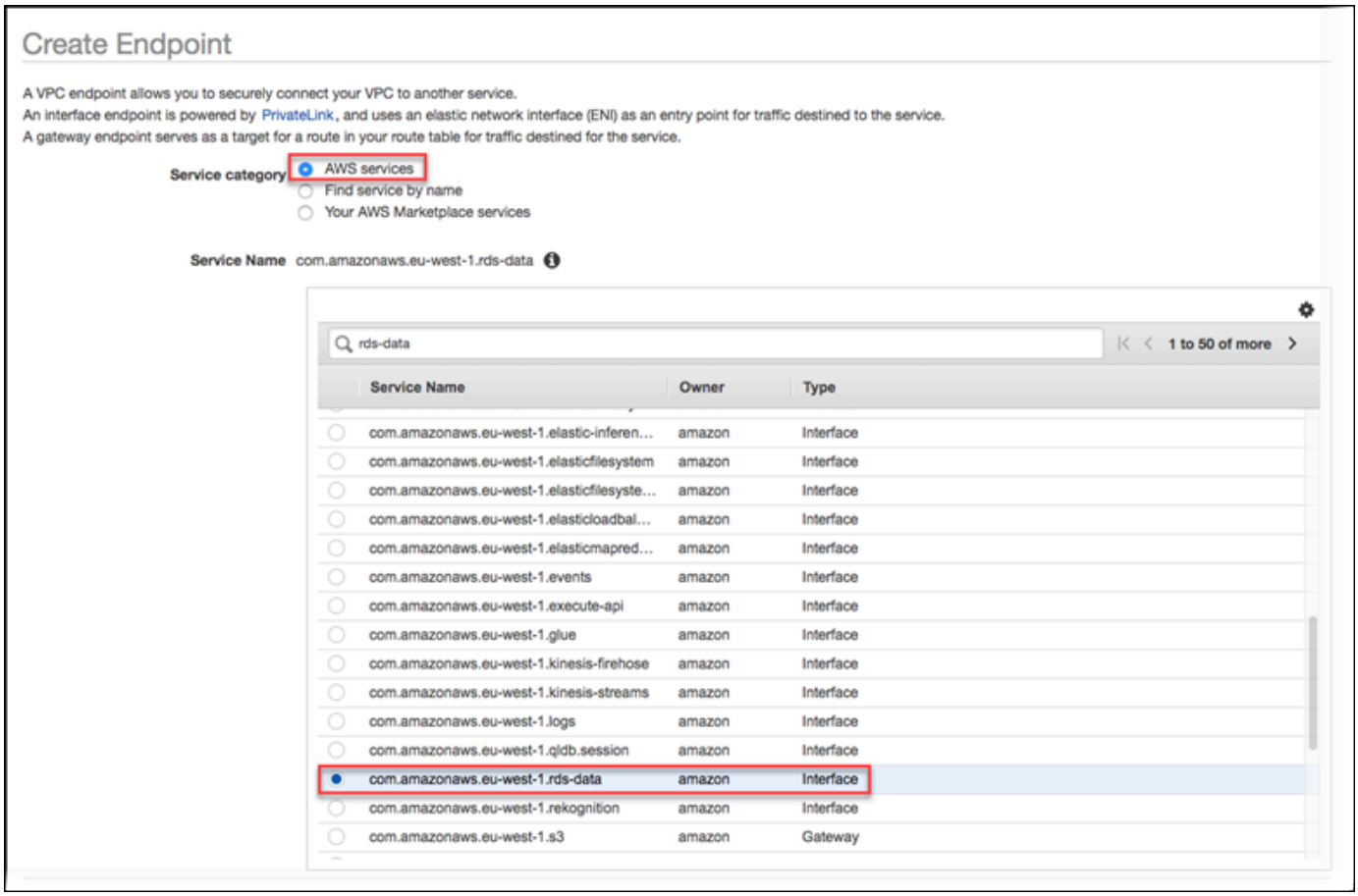
Sie können die RDS-Daten-API (Daten-API) mit Amazon-VPC-Endpunkten aufrufen. Die Verwendung eines Amazon-VPC-Endpunkts hält den Datenverkehr zwischen Anwendungen in Ihrer Amazon VPC und der Daten-API im AWS-Netzwerk aufrecht, ohne öffentliche IP-Adressen zu verwenden. Amazon-VPC-Endpunkte können Ihnen dabei helfen, Compliance- und behördliche Anforderungen im Zusammenhang mit der Einschränkung der öffentlichen Internetkonnektivität zu erfüllen. Wenn Sie beispielsweise einen Amazon-VPC-Endpunkt verwenden, können Sie den Datenverkehr zwischen einer Anwendung, die auf einer Amazon-EC2-Instance ausgeführt wird, und der Daten-API in den VPCs, die diese enthalten, beibehalten.

Nachdem Sie den Amazon VPC-Endpunkt erstellt haben, können Sie ihn verwenden, ohne Code- oder Konfigurationsänderungen in der Anwendung vorzunehmen.

So erstellen Sie einen Amazon-VPC-Endpunkt für die Daten-API

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie Endpunkte und dann Endpunkt erstellen aus.

3. Wählen Sie auf der Seite Create Endpoint (Endpunkt erstellen) für Service category (Servicekategorie) die Option AWS-Services aus. Wählen Sie für Servicename rds-data aus.



4. Wählen Sie für VPC die VPC aus, in der der Endpunkt erstellt werden soll.

   Wählen Sie die VPC aus, die die Anwendung enthält, die Daten-API-Aufrufe ausführt.

5. Wählen Sie für Subnetze das Subnetz für jede Availability Zone (AZ) aus, die vom AWS-Service verwendet wird, auf dem Ihre Anwendung ausgeführt wird.

   Um einen Amazon VPC-Endpunkt zu erstellen, geben Sie den privaten IP-Adressbereich an, in dem auf den Endpunkt zugegriffen werden soll. Wählen Sie dazu das Subnetz für jede

Availability Zone aus. Dadurch wird der VPC-Endpunkt auf den privaten IP-Adressbereich beschränkt, der für jede Availability Zone spezifisch ist. Außerdem wird in jeder Availability Zone ein Amazon VPC-Endpunkt erstellt.

6. Wählen Sie für DNS-Namen aktivieren die Option Für diesen Endpunkt aktivieren aus.

   Private DNS löst den standardmäßigen DNS-Hostnamen der Daten-API (`https://rds-data.`*`region`*`.amazonaws.com`) in die privaten IP-Adressen auf, die mit dem für Ihren Amazon VPC-Endpunkt spezifischen DNS-Hostnamen verknüpft sind. Daher können Sie mit der AWS CLI oder den AWS-SDKs auf den VPC-Endpunkt der Daten-API zugreifen, ohne Code- oder Konfigurationsänderungen zum Aktualisieren der Daten-API-Endpunkt-URL vorzunehmen.

7. Wählen Sie für Sicherheitsgruppe eine Sicherheitsgruppe aus, die dem Amazon VPC-Endpunkt zugeordnet werden soll.

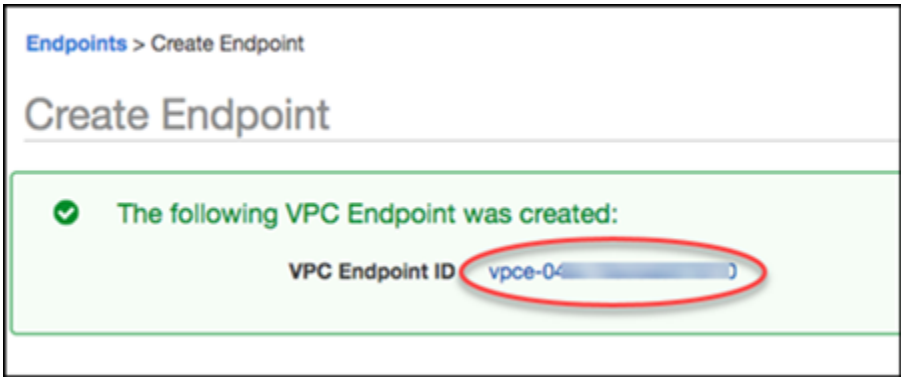
   Wählen Sie die Sicherheitsgruppe aus, die den Zugriff auf den AWS-Service ermöglicht, auf dem Ihre Anwendung ausgeführt wird. Wenn beispielsweise eine Amazon EC2-Instance Ihre Anwendung ausführt, wählen Sie die Sicherheitsgruppe aus, die den Zugriff auf die Amazon EC2-Instance ermöglicht. Mit der Sicherheitsgruppe können Sie den Datenverkehr zum Amazon VPC-Endpunkt von Ressourcen in Ihrer VPC steuern.

8. Wählen Sie unter Richtlinie die Option Voller Zugriff aus, damit jeder innerhalb der Amazon VPC über diesen Endpunkt auf die Daten-API zugreifen kann. Oder wählen Sie Benutzerdefiniert aus, um eine Richtlinie anzugeben, die den Zugriff einschränkt.

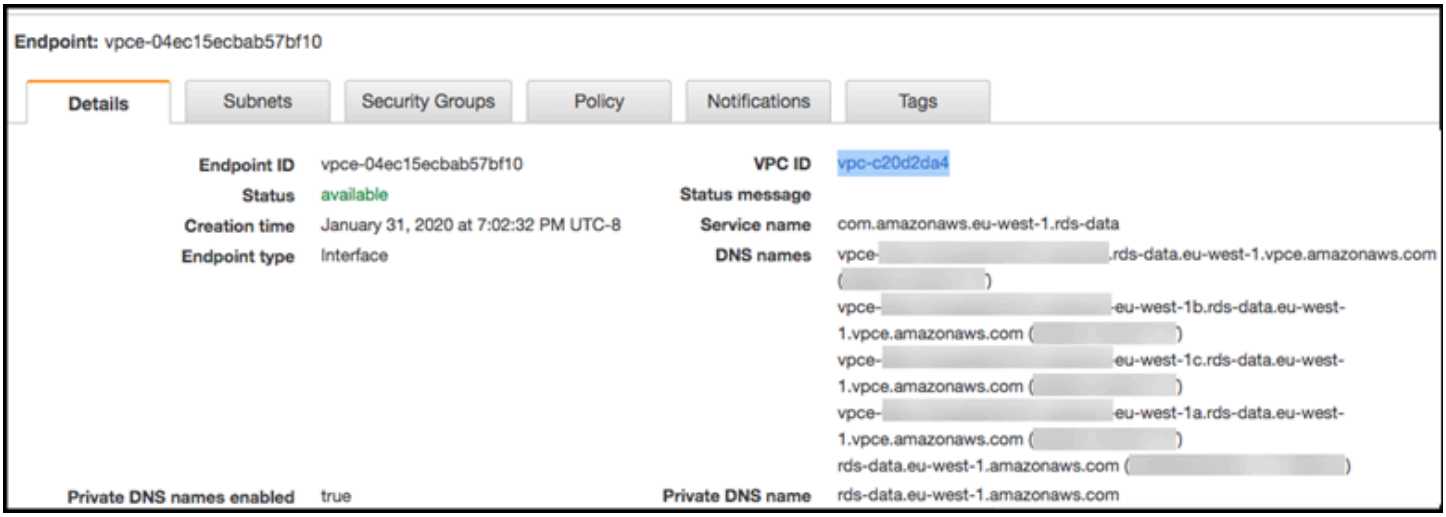
   Wenn Sie Benutzerdefiniert auswählen, geben Sie die Richtlinie im Tool zur Richtlinienerstellung ein.

9. Wählen Sie Endpunkt erstellen aus.

Nachdem der Endpunkt erstellt wurde, wählen Sie den Link in der AWS-Managementkonsole aus, um die Endpunktdetails anzuzeigen.



Auf der Registerkarte Details des Endpunkts werden die DNS-Hostnamen angezeigt, die beim Erstellen des Amazon VPC-Endpunkts generiert wurden.



Sie können den Standardendpunkt (`rds-data.`*`region`*`.amazonaws.com`) oder einen der VPC-spezifischen Endpunkte verwenden, um die Daten-API innerhalb der Amazon VPC aufzurufen. Der standardmäßige Daten-API-Endpunkt leitet automatisch an den Amazon VPC-Endpunkt weiter. Dieses Routing tritt auf, weil der private DNS-Hostname beim Erstellen des Amazon VPC-Endpunkts aktiviert wurde.

Wenn Sie einen Amazon-VPC-Endpunkt in einem Daten-API-Aufruf verwenden, verbleibt der gesamte Datenverkehr zwischen Ihrer Anwendung und der Daten-API in den Amazon VPCs, die sie enthalten. Sie können einen Amazon VPC-Endpunkt für jeden Typ von Daten-API-Aufruf verwenden. Informationen zum Aufrufen der Daten-API finden Sie unter Aufrufen der Daten-API von Amazon RDS.

Aufrufen der Daten-API von Amazon RDS

Wenn die Daten-API von Amazon RDS (Daten-API) auf Ihrem Aurora-DB-Cluster aktiviert ist, können Sie SQL-Anweisungen auf dem Aurora-DB-Cluster ausführen, indem Sie die Daten-API oder die AWS CLI verwenden. Die Daten-API unterstützt die von den AWS SDKs unterstützten Programmiersprachen. Weitere Informationen finden Sie unter Tools für AWS.

Themen

- Referenz zu Operationen mit der Amazon-RDS-Daten-API
- Aufrufen der Daten-API von Amazon RDS über die AWS CLI
- Aufrufen der Daten-API von Amazon RDS aus einer Python-Anwendung
- Aufrufen der Daten-API von Amazon RDS aus einer Java-Anwendung
- Steuern des Timeout-Verhaltens der Daten-API

Referenz zu Operationen mit der Amazon-RDS-Daten-API

Die Amazon-RDS-Daten-API bietet die folgenden Operationen zur Ausführung von SQL-Anweisungen.

Daten-API-Operation	AWS CLI command	Beschreibung
ExecuteStatement	aws rds-data execute-statement	Führt eine SQL-Anweisung in einer Datenbank aus.
BatchExecuteStatement	aws rds-data batch-execute-statement	Führt eine Batch-SQL-Anweisung über ein Array von Daten für Massen-Update- und -Einfügeoperationen aus. Sie können eine DML-Anweisung (Data Manipulation Language) mit einem Array von Parametersätzen ausführen . Eine Batch-SQL-Anweisung kann gegenüber einzelnen Einfügungs- und Aktualisierungsanweisungen eine erhebliche Leistungsverbesserung bieten.

Sie können beide Vorgänge verwenden, um einzelne SQL-Anweisungen oder Transaktionen auszuführen. Für Transaktionen bietet die Daten-API die folgenden Operationen.

Daten-API-Operation	AWS CLI command	Beschreibung
BeginTransaction	aws rds-data begin-transaction	Startet eine SQL-Transaktion.
CommitTransaction	aws rds-data commit-transaction	Beendet eine SQL-Transaktion und schreibt die Änderungen fest.
RollbackTransaction	aws rds-data rollback-transaction	Führt ein Rollback einer Transaktion durch.

Für die Operationen zur Ausführung von SQL-Anweisungen und zur Unterstützung von Transaktionen werden die folgenden allgemeinen Daten-API-Parameter und AWS CLI-Optionen verwendet. Einige Operationen unterstützen andere Parameter oder Optionen.

Daten-API-Operationsparameter	AWS CLI-Befehlsoption	Erforderlich	Beschreibung
`resourceArn`	`--resource-arn`	Ja	Der Amazon-Ressourcenname (ARN) für den Aurora-DB-Cluster. Der Cluster muss sich im selben AWS-Konto befinden wie die IAM-Rolle oder der Benutzer, der die Daten-API aufruft. Um auf einen Cluster in einem anderen Konto zuzugreifen, nehmen Sie eine Rolle in diesem Konto an.
`secretArn`	`--secret-arn`	Ja	Der Name oder ARN des Secrets, das Zugriff auf das DB-Cluster ermöglicht.

Die RDS-Daten-API unterstützt die folgenden Datentypen für Aurora MySQL:

- `TINYINT(1)`, `BOOLEAN`, `BOOL`
- `TINYINT`
- `SMALLINT` [`SIGNED` | `UNSIGNED`]
- `MEDIUMINT` [`SIGNED` | `UNSIGNED`]
- `INT` [`SIGNED` | `UNSIGNED`]
- `BIGINT` [`SIGNED` | `UNSIGNED`]
- `FLOAT`
- `DOUBLE`
- `VARCHAR`, `CHAR`, `TEXT`, `ENUM`
- `VARBINARY`, `BINARY`, `BLOB`
- `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`
- `DECIMAL`
- `JSON`
- `BIT`, `BIT(N)`

Die RDS-Daten-API unterstützt die folgenden skalaren Typen für Aurora-PostgreSQL:

- `BOOL`
- `BYTEA`
- `DATE`
- `CIDR`
- `DECIMAL`, `NUMERIC`
- `ENUM`
- `FLOAT8`, `DOUBLE PRECISION`
- `INET`
- `INT`, `INT4`, `SERIAL`
- `INT2`, `SMALLINT`, `SMALLSERIAL`
- `INT8`, `BIGINT`, `BIGSERIAL`
- `JSONB`, `JSON`
- `REAL`, `FLOAT`

- TEXT, CHAR(N), VARCHAR, NAME
- TIME
- TIMESTAMP
- UUID
- VECTOR

Die RDS-Daten-API unterstützt die folgenden Array-Typen für Aurora PostgreSQL:

- BOOL[], BIT[]
- DATE[]
- DECIMAL[], NUMERIC[]
- FLOAT8[], DOUBLE PRECISION[]
- INT[], INT4[]
- INT2[]
- INT8[], BIGINT[]
- JSON[]
- REAL[], FLOAT[]
- TEXT[], CHAR(N)[], VARCHAR[], NAME[]
- TIME[]
- TIMESTAMP[]
- UUID[]

Sie können Parameter in Daten-API-Aufrufen von `ExecuteStatement` und `BatchExecuteStatement`, sowie bei Ausführung der AWS CLI-Befehle `execute-statement` und `batch-execute-statement` verwenden. Um einen Parameter zu verwenden, geben Sie ein Name-Wert-Paar im Datentyp `SqlParameter` an. Den Wert geben Sie mit dem Datentyp `Field` an. In der folgenden Tabelle sind den Datentypen, die Sie in Daten-API-Aufrufen angeben, JDBC-Datentypen (Java Database Connectivity) zugeordnet.

JDBC-Datentyp	Daten-API-Datentyp
INTEGER, TINYINT, SMALLINT, BIGINT	LONG (oder STRING)

JDBC-Datentyp	Daten-API-Datentyp
`FLOAT, REAL, DOUBLE`	`DOUBLE`
`DECIMAL`	`STRING`
`BOOLEAN, BIT`	`BOOLEAN`
`BLOB, BINARY, LONGVARBINARY, VARBINARY`	`BLOB`
`CLOB`	`STRING`
Andere Typen (einschließlich datums- und zeitbezogener Typen)	`STRING`

> **Note**
>
> Sie können den Datentyp `LONG` oder `STRING` in Ihrem Daten-API-Aufruf für von der Datenbank zurückgegebene `LONG`-Werte angeben. Wir empfehlen Ihnen diese Vorgehensweise, um einen Präzisionsverlust bei extrem großen Zahlen zu vermeiden, was bei der Arbeit mit JavaScript vorkommen kann.

Bestimmte Typen, wie `DECIMAL` und `TIME`, erfordern einen Hinweis, damit die Daten-API `String`-Werte als richtigen Typ an die Datenbank weiterleitet. Um einen Hinweis zu verwenden, schließen Sie Werte für `typeHint` in den Datentyp `SqlParameter` ein. Die folgenden Werte sind für `typeHint` möglich:

- `DATE` – Der entsprechende `String`-Parameter wird als Objekt des Typs `DATE` an die Datenbank gesendet. Das akzeptierte Format ist `YYYY-MM-DD`.
- `DECIMAL` – Der entsprechende `String`-Parameter wird als Objekt des Typs `DECIMAL` an die Datenbank gesendet.
- `JSON` – Der entsprechende `String`-Parameter wird als Objekt des Typs `JSON` an die Datenbank gesendet.
- `TIME` – Der entsprechende `String`-Parameter wird als Objekt des Typs `TIME` an die Datenbank gesendet. Das akzeptierte Format ist `HH:MM:SS[.FFF]`.

- `TIMESTAMP` – Der entsprechende `String`-Parameter wird als Objekt des Typs `TIMESTAMP` an die Datenbank gesendet. Das akzeptierte Format ist `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `UUID` – Der entsprechende `String`-Parameter wird als Objekt des Typs `UUID` an die Datenbank gesendet.

> **Note**
>
> Derzeit unterstützt die Daten-API keine Arrays von Universal Unique Identifiers (UUIDs).

> **Note**
>
> Für Amazon Aurora PostgreSQL gibt die Daten-API immer den Aurora PostgreSQL Datentyp `TIMESTAMPTZ` in der UTC-Zeitzone zurück.

Aufrufen der Daten-API von Amazon RDS über die AWS CLI

Sie können die RDS-Daten-API (Daten-API) über die AWS CLI aufrufen.

In den folgenden Beispielen wird die AWS CLI für die Daten-API verwendet. Weitere Informationen finden Sie in der AWS CLI-Referenz für die Daten-API.

Ersetzen Sie in jedem Beispiel den Amazon-Ressourcennamen (ARN) des DB-Clusters durch den ARN Ihres Aurora-DB-Clusters. Ersetzen Sie außerdem den geheimen ARN durch den ARN des geheimen Schlüssels in Secrets Manager, der den Zugriff auf den DB-Cluster ermöglicht.

> **Note**
>
> Die AWS CLI kann Antworten in JSON formatieren.

Themen

- Starten einer SQL-Transaktion
- Ausführen einer SQL-Anweisung
- Ausführen einer Stapel-SQL-Anweisung über ein Daten-Array
- Übergeben einer SQL-Transaktion

- Rollback einer SQL-Transaktion

Starten einer SQL-Transaktion

Sie können eine SQL-Transaktion mit dem CLI-Befehl `aws rds-data begin-transaction` starten. Der Aufruf gibt eine Transaktions-ID zurück.

> **Important**
>
> Auf der Daten-API kommt es bei einer Transaktion zu einer Zeitüberschreitung, wenn es innerhalb von drei Minuten keine Aufrufe gibt, die ihre Transaktions-ID verwenden. Wenn es zu einer Zeitüberschreitung kommt, bevor die Transaktion festgeschrieben wird, setzt die Daten-API die Transaktion automatisch zurück.
> MySQL Data Definition Language (DDL)-Anweisungen innerhalb einer Transaktion führen zu einem impliziten Commit. Wir empfehlen Ihnen, die einzelnen MySQL-DDL-Anweisungen jeweils in einem separaten Befehl `execute-statement` mit der Option `--continue-after-timeout` auszuführen.

Geben Sie zusätzlich zu den allgemeinen Optionen die Option `--database` an, die den Namen der Datenbank enthält.

Der folgende CLI-Befehl beispielsweise startet eine SQL-Transaktion.

Für Linux, macOS oder Unix:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

Für Windows:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

Im Folgenden sehen Sie ein Beispiel für die Antwort.

```
{
    "transactionId": "ABC1234567890xyz"
}
```

Ausführen einer SQL-Anweisung

Sie können mittels des CLI-Befehls `aws rds-data execute-statement` eine SQL-Anweisung ausführen.

Sie können die SQL-Anweisung in einer Transaktion ausführen, indem Sie die Transaktions-ID mit der Option `--transaction-id` angeben. Sie können eine SQL-Transaktion mit dem CLI-Befehl `aws rds-data begin-transaction` starten. Sie können eine Transaktion beenden und übergeben, indem Sie den CLI-Befehl `aws rds-data commit-transaction` verwenden.

> **Important**
>
> Wenn Sie die Option `--transaction-id` nicht angeben, werden Änderungen, die sich durch den Aufruf ergeben, automatisch übergeben.

Geben Sie zusätzlich zu den allgemeinen Optionen die folgenden Optionen an:

- `--sql` (erforderlich) – eine SQL-Anweisung, die auf dem DB-Cluster ausgeführt werden soll.
- `--transaction-id` (optional) – die ID einer Transaktion, die über den CLI-Befehl `begin-transaction` gestartet wurde. Geben Sie die Transaktions-ID der Transaktion an, in die Sie die SQL-Anweisung einfügen möchten.
- `--parameters` (optional) – die Parameter für die SQL-Anweisung.
- `--include-result-metadata | --no-include-result-metadata` (optional) – ein Wert, der angibt, ob Metadaten in die Ergebnisse eingefügt werden sollen. Der Standardwert ist `--no-include-result-metadata`.
- `--database` (optional) – der Name der Datenbank.

  Die `--database`-Option funktioniert möglicherweise nicht, wenn Sie eine SQL-Anweisung ausführen, nachdem Sie bei der vorherige Anfrage `--sql "use database_name;"` ausgeführt haben. Es wird empfohlen, die `--database`-Option zu verwenden statt `--sql "use database_name;"`-Anweisungen auszuführen.

- `--continue-after-timeout | --no-continue-after-timeout` (optional) – ein Wert, der angibt, ob die Anweisung weiter ausgeführt werden soll, wenn der Aufruf das Daten-API-Zeitintervall von 45 Sekunden überschreitet. Der Standardwert ist `--no-continue-after-timeout`.

  Im Fall von Data Definition Language (DDL)-Anweisungen sollten Sie die Anweisung nach Ablauf des Aufrufs weiter ausführen, um Fehler und die Möglichkeit beschädigter Datenstrukturen zu vermeiden.
- `--format-records-as "JSON"|"NONE"` – Ein optionaler Wert, der angibt, ob die Ergebnismenge als JSON-Zeichenfolge formatiert werden soll. Der Standardwert ist `"NONE"`. Nutzungsinformationen über die Verarbeitung von JSON-Ergebnismengen finden Sie unter Verarbeiten von Abfrageergebnissen der Amazon-RDS-Data-API im JSON-Format.

Das DB-Cluster gibt für den Aufruf eine Antwort zurück.

> **Note**
>
> Das Limit für Antwortgrößen beträgt 1 MiB. Wenn der Aufruf mehr als 1 MiB an Antwortdaten zurückgibt, wird der Aufruf beendet.
> Für Aurora Serverless v1 beträgt die maximale Anzahl der Anforderungen pro Sekunde 1 000. Für alle anderen unterstützten Datenbanken gibt es kein Limit.

Der folgende CLI-Befehl führt beispielsweise eine einzelne SQL-Anweisung aus und lässt die Metadaten in den Ergebnissen weg (Standard).

Für Linux, macOS oder Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

Für Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

Im Folgenden sehen Sie ein Beispiel für die Antwort.

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            },
            {
                "stringValue": "ValueOne"
            }
        ],
        [
            {
                "longValue": 2
            },
            {
                "stringValue": "ValueTwo"
            }
        ],
        [
            {
                "longValue": 3
            },
            {
                "stringValue": "ValueThree"
            }
        ]
    ]
}
```

Der folgende CLI-Befehl führt eine einzelne SQL-Anweisung in einer Transaktion aus, indem die Option `--transaction-id` angegeben wird.

Für Linux, macOS oder Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Für Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Im Folgenden sehen Sie ein Beispiel für die Antwort.

```
{
    "numberOfRecordsUpdated": 1
}
```

Der folgende CLI-Befehl führt eine einzelne SQL-Anweisung mit Parametern aus.

Für Linux, macOS oder Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Für Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Im Folgenden sehen Sie ein Beispiel für die Antwort.

```
{
    "numberOfRecordsUpdated": 1
}
```

Der folgende CLI-Befehl führt eine Data Definition Language (DDL)-SQL-Anweisung aus. Die DDL-Anweisung benennt die Spalte `job` in die Spalte `role` um.

> **Important**
>
> Im Fall von DDL-Anweisungen sollten Sie die Anweisung auch nach Ablauf des Aufrufs weiter ausführen. Wenn eine DDL-Anweisung vor Ende der Ausführung beendet wird, kann dies zu Fehlern und möglicherweise beschädigten Datenstrukturen führen. Um eine Anweisung weiter auszuführen, nachdem ein Aufruf das für die RDS-Daten-API geltende Zeitintervall von 45 Sekunden überschritten hat, geben Sie die Option `--continue-after-timeout` an.

Für Linux, macOS oder Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Für Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Im Folgenden sehen Sie ein Beispiel für die Antwort.

```
{
    "generatedFields": [],
    "numberOfRecordsUpdated": 0
}
```

> **Note**
>
> Die `generatedFields`-Daten werden von Aurora PostgreSQL nicht unterstützt. Zum Abrufen der Werte von generierten Feldern verwenden Sie die `RETURNING`-Klausel. Weitere Informationen finden Sie in unter Returning Data From Modified Rows in der PostgreSQL-Dokumentation.

Ausführen einer Stapel-SQL-Anweisung über ein Daten-Array

Sie können eine Batch-SQL-Anweisung über ein Daten-Array ausführen, indem Sie den CLI-Befehl `aws rds-data batch-execute-statement` verwenden. Sie können dieses Befehl verwenden, um einen Massenimport oder eine Update-Operation auszuführen.

Sie können die SQL-Anweisung in einer Transaktion ausführen, indem Sie die Transaktions-ID mit der Option `--transaction-id` angeben. Sie können eine SQL-Transaktion mit dem CLI-Befehl `aws rds-data begin-transaction` starten. Sie können eine Transaktion mit dem CLI-Befehl `aws rds-data commit-transaction` beenden und übergeben.

> **Important**
>
> Wenn Sie die Option `--transaction-id` nicht angeben, werden Änderungen, die sich durch den Aufruf ergeben, automatisch übergeben.

Geben Sie zusätzlich zu den allgemeinen Optionen die folgenden Optionen an:

- `--sql` (erforderlich) – eine SQL-Anweisung, die auf dem DB-Cluster ausgeführt werden soll.

  > **Tip**
  >
  > Fügen Sie bei MySQL-kompatiblen Anweisungen kein Semikolon am Ende des `--sql`-Parameters ein. Ein abschließendes Semikolon kann einen Syntaxfehler verursachen.

- `--transaction-id` (optional) – die ID einer Transaktion, die über den CLI-Befehl `begin-transaction` gestartet wurde. Geben Sie die Transaktions-ID der Transaktion an, in die Sie die SQL-Anweisung einfügen möchten.
- `--parameter-set` (optional) – die Parametersätze für die Batch-Operation.

- `--database` (optional) – der Name der Datenbank.

Das DB-Cluster gibt für den Aufruf eine Antwort zurück.

> **Note**
>
> Es gibt keine feste Obergrenze für die Anzahl der Parametersätze. Die maximale Größe der HTTP-Anforderung, die über die Daten-API gesendet wird, beträgt jedoch 4 MiB. Wenn die Anforderung dieses Limit überschreitet, gibt die Daten-API einen Fehler zurück und die Anfrage wird nicht verarbeitet. Dieses 4-MiB-Limit umfasst die Größe der HTTP-Header und der JSON-Notation in der Anforderung. Die Anzahl der Parametersätze, die Sie einbinden können, hängt demnach von mehreren Faktoren ab, z. B. von der Größe der SQL-Anweisung und der Größe der individuellen Parametersätze.
> Das Limit für Antwortgrößen beträgt 1 MiB. Wenn der Aufruf mehr als 1 MiB an Antwortdaten zurückgibt, wird der Aufruf beendet.
> Für Aurora Serverless v1 beträgt die maximale Anzahl der Anforderungen pro Sekunde 1 000. Für alle anderen unterstützten Datenbanken gibt es kein Limit.

Der folgende CLI-Befehl führt beispielsweise eine Batch-SQL-Anweisung für ein Daten-Array mit einem Parametersatz aus.

Für Linux, macOS oder Unix:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

Für Windows:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

> **Note**
>
> Verwenden Sie in der Option `--parameter-sets` keine Zeilenumbrüche.

Übergeben einer SQL-Transaktion

Mit dem CLI-Befehl `aws rds-data commit-transaction` können Sie eine SQL-Transaktion beenden, die mit `aws rds-data begin-transaction` gestartet wurde, und die Änderungen übergeben.

Geben Sie zusätzlich zu den allgemeinen Optionen die folgende Option an:

- `--transaction-id` (erforderlich) – die ID einer Transaktion, die über den CLI-Befehl `begin-transaction` gestartet wurde. Geben Sie die Transaktions-ID der Transaktion an, die Sie beenden und übergeben möchten.

Der folgende CLI-Befehl beendet beispielsweise eine SQL-Transaktion und übergibt die Änderungen.

Für Linux, macOS oder Unix:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Für Windows:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
```

```
--transaction-id "ABC1234567890xyz"
```

Im Folgenden sehen Sie ein Beispiel für die Antwort.

```
{
    "transactionStatus": "Transaction Committed"
}
```

Rollback einer SQL-Transaktion

Mit dem CLI-Befehl `aws rds-data rollback-transaction` können Sie einen Rollback für eine SQL-Transaktion ausführen, die mit `aws rds-data begin-transaction` gestartet wurde. Durch das Rollback einer Transaktion werden für sie ausgeführte Änderungen rückgängig gemacht.

> **Important**
>
> Wenn die Transaktions-ID abgelaufen ist, wurde automatisch ein Rollback für die Transaktion ausgeführt. In diesem Fall gibt ein `aws rds-data rollback-transaction`-Befehl, der die abgelaufene Transaktions-ID angibt, einen Fehler zurück.

Geben Sie zusätzlich zu den allgemeinen Optionen die folgende Option an:

- `--transaction-id` (erforderlich) – die ID einer Transaktion, die über den CLI-Befehl `begin-transaction` gestartet wurde. Geben Sie die Transaktions-ID der Transaktion an, für die Sie ein Rollback ausführen möchten.

Der folgende AWS CLI-Befehl führt beispielsweise einen Rollback für eine SQL-Transaktion aus.

Für Linux, macOS oder Unix:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Für Windows:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
```

```
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

Im Folgenden sehen Sie ein Beispiel für die Antwort.

```
{
    "transactionStatus": "Rollback Complete"
    }
```

Aufrufen der Daten-API von Amazon RDS aus einer Python-Anwendung

Sie können die Daten-API von Amazon RDS (Daten-API) aus einer Python-Anwendung aufrufen.

Die folgenden Beispiele verwenden das AWS SDK for Python (Boto). Weitere Informationen zu Boto finden Sie in der AWS SDK for Python (Boto 3)-Dokumentation.

Ersetzen Sie in jedem Beispiel den Amazon-Ressourcennamen (ARN) des DB-Clusters durch den ARN Ihres Aurora-DB-Clusters. Ersetzen Sie außerdem den geheimen ARN durch den ARN des geheimen Schlüssels in Secrets Manager, der den Zugriff auf den DB-Cluster ermöglicht.

Themen

- Ausführen einer SQL-Abfrage
- Ausführen einer DML SQL-Anweisung
- Ausführen einer SQL-Transaktion

Ausführen einer SQL-Abfrage

Sie können eine SELECT-Anweisung ausführen und die Ergebnisse mit einer Python-Anwendung abrufen.

Im folgenden Beispiel wird eine SQL-Abfrage ausgeführt.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

response1 = rdsData.execute_statement(
```

```
            resourceArn = cluster_arn,
            secretArn = secret_arn,
            database = 'mydb',
            sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
```

```
        },
        {
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

Ausführen einer DML SQL-Anweisung

Sie können eine Data Manipulation Language (DML)-Anweisung ausführen, um in Ihrer Datenbank Daten einzufügen, zu aktualisieren oder zu löschen. Sie können in DML-Anweisungen auch Parameter verwenden.

> **Important**
>
> Wenn ein Aufruf kein Teil einer Transaktion ist, da er den Parameter `transactionID` nicht enthält, werden Änderungen, die sich aus dem Aufruf ergeben, automatisch übergeben.

Im folgenden Beispiel werden eine SQL Insert-Anweisung ausgeführt und Parameter verwendet.

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

rdsData = boto3.client('rds-data')


param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
paramSet = [param1, param2]

response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
 VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

Ausführen einer SQL-Transaktion

Sie können eine SQL-Transaktion starten, eine oder mehrere SQL-Anweisungen ausführen und anschließend die Änderungen mit einer Python-Anwendung übergeben.

> **Important**
>
> Bei einer Transaktion kommt es zu einer Zeitüberschreitung, wenn es innerhalb von drei Minuten keine Aufrufe gibt, die ihre Transaktions-ID verwenden. Wenn es zu einer Zeitüberschreitung kommt, bevor die Transaktion festgeschrieben wird, wird die Transaktion automatisch zurückgesetzt.
> Wenn Sie keine Transaktions-ID angeben, werden Änderungen, die sich durch den Aufruf ergeben, automatisch übergeben.

Im folgenden Beispiel wird eine SQL-Transaktion ausgeführt, die eine Zeile in eine Tabelle einfügt.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb')

response3 = rdsData.execute_statement(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb',
     sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
     transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     transactionId = tr['transactionId'])

cr['transactionStatus']
```

```
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

> **Note**
>
> Wenn Sie eine DDL-Anweisung ausführen, sollten Sie die Anweisung auch nach Ablauf des Aufrufs weiter ausführen. Wenn eine DDL-Anweisung vor Ende der Ausführung beendet wird, kann dies zu Fehlern und möglicherweise beschädigten Datenstrukturen führen. Um eine Anweisung weiter auszuführen, nachdem ein Aufruf das für die RDS-Daten-API geltende Zeitintervall von 45 Sekunden überschritten hat, legen Sie den Parameter `continueAfterTimeout` auf `true` fest.

Aufrufen der Daten-API von Amazon RDS aus einer Java-Anwendung

Sie können die Daten-API von Amazon RDS (Daten-API) aus einer Java-Anwendung aufrufen.

Die folgenden Beispiele verwenden das AWS SDK for Java. Weitere Informationen finden Sie im AWS SDK für Java-Entwicklerhandbuch.

Ersetzen Sie in jedem Beispiel den Amazon-Ressourcennamen (ARN) des DB-Clusters durch den ARN Ihres Aurora-DB-Clusters. Ersetzen Sie außerdem den geheimen ARN durch den ARN des geheimen Schlüssels in Secrets Manager, der den Zugriff auf den DB-Cluster ermöglicht.

Themen

- Ausführen einer SQL-Abfrage
- Ausführen einer SQL-Transaktion
- Ausführen einer Stapel-SQL-Operation

Ausführen einer SQL-Abfrage

Sie können eine `SELECT`-Anweisung ausführen und die Ergebnisse mit einer Java-Anwendung abrufen.

Im folgenden Beispiel wird eine SQL-Abfrage ausgeführt.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

    ExecuteStatementResult result = rdsData.executeStatement(request);

    for (List<Field> fields: result.getRecords()) {
      String stringValue = fields.get(0).getStringValue();
      long numberValue = fields.get(1).getLongValue();

      System.out.println(String.format("Fetched row: string = %s, number = %d",
 stringValue, numberValue));
    }
  }
}
```

Ausführen einer SQL-Transaktion

Sie können eine SQL-Transaktion starten, eine oder mehrere SQL-Anweisungen ausführen und anschließend die Änderungen mit einer Java-Anwendung übergeben.

> **Important**
>
> Bei einer Transaktion kommt es zu einer Zeitüberschreitung, wenn es innerhalb von drei Minuten keine Aufrufe gibt, die ihre Transaktions-ID verwenden. Wenn es zu einer Zeitüberschreitung kommt, bevor die Transaktion festgeschrieben wird, wird die Transaktion automatisch zurückgesetzt.
>
> Wenn Sie keine Transaktions-ID angeben, werden Änderungen, die sich durch den Aufruf ergeben, automatisch übergeben.

Im folgenden Beispiel wird eine SQL-Transaktion ausgeführt.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
    BeginTransactionResult beginTransactionResult =
 rdsData.beginTransaction(beginTransactionRequest);
    String transactionId = beginTransactionResult.getTransactionId();

    ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
```

```
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
    rdsData.executeStatement(executeStatementRequest);

    CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
    rdsData.commitTransaction(commitTransactionRequest);
  }
}
```

> **Note**
>
> Wenn Sie eine DDL-Anweisung ausführen, sollten Sie die Anweisung auch nach Ablauf des Aufrufs weiter ausführen. Wenn eine DDL-Anweisung vor Ende der Ausführung beendet wird, kann dies zu Fehlern und möglicherweise beschädigten Datenstrukturen führen. Um eine Anweisung weiter auszuführen, nachdem ein Aufruf das für die RDS-Daten-API geltende Zeitintervall von 45 Sekunden überschritten hat, legen Sie den Parameter `continueAfterTimeout` auf `true` fest.

Ausführen einer Stapel-SQL-Operation

Sie können mit einer Java-Anwendung Operationen für Masseneinfügungen und -aktualisierungen für ein Daten-Array ausführen. Sie können eine DML-Anweisung mit einem Array von Parametersätzen ausführen.

> **Important**
>
> Wenn Sie keine Transaktions-ID angeben, werden Änderungen, die sich durch den Aufruf ergeben, automatisch übergeben.

Im folgenden Beispiel wird eine Batch-Einfügungs-Operation ausgeführt.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
```

```
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
      AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("Hello")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(1L))
                    ),
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("World")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(2L))
                    )
            ));

    rdsData.batchExecuteStatement(request);
  }
}
```

Steuern des Timeout-Verhaltens der Daten-API

Alle Aufrufe der Daten-API erfolgen synchron. Angenommen, Sie führen eine Daten-API-Operation durch, die eine SQL-Anweisung wie `INSERT` oder `CREATE TABLE` ausführt. Wenn der Daten-API-Aufruf erfolgreich Ergebnisse zurückgibt, ist die SQL-Verarbeitung abgeschlossen.

Standardmäßig bricht die Daten-API eine Operation ab und gibt einen Timeout-Fehler zurück, wenn sie nicht innerhalb von 45 Sekunden abgeschlossen wird. In diesem Fall werden die Daten nicht eingefügt, die Tabelle wird nicht erstellt usw.

Sie können die Daten-API verwenden, um lang andauernde Operationen auszuführen, die nicht innerhalb von 45 Sekunden abgeschlossen werden können. Wenn Sie davon ausgehen, dass ein Vorgang, z. B. eine Massen-`INSERT`- oder DDL-Operation an einer großen Tabelle, länger als 45 Sekunden dauert, können Sie für die Operation `ExecuteStatement` den Parameter `continueAfterTimeout` angeben. Ihre Anwendung erhält weiterhin den Timeout-Fehler. Die Operation wird jedoch weiter ausgeführt und nicht abgebrochen. Ein Beispiel finden Sie unter Ausführen einer SQL-Transaktion.

Wenn das AWS SDK für Ihre Programmiersprache über einen eigenen Timeout-Zeitraum für API-Aufrufe oder HTTP-Socket-Verbindungen verfügt, stellen Sie sicher, dass alle diese Timeout-Zeiträume mehr als 45 Sekunden betragen. Bei einigen SDKs beträgt der Timeout-Zeitraum standardmäßig weniger als 45 Sekunden. Wir empfehlen, alle SDK-spezifischen oder Client-spezifischen Timeout-Zeiträume auf mindestens eine Minute festzulegen. Dadurch wird die Möglichkeit vermieden, dass Ihre Anwendung einen Timeout-Fehler erhält, während der Daten-API-Vorgang noch erfolgreich abgeschlossen wird. Auf diese Weise können Sie sicher sein, ob Sie den Vorgang wiederholen sollten oder nicht.

Nehmen wir beispielsweise an, dass das SDK einen Timeout-Fehler an Ihre Anwendung zurückgibt, die Daten-API-Operation aber trotzdem innerhalb des Daten-API-Timeout-Intervalls abgeschlossen wird. In diesem Fall könnte ein erneuter Ausführungsversuch der Operation Datenduplikate verursachen oder auf andere Weise zu falschen Ergebnissen führen. Das SDK wiederholt die Operation möglicherweise automatisch, wodurch falsche Daten entstehen, ohne dass Ihre Anwendung etwas unternommen hat.

Das Timeout-Intervall ist besonders wichtig für das Java-2-SDK. In diesem SDK betragen das API-Aufruf-Timeout und das HTTP-Socket-Timeout standardmäßig beide 30 Sekunden. In diesem Beispiel wird für das Timeout ein höherer Wert festgelegt:

```
public RdsDataClient createRdsDataClient() {
    return RdsDataClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .httpClientBuilder(createHttpClientBuilder())
        .credentialsProvider(defaultCredentialsProvider()) // Change this to your
 desired credentials provider
```

```
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return ApacheHttpClient.builder() // Change this to your desired HttpClient
        .socketTimeout(Duration.ofSeconds(60));
}
```

In diesem äquivalenten Beispiel wird der asynchrone Daten-Client verwendet:

```
public static RdsDataAsyncClient createRdsDataAsyncClient() {
    return RdsDataAsyncClient.builder()
        .region(Region.US_EAST_1) // Change this to your desired Region
        .overrideConfiguration(createOverrideConfiguration())
        .credentialsProvider(defaultCredentialsProvider())  // Change this to your
 desired credentials provider
        .build();
}

private static ClientOverrideConfiguration createOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
        .apiCallAttemptTimeout(Duration.ofSeconds(60))
        .build();
}

private HttpClientBuilder createHttpClientBuilder() {
    return NettyNioAsyncHttpClient.builder() // Change this to your desired
 AsyncHttpClient
        .readTimeout(Duration.ofSeconds(60));
}
```

Verwenden der Java-Client-Bibliothek für die RDS-Daten-API

Sie können eine Java-Client-Bibliothek für die RDS-Daten-API (Daten-API) herunterladen und verwenden. Die Java-Client-Bibliothek stellt eine Alternative zur Verwendung der Daten-API dar. Mit dieser Bibliothek können Sie Ihre clientseitigen Klassen den Anfragen und Antworten der Daten-API

zuordnen. Dieser Mapping-Support kann die Integration mit einigen bestimmten Java-Typen wie etwa `Date`, `Time` und `BigDecimal` erleichtern.

Herunterladen der Java Client-Bibliothek für die Daten-API

Die Data API Java-Client-Bibliothek ist in GitHub an folgender Stelle als Open Source verfügbar:

https://github.com/awslabs/rds-data-api-client-library-java

Sie können die Bibliothek manuell aus den Quelldateien aufbauen, die bewährte Methode ist jedoch, die Bibliothek unter Verwendung der Apache Maven-Dependenzverwaltung zu nutzen. Fügen Sie die folgende Abhängigkeit zu Ihrer Maven-POM-Datei hinzu.

Verwenden Sie für Version 2.x, die mit AWS-SDK-2.x kompatibel ist, Folgendes:

```
<dependency>
   <groupId>software.amazon.rdsdata</groupId>
   <artifactId>rds-data-api-client-library-java</artifactId>
   <version>2.0.0</version>
</dependency>
```

Verwenden Sie für Version 1.x, die mit AWS-SDK-1.x kompatibel ist, Folgendes:

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

Java Client-Bibliothek-Beispiele

Nachfolgend finden Sie einige typische Beispiele für die Verwendung der Daten-API-Java-Client-Bibliothek. Diese Beispiele gehen davon aus, dass Sie eine Tabelle `accounts` mit zwei Spalten haben: `accountId` und `name`. Weiterhin haben Sie das folgende Datentransferobjekt (DTO):

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

Die Client-Bibliothek ermöglicht die Übergabe von DTOs als Eingabeparameter. Das folgende Beispiel zeigt, wie Kunden-DTOs Eingabeparametersätzen zugeordnet werden.

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParamSets(account1, account2)
         .execute();
```

In manchen Fällen ist es einfacher, mit einfachen Werten als Eingabeparametern zu arbeiten. Verwenden Sie dazu die folgende Syntax.

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParameter("accountId", 3)
         .withParameter("name", "Zhang")
         .execute();
```

Es folgt ein weiteres Beispiel, das mit einfachen Werten als Eingabeparametern arbeitet.

```

client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
         .execute();

```

Die Client-Bibliothek bietet das automatische Mapping zu DTOs, wenn ein Ergebnis zurückgegeben wird. Die folgenden Beispiele zeigen, wie das Ergebnis Ihren DTOs zugeordnet wird.

```
List<Account> result = client.forSql("SELECT * FROM accounts")
          .execute()
          .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
          .execute()
          .mapToSingle(Account.class);
```

In vielen Fällen enthält die Datenbank-Ergebnismenge einen einzigen Wert. Um das Abrufen solcher Ergebnisse zu vereinfachen, bietet die Kundenbibliothek die folgende API an:

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
          .execute()
```

```
            .singleValue(Integer.class);
```

> **Note**
>
> Die `mapToList`-Funktion konvertiert einen SQL-Ergebnissatz in eine benutzerdefinierte Objektliste. Wir unterstützen die Verwendung der `.withFormatRecordsAs(RecordsFormatType.JSON)`-Anweisung in einem `ExecuteStatement`-Anruf für die Java-Clientbibliothek nicht, weil sie dem gleichen Zweck dient. Weitere Informationen finden Sie unter Verarbeiten von Abfrageergebnissen der Amazon-RDS-Data-API im JSON-Format.

Verarbeiten von Abfrageergebnissen der Amazon-RDS-Data-API im JSON-Format

Wenn Sie die `ExecuteStatement`-Operation aufrufen, können Sie auswählen, ob die Abfrageergebnisse als Zeichenfolge im JSON-Format zurückgegeben werden sollen. Auf diese Weise können Sie die JSON-Parsing-Funktionen Ihrer Programmiersprache verwenden, um die Ergebnismenge zu interpretieren und neu zu formatieren. Dies kann dazu beitragen, zu vermeiden, zusätzlichen Code zu schreiben, um die Ergebnismenge zu durchlaufen und jeden Spaltenwert zu interpretieren.

Zum Anfordern der Ergebnismenge im JSON-Format übergeben Sie den optionalen `formatRecordsAs`-Parameter mit dem Wert `JSON`. Die JSON-formatierte Ergebnismenge wird im Feld `formattedRecords` der `ExecuteStatementResponse`-Struktur zurückgegeben.

Die `BatchExecuteStatement`-Aktion gibt keine Ergebnismenge zurück. Daher gilt die JSON-Option nicht für diese Aktion.

Wenn Sie die Schlüssel in der JSON-Hash-Struktur anpassen möchten, definieren Sie Spaltenaliasnamen in der Ergebnismenge. Verwenden Sie dazu die `AS`-Klausel in der Spaltenliste Ihrer SQL-Abfrage.

Sie können die JSON-Funktion verwenden, um die Ergebnismenge besser lesbar zu machen und den Inhalt sprachspezifischen Frameworks zuzuordnen. Da das Volume der ASCII-kodierten Ergebnismenge größer als die Standarddarstellung ist, können Sie die Standarddarstellung für Abfragen auswählen, die eine große Anzahl von Zeilen oder große Spaltenwerte zurückgeben, die mehr Speicher belegen, als für Ihre Anwendung verfügbar ist.

Themen

- Abrufen von Abfrageergebnissen im JSON-Format
- Datentypenzuordnung
- Fehlerbehebung
- Beispiele

Abrufen von Abfrageergebnissen im JSON-Format

Damit Sie die Ergebnismenge als JSON-Zeichenfolge erhalten, schließen Sie `.withFormatRecordsAs(RecordsFormatType.JSON)` in den `ExecuteStatement`-Aufruf ein. Der Rückgabewert wird im Feld `formattedRecords` als JSON-Zeichenfolge angezeigt. In diesem Fall hat `columnMetadata` den Wert `null`. Die Spaltenbeschriftungen sind die Schlüssel des Objekts, das jede Zeile darstellt. Diese Spaltennamen werden für jede Zeile in der Ergebnismenge wiederholt. Die Spaltenwerte sind in Anführungszeichen gesetzte Zeichenfolgen, numerische Werte oder Sonderwerte, die `true`, `false` oder `null` darstellen. Spaltenmetadaten wie Längenbeschränkungen und der genaue Typ für Zahlen und Zeichenfolgen werden in der JSON-Antwort nicht beibehalten.

Wenn Sie den `.withFormatRecordsAs()`-Aufruf weglassen oder einen Parameter `NONE` angeben, wird die Ergebnismenge im Binärformat unter Verwendung der Felder `Records` und `columnMetadata` zurückgegeben.

Datentypenzuordnung

Die SQL-Werte in der Ergebnismenge werden einem kleineren Satz von JSON-Typen zugeordnet. Die Werte werden in JSON als Zeichenfolgen, Zahlen und einige spezielle Konstanten wie `true`, `false` und `null` dargestellt. Sie können diese Werte in Variablen in Ihrer Anwendung umwandeln, indem Sie eine starke oder schwache Eingabe verwenden, die für Ihre Programmiersprache geeignet ist.

JDBC-Datentyp	JSON-Datentyp
`INTEGER`, `TINYINT`, `SMALLINT`, `BIGINT`	Die Standardeinstellung ist Zahl. Zeichenfolge, wenn die Option `LongReturnType` auf `STRING` eingestellt ist.

JDBC-Datentyp	JSON-Datentyp
`FLOAT`, `REAL`, `DOUBLE`	Anzahl
`DECIMAL`	Die Standardeinstellung ist Zeichenfolge. Zahl, wenn die Option `DecimalReturnType` auf `DOUBLE_OR_LONG` eingestellt ist.
`STRING`	String
`BOOLEAN`, `BIT`	Boolesch
`BLOB`, `BINARY`, `VARBINARY` , `LONGVARBINARY`	Zeichenfolge in Base64-Kodierung.
`CLOB`	String
`ARRAY`	Array
`NULL`	`null`
Andere Typen (einschließlich datums- und zeitbezogener Typen)	String

Fehlerbehebung

Die JSON-Antwort ist auf 10 Megabyte begrenzt. Wenn die Antwort größer als dieses Limit ist, erhält Ihr Programm einen `BadRequestException`-Fehler. In diesem Fall können Sie den Fehler mit einer der folgenden Methoden beheben:

- Reduzieren Sie die Anzahl der Zeilen in der Ergebnismenge. Fügen Sie dazu eine `LIMIT`-Klausel hinzu. Sie können eine große Ergebnismenge in mehrere kleinere aufteilen, indem Sie mehrere Abfragen mit den Klauseln `LIMIT` und `OFFSET` senden.

  Wenn die Ergebnismenge Zeilen enthält, die durch die Anwendungslogik herausgefiltert werden, können Sie diese Zeilen aus der Ergebnismenge entfernen, indem Sie der `WHERE`-Klausel weitere Bedingungen hinzufügen.
- Reduzieren Sie die Anzahl der Spalten in der Ergebnismenge. Entfernen Sie dazu Elemente aus der Auswahlliste der Abfrage.

- Verkürzen Sie die Spaltenbeschriftungen, indem Sie Spaltenaliasnamen in der Abfrage verwenden. Jeder Spaltenname wird in der JSON-Zeichenfolge für jede Zeile der Ergebnismenge wiederholt. Daher könnte ein Abfrageergebnis mit langen Spaltennamen und vielen Zeilen die Größenbeschränkung überschreiten. Verwenden Sie insbesondere Spaltenaliasnamen für komplizierte Ausdrücke, um zu vermeiden, dass der gesamte Ausdruck in der JSON-Zeichenfolge wiederholt wird.
- Während Sie mit SQL Spaltenaliasnamen verwenden können, um eine Ergebnismenge mit mehr als einer Spalte mit demselben Namen zu erzeugen, sind doppelte Schlüsselnamen in JSON nicht zulässig. Die RDS-Daten-API gibt einen Fehler zurück, wenn Sie die Ergebnismenge im JSON-Format anfordern und mehr als eine Spalte denselben Namen hat. Stellen Sie daher sicher, dass alle Spaltenbeschriftungen eindeutige Namen haben.

Beispiele

Die folgenden Java-Beispiele zeigen, wie Sie `ExecuteStatement` mit der Antwort als JSON-formatierte Zeichenfolge aufrufen und dann die Ergebnismenge interpretieren. Ersetzen Sie die entsprechenden Werte für die Parameter *`databaseName`*, *`secretStoreArn`* und *`clusterArn`*.

Das folgende Java-Beispiel veranschaulicht eine Abfrage, die einen numerischen Dezimalwert in der Ergebnismenge zurückgibt. `assertThat`-Aufrufe testen anhand der Regeln für JSON-Ergebnismengen, ob die Felder der Antwort die erwarteten Eigenschaften haben.

Dieses Beispiel funktioniert mit dem folgenden Schema und den folgenden Beispieldaten:

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);
```

```
public void JSON_result_set_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request);
}
```

Der Wert des Felds `formattedRecords` aus dem vorhergehenden Programm lautet:

```
[{"a":10.0}]
```

Die Felder `Records` und `ColumnMetadata` in der Antwort sind aufgrund des Vorhandenseins der JSON-Ergebnismenge beide null.

Das folgende Java-Beispiel veranschaulicht eine Abfrage, die einen numerischen Ganzzahlwert in der Ergebnismenge zurückgibt. Das Beispiel ruft `getFormattedRecords` auf, um nur die JSON-formatierte Zeichenfolge zurückzugeben und die anderen Antwortfelder zu ignorieren, die leer oder null sind. Im Beispiel wird das Ergebnis in eine Struktur deserialisiert, die eine Liste von Datensätzen darstellt. Jeder Datensatz enthält Felder, deren Namen den Spaltenaliasnamen aus der Ergebnismenge entsprechen. Diese Methode vereinfacht den Code, der die Ergebnismenge analysiert. Ihre Anwendung muss nicht die Zeilen und Spalten der Ergebnismenge durchlaufen und jeden Wert in den entsprechenden Typ konvertieren.

Dieses Beispiel funktioniert mit dem folgenden Schema und den folgenden Beispieldaten:

```
create table test_simplified_json (a int);
insert into test_simplified_json values(17);
```

```
public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
      .getFormattedRecords();

/* Turn the result set into a Java object, a list of records.
   Each record has a field 'a' corresponding to the column
   labelled 'a' in the result set. */
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
        });
}
```

Der Wert des Felds `formattedRecords` aus dem vorhergehenden Programm lautet:

```
[{"a":17}]
```

Zum Abrufen der Spalte a der Ergebniszeile 0 würde sich die Anwendung auf `recordsList.get(0).a` beziehen.

Im Gegensatz dazu zeigt das folgende Java-Beispiel die Art von Code, der zum Erstellen einer Datenstruktur erforderlich ist, die die Ergebnismenge enthält, wenn Sie das JSON-Format nicht verwenden. In diesem Fall enthält jede Zeile der Ergebnismenge Felder mit Informationen über einen einzelnen Benutzer. Das Erstellen einer Datenstruktur zur Darstellung der Ergebnismenge erfordert das Durchlaufen der Zeilen. Für jede Zeile ruft der Code den Wert jedes Felds ab, führt eine entsprechende Typkonvertierung durch und weist das Ergebnis dem entsprechenden Feld im Objekt zu, das die Zeile darstellt. Dann fügt der Code das Objekt, das jeden Benutzer repräsentiert, der Datenstruktur hinzu, die die gesamte Ergebnismenge darstellt. Wenn die Abfrage geändert wurde, um Felder in der Ergebnismenge neu anzuordnen, hinzuzufügen oder zu entfernen, müsste sich der Anwendungscode ebenfalls ändern.

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
      .userId(row.get(0).getLongValue())
      .firstName(row.get(1).getStringValue())
      .lastName(row.get(2).getStringValue())
      .dob(Instant.parse(row.get(3).getStringValue()))
      .build();
    result.add(user);
  }
```

Die folgenden Beispielwerte zeigen die Werte des Felds `formattedRecords` für Ergebnismengen mit unterschiedlicher Anzahl von Spalten, Spaltenaliasnamen und Spaltendatentypen.

Wenn die Ergebnismenge mehrere Zeilen enthält, wird jede Zeile als Objekt dargestellt, das ein Array-Element ist. Jede Spalte in der Ergebnismenge wird zu einem Schlüssel im Objekt. Die Schlüssel werden für jede Zeile in der Ergebnismenge wiederholt. Daher müssen Sie für Ergebnismengen, die aus vielen Zeilen und Spalten bestehen, möglicherweise kurze Spaltenaliasnamen definieren, um zu vermeiden, dass das Längenlimit für die gesamte Antwort überschritten wird.

Dieses Beispiel funktioniert mit dem folgenden Schema und den folgenden Beispieldaten:

```
create table sample_names (id int, name varchar(128));
```

```
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"),
 (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"},{"id":1,"name":"Mohan"},
{"id":2,"name":"Maria"},{"id":3,"name":"Bruce"},{"id":4,"name":"Jasmine"}]
```

Wenn eine Spalte in der Ergebnismenge als Ausdruck definiert ist, wird der Text des Ausdrucks zum JSON-Schlüssel. Daher ist es normalerweise praktisch, einen beschreibenden Spaltenalias für jeden Ausdruck in der Auswahlliste der Abfrage zu definieren. Die folgende Abfrage enthält beispielsweise Ausdrücke wie Funktionsaufrufe und arithmetische Operationen in ihrer Auswahlliste.

```
select count(*), max(id), 4+7 from sample_names;
```

Diese Ausdrücke werden als Schlüssel an die JSON-Ergebnismenge übergeben.

```
[{"count(*)":5,"max(id)":4,"4+7":11}]
```

Wenn Sie AS-Spalten mit beschreibenden Beschriftungen hinzufügen, können die Schlüssel in der JSON-Ergebnismenge einfacher interpretiert werden.

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from
 sample_names;
```

Bei der überarbeiteten SQL-Abfrage werden die Spaltenbeschriftungen, die durch die AS-Klauseln definiert werden, als Schlüsselnamen verwendet.

```
[{"rows":5,"largest_id":4,"addition_result":11}]
```

Der Wert für jedes Schlüssel-Wert-Paar in der JSON-Zeichenfolge kann eine Zeichenfolge in Anführungszeichen sein. Die Zeichenfolge enthält möglicherweise Unicode-Zeichen. Wenn die Zeichenfolge Escape-Sequenzen oder die Zeichen " oder \ enthält, wird diesen Zeichen ein Escape-Zeichen mit umgekehrtem Schrägstrich vorangestellt. Die folgenden Beispiele für JSON-Zeichenfolgen veranschaulichen diese Möglichkeiten. Das Ergebnis `string_with_escape_sequences` enthält z. B. die Sonderzeichen Rücktaste, Zeilenumbruch, Wagenrücklauf, Tabulator, Seitenvorschub und \.

```
[{"quoted_string":"hello"}]
[{"unicode_string":"####"}]
```

```
[{"string_with_escape_sequences":"\b \n \r \t \f \\ '"}]
```

Der Wert für jedes Schlüssel-Wert-Paar in der JSON-Zeichenfolge kann auch eine Zahl darstellen. Die Zahl kann eine Ganzzahl, ein Gleitkommawert, ein negativer Wert oder ein Wert sein, der in Exponentialschreibweise dargestellt wird. Die folgenden Beispiele für JSON-Zeichenfolgen veranschaulichen diese Möglichkeiten.

```
[{"integer_value":17}]
[{"float_value":10.0}]
[{"negative_value":-9223372036854775808,"positive_value":9223372036854775807}]
[{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.79769313486231
```

Boolesche und Nullwerte werden mit den Sonderschlüsselwörtern `true`, `false` und `null` ohne Anführungszeichen dargestellt. Die folgenden Beispiele für JSON-Zeichenfolgen veranschaulichen diese Möglichkeiten.

```
[{"boolean_value_1":true,"boolean_value_2":false}]
[{"unknown_value":null}]
```

Wenn Sie einen Wert eines BLOB-Typs auswählen, wird das Ergebnis in der JSON-Zeichenfolge als Base64-kodierter Wert dargestellt. Wenn Sie den Wert wieder in seine ursprüngliche Darstellung umwandeln möchten, können Sie die entsprechende Dekodierungsfunktion in der Sprache Ihrer Anwendung verwenden. In Java rufen Sie beispielsweise die Funktion `Base64.getDecoder().decode()` auf. Die folgende Beispielausgabe zeigt das Ergebnis der Auswahl des BLOB-Werts `hello world` und gibt die Ergebnismenge als JSON-Zeichenfolge zurück.

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

Das folgende Python-Beispiel zeigt, wie Sie auf die Werte aus dem Ergebnis eines Aufrufs der Python-Funktion `execute_statement` zugreifen. Die Ergebnismenge ist ein Zeichenfolgenwert im Feld `response['formattedRecords']`. Der Code verwandelt die JSON-Zeichenfolge durch Aufrufen der Funktion `json.loads` in eine Datenstruktur. Dann ist jede Zeile der Ergebnismenge ein Listenelement innerhalb der Datenstruktur und innerhalb jeder Zeile können Sie auf jedes Feld der Ergebnismenge nach Namen verweisen.

```
import json

result = json.loads(response['formattedRecords'])
```

```
print (result[0]["id"])
```

Das folgende JavaScript-Beispiel zeigt, wie Sie auf die Werte aus dem Ergebnis eines Aufrufs der JavaScript-Funktion `executeStatement` zugreifen. Die Ergebnismenge ist ein Zeichenfolgenwert im Feld `response.formattedRecords`. Der Code verwandelt die JSON-Zeichenfolge durch Aufrufen der Funktion `JSON.parse` in eine Datenstruktur. Dann ist jede Zeile der Ergebnismenge ein Array-Element innerhalb der Datenstruktur und innerhalb jeder Zeile können Sie auf jedes Feld der Ergebnismenge nach Namen verweisen.

```
<script>
    const result = JSON.parse(response.formattedRecords);
    document.getElementById("display").innerHTML = result[0].id;
</script>
```

Fehlerbehebung bei Problemen mit der Amazon-RDS-Daten-API

Verwenden Sie die folgenden Abschnitte mit dem Titel „Allgemeine Fehlermeldungen“, um Probleme zu beheben, die Sie mit der Amazon-RDS-Daten-API (Daten-API) haben.

Themen

- Transaktion <transaction_ID> nicht gefunden
- Packet for Query Is Too Large (Paket für Abfrage zu groß)
- Database Response Exceeded Size Limit Datenbankantwort überschreitet Größenlimit)
- HttpEndpointist nicht für Cluster aktiviert <cluster_ID>
- DatabaseErrorException: Transaction führt immer noch eine Abfrage aus
- Ausnahme aufgrund eines nicht unterstützten Ergebnisses
- Keine Unterstützung für mehrere Anweisungen
- Keine Unterstützung für den Schemaparameter
- IPv6 Probleme mit der Konnektivität

Transaktion <transaction_ID> nicht gefunden

In diesem Fall wurde die in einem Data-API-Aufruf angegebene Transaktions-ID nicht gefunden. Die Ursache für dieses Problem wird an die Fehlermeldung angehängt und ist eine der folgenden:

- Die Transaktion ist möglicherweise abgelaufen.

Stellen Sie sicher, dass jeder Transaktionsaufruf innerhalb von drei Minuten nach dem letzten ausgeführt wird.

Es ist auch möglich, dass die angegebene Transaktions-ID nicht durch einen BeginTransactionAnruf erstellt wurde. Stellen Sie sicher, dass Ihr Aufruf eine gültige Transaktions-ID hat.

- Ein vorheriger Aufruf führte zu einer Beendigung Ihrer Transaktion.

  Die Transaktion wurde bereits von Ihrem `CommitTransaction`- oder `RollbackTransaction`-Aufruf beendet.
- Die Transaktion wurde aufgrund eines Fehlers eines früheren Aufrufs abgebrochen.

  Prüfen Sie, ob Ihre vorherigen Aufrufe Ausnahmen ausgelöst haben.

Informationen zum Ausführen von Transaktionen finden Sie unter Aufrufen der Daten-API von Amazon RDS.

Packet for Query Is Too Large (Paket für Abfrage zu groß)

In diesem Fall war die zurückgegebene Ergebnismenge für eine Zeile zu groß. Die Größenbegrenzung der Data-API beträgt 64 KB pro Zeile in der von der Datenbank zurückgegebenen Ergebnismenge.

Um dieses Problem zu beheben, stellen Sie sicher, dass jede Zeile in einem Ergebnissatz höchstens 64 KB groß ist.

Database Response Exceeded Size Limit Datenbankantwort überschreitet Größenlimit)

In diesem Fall war die Größe der von der Datenbank zurückgegebenen Ergebnismenge zu groß. Das Data-API-Limit beträgt 1 MiB für die von der Datenbank zurückgegebene Ergebnismenge.

Um dieses Problem zu beheben, stellen Sie sicher, dass Aufrufe der Daten-API höchstens 1 MiB Daten zurückgeben. Wenn Sie mehr als 1 MiB zurückgeben müssen, können Sie mit der `LIMIT`-Klausel in Ihrer Abfrage mehrere `ExecuteStatement`-Aufrufe verwenden.

Weitere Informationen über die `LIMIT`-Klausel finden Sie unter SELECT-Syntax in der MySQL-Dokumentation.

HttpEndpointist nicht für Cluster aktiviert <cluster_ID>

Überprüfen Sie die folgenden möglichen Ursachen für dieses Problem:

- Der Aurora-DB-Cluster unterstützt die Daten-API nicht. Informationen zu den Typen von DB-Clustern, die die RDS-Daten-API unterstützt, finden Sie unter the section called “Verfügbarkeit von Regionen und Versionen für die Amazon-RDS-Daten-API”.
- Die Daten-API ist für den Aurora-DB-Cluster nicht aktiviert. Um die Daten-API mit einem Aurora-DB-Cluster zu verwenden, muss die Daten-API für den DB-Cluster aktiviert sein. Informationen zur Aktivierung der Data-API finden Sie unter Aktivieren der Amazon-RDS-Daten-API.
- Der DB-Cluster wurde umbenannt, nachdem die Daten-API für ihn aktiviert wurde. Deaktivieren Sie in diesem Fall die Daten-API für diesen Cluster und aktivieren Sie sie dann erneut.
- Der von Ihnen angegebene ARN stimmt nicht genau mit dem ARN des Clusters überein. Stellen Sie sicher, dass der von einer anderen Quelle zurückgegebene oder durch Programmlogik erstellte ARN genau mit dem ARN des Clusters übereinstimmt. Stellen Sie beispielsweise sicher, dass der von Ihnen verwendete ARN die richtigen Groß-/Kleinschreibung für alle alphabetischen Zeichen aufweist.

DatabaseErrorException: Transaction führt immer noch eine Abfrage aus

Wenn Ihre Anwendung eine Anfrage mit einer Transaktions-ID sendet und diese Transaktion gerade eine weitere Anfrage verarbeitet, gibt die Daten-API sofort diesen Fehler an Ihre Anwendung zurück. Dieser Zustand kann auftreten, wenn Ihre Anwendung asynchrone Anfragen stellt und dabei einen Mechanismus wie „Promises“ in Javascript verwendet.

Um dieses Problem zu lösen, warten Sie, bis die vorherige Anfrage abgeschlossen ist, und versuchen Sie es dann erneut. Sie können den Schritt so lange wiederholen, bis der Fehler nicht mehr auftritt oder die Anwendung eine andere Art von Fehler erhält.

Dieser Zustand kann bei der Daten-API für Aurora Serverless v2- und für bereitgestellte Instances auftreten. In der Daten-API für Aurora Serverless v1 warten nachfolgende Anfragen für dieselbe Transaktions-ID automatisch, bis die vorherige Anfrage abgeschlossen ist. Bei diesem älteren Verhalten kann es jedoch möglicherweise zu Timeouts kommen, wenn die vorherige Anfrage zu lange gedauert. Wenn Sie eine ältere Daten-API-Anwendung portieren, die gleichzeitige Anfragen stellt, ändern Sie Ihre Logik für die Ausnahmeverarbeitung so, dass sie diese neue Art von Fehler berücksichtigt.

Ausnahme aufgrund eines nicht unterstützten Ergebnisses

Die Daten-API unterstützt nicht alle Datentypen. Dieser Fehler tritt auf, wenn Sie eine Abfrage ausführen, die einen nicht unterstützten Datentyp zurückgibt.

Um dieses Problem zu umgehen, konvertieren Sie den nicht unterstützten Datentyp in `TEXT`. Zum Beispiel:

```
SELECT custom_type::TEXT FROM my_table;
-- OR
SELECT CAST(custom_type AS TEXT) FROM my_table;
```

Keine Unterstützung für mehrere Anweisungen

Mehrere Anweisungen werden in der Daten-API für Aurora Serverless v2 und bereitgestellte Cluster nicht unterstützt. Wenn Sie versuchen, mehrere Anweisungen in einem einzigen API-Aufruf auszuführen, wird der Fehler zurückgegeben.

Um mehrere Anweisungen auszuführen, nutzen Sie separate `ExecuteStatement`-API-Aufrufe oder die `BatchExecuteStatement`-API für die Stapelverarbeitung.

Keine Unterstützung für den Schemaparameter

Aurora Serverless v1 ignoriert den Schemaparameter stillschweigend. Aurora Serverless v2 und bereitgestellte Cluster hingegen lehnen API-Aufrufe, die den Schemaparameter enthalten, explizit ab.

Um dieses Problem zu lösen, entfernen Sie den Schemaparameter aus allen Aufrufen der Daten-API, wenn Sie Aurora Serverless v2 oder bereitgestellte Cluster verwenden.

IPv6 Probleme mit der Konnektivität

Wenn beim Herstellen einer Verbindung zur Daten-API über IPv6 Endpunkte Probleme auftreten, überprüfen Sie die folgenden möglichen Ursachen:

- Netzwerk unterstützt nicht IPv6: Stellen Sie sicher, dass Ihre Netzwerkinfrastruktur dies unterstützt IPv6 und ob das IPv6 Routing korrekt konfiguriert ist.
- Probleme mit der DNS-Auflösung: Stellen Sie sicher, dass Ihr DNS-Resolver AAAA-Datensätze für die Dual-Stack-Endpunkte auflösen kann (z. B. `rds-data.us-east-1.api.aws`).

- Sicherheitsgruppenkonfiguration: Aktualisieren Sie die Sicherheitsgruppenregeln, um IPv6 Datenverkehr auf Port 443 (HTTPS) zuzulassen. Fügen Sie Regeln für IPv6 CIDR-Blöcke hinzu (z. B. `::/0` für alle IPv6 Adressen).
- Netzwerk-ACL-Konfiguration: Stellen Sie sicher, dass das Netzwerk den IPv6 Datenverkehr an den erforderlichen Ports ACLs zulässt.
- Kompatibilität mit der Clientbibliothek: Stellen Sie sicher, dass Ihre HTTP-Clientbibliotheken IPv6 und Dual-Stack-Konnektivität AWS SDKs unterstützen.
- VPC-Endpunktkonfiguration: Stellen Sie bei Verwendung sicher PrivateLink, dass Ihr VPC-Endpunkt für die Unterstützung konfiguriert ist IPv6 und dass den zugehörigen Subnetzen IPv6 CIDR-Blöcke zugewiesen sind.

So beheben Sie Verbindungsprobleme: IPv6

1. Testen Sie die Konnektivität mit den Endpunkten „ IPv4-only“ (`.amazonaws.com`), um sicherzustellen, dass das Problem spezifisch für ist. IPv6
2. Verwenden Sie Netzwerkdiagnosetools, um die IPv6 Konnektivität zu den Dual-Stack-Endpunkten zu überprüfen.
3. Überprüfen Sie die CloudTrail Protokolle auf Authentifizierungs- oder Autorisierungsfehler bei der Verwendung von IPv6 Endpunkten.
4. Stellen Sie sicher, dass Ihre Anwendung korrekt für die Verwendung des neuen Dual-Stack-Endpunkts konfiguriert ist. URLs

Protokollieren von Aufrufen der Amazon-RDS-Daten-API mit AWS CloudTrail

RDS-Daten-API (Daten-API) ist in AWS CloudTrail integriert. Hierbei handelt es sich um einen Service, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS-Service in der Daten-API durchgeführt wurden. CloudTrail erfasst alle API-Aufrufe für die Daten-API als Ereignisse, einschließlich Aufrufe von der Amazon-RDS-Konsole und von Codeaufrufen an Daten-API-Vorgänge. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail-Ereignissen an einen Amazon S3-Bucket, einschließlich Ereignisse für Data API aktivieren. Mit den von CloudTrail gesammelten Daten können Sie viele Informationen bestimmen. Diese Informationen umfassen die Anforderung an Data API, die IP-Adresse, von der die

Anforderung gestellt wurde, wer die Anforderung gestellt hat, wann sie gestellt wurde, und zusätzliche Details.

Weitere Informationen zu CloudTrail finden Sie im AWS CloudTrail-Benutzerhandbuch.

Arbeiten mit Data API-Informationen in CloudTrail

CloudTrail wird beim Erstellen Ihres AWS-Kontos für Sie aktiviert. Wenn eine unterstützte Aktivität (Verwaltungsereignisse) in der Daten-API auftritt, wird diese in einem CloudTrail-Ereignisprotokoll zusammen mit anderen AWS-Serviceereignissen im Ereignisverlauf aufgezeichnet. Sie können die neusten Verwaltungsereignisse in Ihrem AWS-Konto anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Arbeiten mit dem CloudTrail-Ereignisverlauf im AWS CloudTrail-Benutzerhandbuch.

Erstellen Sie für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS-Konto, einschließlich Ereignissen für Data API, einen Trail. Ein Trail ermöglicht es CloudTrail, Protokolldateien in einem Amazon-S3-Bucket bereitzustellen. Wenn Sie einen Trail in der Konsole anlegen, gilt dieser für alle AWS-Regionen. Der Trail protokolliert Ereignisse aus allen AWS-Regionen in der AWS-Partition und stellt die Protokolldateien in dem Amazon-S3-Bucket bereit, den Sie angeben. Darüber hinaus können Sie andere AWS-Services konfigurieren, um die in den CloudTrail-Protokollen erfassten Ereignisdaten weiter zu analysieren und entsprechend zu agieren. Weitere Informationen finden Sie in folgenden Themen im AWS CloudTrail-Benutzerhandbuch:

- Übersicht zum Erstellen eines Trails
- In CloudTrail unterstützte Services und Integrationen
- Konfigurieren von Amazon SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail-Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail-Protokolldateien aus mehreren Konten

Alle Daten-API-Operationen werden von CloudTrail protokolliert und in der Referenz zur Datenservice-API von Amazon RDS dokumentiert. Zum Beispiel werden durch Aufrufe der `BatchExecuteStatement`-, `BeginTransaction`-, `CommitTransaction`- und `ExecuteStatement`-Operationen Einträge in den CloudTrail-Protokolldateien generiert.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anforderung mit Root- oder -Benutzeranmeldeinformationen ausgeführt wurde.

- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde

Weitere Informationen finden Sie unter CloudTrail-Element userIdentity.

Einschließen und Ausschließen von Daten-API-Ereignissen aus einem AWS CloudTrail-Trail

Die meisten Daten-API-Benutzer verlassen sich auf die Ereignisse in einem AWS CloudTrail-Trail, um eine Aufzeichnung der Daten-API-Operationen bereitzustellen. Die Ereignisdaten geben weder den Namen der Datenbank noch den Namen des Schemas oder die SQL-Anweisungen in Anforderungen an die Daten-API preis. Wenn Sie jedoch wissen, welcher Benutzer zu einem bestimmten Zeitpunkt eine Typ von Aufruf für einen bestimmten DB-Cluster durchgeführt hat, kann dies dazu beitragen, anomale Zugriffsmuster zu erkennen.

Einschließen von Daten-API-Ereignissen in einen AWS CloudTrail-Trail

Für Aurora PostgreSQL Serverless v2 und bereitgestellte Datenbanken werden die folgenden Daten-API-Vorgänge als Datenereignisse in AWS CloudTrail protokolliert. Datenereignisse sind API-Operationen auf Datenebene mit hohem Volumen, die CloudTrail standardmäßig nicht protokolliert. Für Datenereignisse werden zusätzliche Gebühren fällig. Weitere Informationen zu CloudTrail-Preisen finden Sie unter AWS CloudTrail – Preise.

- BatchExecuteStatement
- BeginTransaction
- CommitTransaction
- ExecuteStatement
- RollbackTransaction

Sie können die CloudTrail-Konsole, AWS CLI oder CloudTrail-API-Vorgänge verwenden, um diese Daten-API-Vorgänge zu protokollieren. Wählen Sie in der CloudTrail-Konsole RDS-Daten-API – DB-Cluster als Datenereignistyp aus. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen mit der AWS-Managementkonsole im Benutzerhandbuch für AWS CloudTrail.

Führen Sie den Befehl `aws cloudtrail put-event-selectors` über die AWS CLI aus, um diese Daten-API-Vorgänge für Ihren Trail zu protokollieren. Um alle Daten-API-Ereignisse auf DB-

Clustern zu protokollieren, geben Sie `AWS::RDS::DBCluster` als Ressourcentyp an. Im folgenden Beispiel werden alle Daten-API-Ereignisse auf DB-Clustern protokolliert. Weitere Informationen finden Sie unter Protokollieren von Datenereignissen mit der AWS Command Line Interface im Benutzerhandbuch für AWS CloudTrail.

```
aws cloudtrail put-event-selectors --trail-name trail_name --advanced-event-selectors \
'{
   "Name": "RDS Data API Selector",
   "FieldSelectors": [
      {
         "Field": "eventCategory",
         "Equals": [
            "Data"
         ]
      },
      {
         "Field": "resources.type",
         "Equals": [
            "AWS::RDS::DBCluster"
         ]
      }
   ]
}'
```

Sie können erweiterte Ereignisauswahlen so konfigurieren, dass sie zusätzlich nach den Feldern `readOnly`, `eventName`, und `resources.ARN` gefiltert werden. Weitere Informationen zu diesen Feldern finden Sie unter AdvancedFieldSelector.

Ausschließen von Data API-Ereignissen aus einem AWS CloudTrail-Trail (nur Aurora Serverless v1)

Für Aurora Serverless v1 sind Daten-API-Aufrufe Verwaltungsereignisse. Standardmäßig sind alle Daten-API-Ereignisse in einem AWS CloudTrail Trail enthalten. Da die Daten-API jedoch eine große Anzahl von Ereignissen generieren kann, können Sie diese Ereignisse aus Ihrem CloudTrail-Trail ausschließen. Die Einstellung Amazon-RDS-Daten-API-Ereignisse ausschließen schließt alle Daten-API-Ereignisse aus dem Trail aus. Sie können bestimmte Daten-API-Ereignisse nicht ausschließen.

Zum Ausschließen von Daten-API-Ereignissen aus einem Trail führen Sie die folgenden Schritte aus:

- Wählen Sie in der CloudTrail-Konsole die Ereignisse der Amazon-RDS-Daten-API ausschließen-Einstellung, wenn Sie Einen Trail anlegen oder einen Trail aktualisieren.

- Verwenden Sie in der CloudTrail-API die Operation PutEventSelectors. Wenn Sie erweiterte Ereignisauswahlen verwenden, können Sie Daten-API-Ereignisse ausschließen, indem Sie das Feld `eventSource` so festlegen, dass es nicht gleich `rdsdata.amazonaws.com` ist. Wenn Sie erweiterte Ereignisauswahlen verwenden, können Sie Daten-API-Ereignisse ausschließen, indem Sie das Attribut `ExcludeManagementEventSources` auf `rdsdata.amazonaws.com` festlegen. Weitere Informationen finden Sie unter Protokollieren von Ereignissen über die AWS Command Line Interface im Benutzerhandbuch für AWS CloudTrail.

> **Warning**
>
> Das Ausschließen von Daten-API-Ereignissen aus einem CloudTrail-Protokoll kann Daten-API-Aktionen verdecken. Seien Sie vorsichtig, wenn Sie Prinzipalen die `cloudtrail:PutEventSelectors`-Berechtigung erteilen, die zum Ausführen dieses Vorgangs erforderlich ist.

Sie können diesen Ausschluss jederzeit deaktivieren, indem Sie die Konsoleneinstellung oder die Ereignisselektoren für einen Trail ändern. Der Trail beginnt dann mit der Aufzeichnung von Daten-API-Ereignissen. Er kann jedoch keine Daten-API-Ereignisse wiederherstellen, die aufgetreten sind, während der Ausschluss wirksam war.

Wenn Sie Daten-API-Ereignisse mithilfe der Konsole oder API ausschließen, wird die resultierende `PutEventSelectors`-CloudTrail-API-Operation auch in den CloudTrail-Protokollen protokolliert. Wenn Daten-API-Ereignisse nicht in Ihren CloudTrail-Protokollen angezeigt werden, suchen Sie nach einem `ExcludeManagementEventSources`-Ereignis, bei dem das Attribut `PutEventSelectors` auf `rdsdata.amazonaws.com` gesetzt ist.

Weitere Informationen finden Sie unter Protokollieren von Managementereignissen für Trails im AWS CloudTrail-Benutzerhandbuch.

Grundlegendes zu Data API-Protokolldateieinträgen

Ein Trail ist eine Konfiguration, durch die Ereignisse als Protokolldateien an den von Ihnen angegebenen Amazon-S3-Bucket übermittelt werden. CloudTrail-Protokolldateien können einen oder mehrere Einträge enthalten. Ein Ereignis stellt eine einzelne Anfrage aus einer beliebigen Quelle dar und enthält unter anderem Informationen über die angeforderte Aktion, das Datum und die Uhrzeit der Aktion sowie über die Anfrageparameter. CloudTrail-Protokolleinträge sind kein geordnetes Stack-Trace der öffentlichen API-Aufrufe und erscheinen daher in keiner bestimmten Reihenfolge.

Aurora PostgreSQL Serverless v2 und bereitgestellt

Das folgende Beispiel zeigt einen CloudTrail-Protokolleintrag, der den `ExecuteStatement`-Vorgang für Aurora PostgreSQL Serverless v2 und bereitgestellte Datenbanken demonstriert: Für diese Datenbanken sind alle Daten-API-Ereignisse Datenereignisse, bei denen die Ereignisquelle rdsdataapi.amazonaws.com und der Ereignistyp RDS-Daten-Service lautet.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdataapi.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "Rds Data Service",
    "recipientAccountId": "123456789012"
}
```


Aurora Serverless v1

Das folgende Beispiel zeigt, wie der vorherige Beispiel-CloudTrail-Protokolleintrag für Aurora Serverless v1 angezeigt wird. Für Aurora Serverless v1 sind alle Ereignisse Verwaltungsereignisse, bei denen die Ereignisquelle rdsdata.amazonaws.com und der Ereignistyp AwsApiCall lautet.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdata.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Überwachen von RDS-Daten-API-Abfragen mit Performance Insights

Wenn Ihr Aurora-Cluster Aurora Serverless v2 oder bereitgestellte Instances ausführt, können Sie Performance Insights mit der RDS-Daten-API verwenden.

Weitere Informationen zur Verwendung von Performance Insights mit Aurora finden Sie unter Überwachung mit Performance Insights auf .

So werden RDS-Daten-API-Abfragen in Performance Insights dargestellt

Mit der Daten-API verarbeitet Ihr Aurora-Cluster Abfragen basierend auf Daten-API-Aufrufen, die Sie über Ihre Anwendung übermitteln. Die Daten-API führt im Rahmen ihrer eigenen internen Abläufe auch einige SQL-Anweisungen aus, z. B. das Abbrechen von Abfragen, die den Timeout-Schwellenwert überschreiten. Beide Arten von SQL-Vorgängen werden in den Statistiken und Diagrammen von Performance Insights angezeigt.

- Bei Daten-API-Abfragen, die Sie an einen Aurora-Cluster senden, ist das Host-Feld im PI-Dashboard als RDS-Daten-API gekennzeichnet. Für Aurora PostgreSQL hat das Feld application_name den Wert `rds-data-api`. Suchen Sie nach diesen Beschriftungen, wenn Sie die Datenbanklast mit den Dimensionen Top-Hosts oder Top-Anwendungen analysieren.
- Alle internen Abfragen, die von der Daten-API ausgeführt werden, um Datenbankaspekte wie den Verbindungspool und Abfrage-Timeouts zu verwalten, sind mit dem Präfix RDS-Daten-API versehen. Beispiel: `/* RDS Data API */ select * from my_table;` Sucht nach diesen Präfixen, wenn Sie die Datenbanklast anhand von Top SQL als Dimension analysieren. Anweisungen werden mit einem SQL-Kommentar von `/* RDS Data API */` versehen.

Arbeiten mit dem Aurora-Abfrage-Editor

Mit dem Aurora-Abfrage-Editor können Sie SQL-Anweisungen auf Ihrem Aurora-DB-Cluster über die AWS-Managementkonsole ausführen. Sie können SQL-Abfragen, DML-Anweisungen (Data Manipulation Language) und DDL-Anweisungen (Data Definition Language) ausführen. Mithilfe der Konsolenschnittstelle können Sie die Datenbankwartung durchführen, Berichte erstellen und SQL-Experimente durchführen. Sie können vermeiden, die Netzwerkkonfiguration so einzurichten, dass Sie von einem separaten Clientsystem wie einer EC2 Instance oder einem Laptop aus eine Verbindung zu Ihrem DB-Cluster herstellen.

Der Abfrage-Editor erfordert einen Aurora-DB-Cluster mit aktivierter RDS-Daten-API (Daten-API). Informationen zu DB-Clustern, die die Daten-API unterstützen, und zu deren Aktivierung finden Sie unter Verwenden der Daten-API von Amazon RDS. Das SQL, das Sie ausführen können, unterliegt den Einschränkungen der Daten-API. Weitere Informationen finden Sie unter the section called “Einschränkungen”.

Verfügbarkeit des Abfrage-Editors

Der Abfrage-Editor ist für Aurora-DB-Cluster verfügbar, die Aurora MySQL- und Aurora PostgreSQL-Engine-Versionen verwenden, die Data API unterstützen, und dort, AWS-Regionen wo Data API verfügbar ist. Weitere Informationen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für RDS-Daten-API.

Autorisieren des Zugriffs auf den Abfrage-Editor

Ein Benutzer muss zum Ausführen von Abfragen im Abfrage-Editor autorisiert sein. Sie können einen Benutzer autorisieren, Abfragen im Abfrage-Editor auszuführen, indem Sie diesem Benutzer die `AmazonRDSDataFullAccess` Richtlinie, eine vordefinierte Richtlinie AWS Identity and Access Management (IAM), hinzufügen.

> **Note**
>
> Vergewissern Sie sich, dass Sie beim Erstellen des IAM-Benutzers denselben Benutzernamen und dasselbe Passwort verwenden wie für den Datenbankbenutzer, z. B. den Namen und das Passwort des Administratorbenutzers. Weitere Informationen zum

Hinzufügen von Benutzern finden Sie unter Erstellen eines IAM-Benutzers in Ihrem AWS-Konto im AWS Identity and Access Management -Benutzerhandbuch.

Sie können auch eine IAM-Richtlinie erstellen die Zugriff auf den Abfrage-Editor gewährt. Nach dem Erstellen der Richtlinie können Sie diese jedem Benutzer hinzufügen, der Zugriff auf den Abfrage-Editor benötigt.

Die folgende Richtlinie bietet die mindestens erforderlichen Berechtigungen für einen Benutzer, um auf den Abfrage-Editor zuzugreifen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "QueryEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "tag:GetResources",
                "secretsmanager:CreateSecret",
                "secretsmanager:ListSecrets",
                "dbqms:CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms:DeleteFavoriteQueries",
```

```
                "dbqms:GetQueryString",
                "dbqms:CreateQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms:DeleteQueryHistory",
                "dbqms:DescribeQueryHistory",
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "*"
        }
    ]
}
```

Informationen zum Erstellen einer IAM-Richtlinie finden Sie unter Erstellen von IAM-Richtlinien im AWS Identity and Access Management -Benutzerhandbuch.

Informationen zum Hinzufügen einer IAM-Richtlinie zu einem Benutzer finden Sie im Abschnitt Hinzufügen und Entfernen von IAM-Identitätsberechtigungen im AWS Identity and Access Management -Benutzerhandbuch.

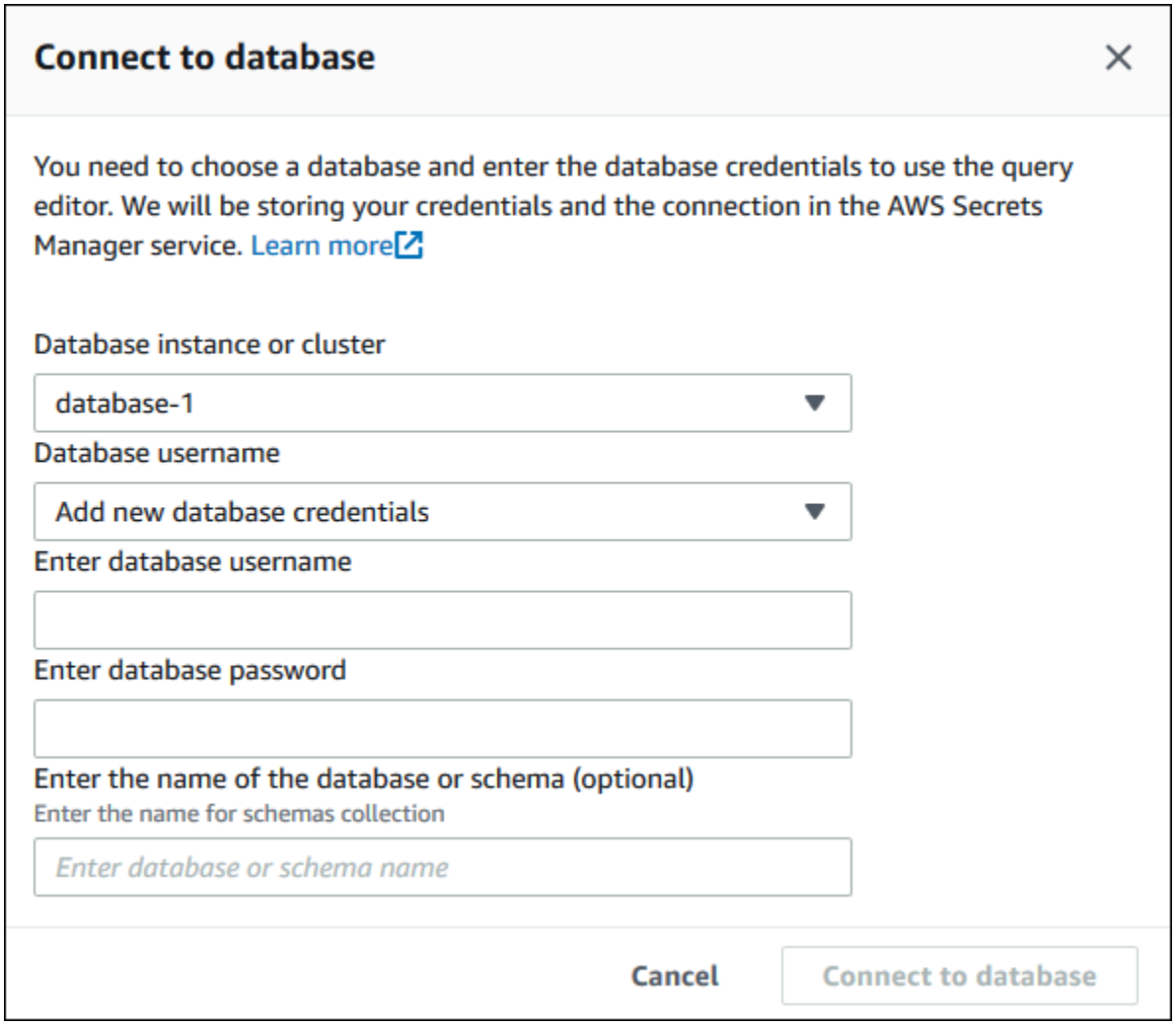
Ausführen von Abfragen im Abfrage-Editor

Sie können SQL-Anweisungen auf einem Aurora-DB-Cluster im Abfrage-Editor ausführen. Das SQL, das Sie ausführen können, unterliegt den Einschränkungen der Daten-API. Weitere Informationen finden Sie unter the section called “Einschränkungen”.

So führen Sie eine Abfrage im Abfrage-Editor aus

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der oberen rechten Ecke des den aus AWS-Managementkonsole, AWS-Region in dem Sie die Aurora-DB-Cluster erstellt haben, die Sie abfragen möchten.
3. Wählen Sie im Navigationsbereich Datenbanken aus.
4. Wählen Sie den Aurora-DB-Cluster aus, für den Sie SQL-Abfragen ausführen möchten.

5. Wählen Sie für Aktionen die Option Abfrage aus. Falls Sie noch keine Verbindung zur Datenbank hergestellt haben, wird die Seite Connect to database (Verbindung zur Datenbank herstellen) geöffnet.



6. Geben Sie die folgenden Informationen ein:

   a. Wählen Sie für Datenbank-Instance oder -Cluster den Aurora-DB-Cluster aus, für den Sie SQL-Abfragen ausführen möchten.

   b. Wählen Sie in Database username (Datenbank-Benutzername) den Benutzernamen des Benutzers der Datenbank aus, mit der Sie eine Verbindung herstellen möchten, oder wählen Sie Add new database credentials (Neue Datenbankanmeldeinformationen hinzufügen) aus. Wenn Sie Add new database credentials (Neue Datenbankanmeldeinformationen) auswählen, geben Sie den Benutzernamen für die neuen Datenbankanmeldeinformationen in Enter database username (Datenbank-Benutzername eingeben) ein.

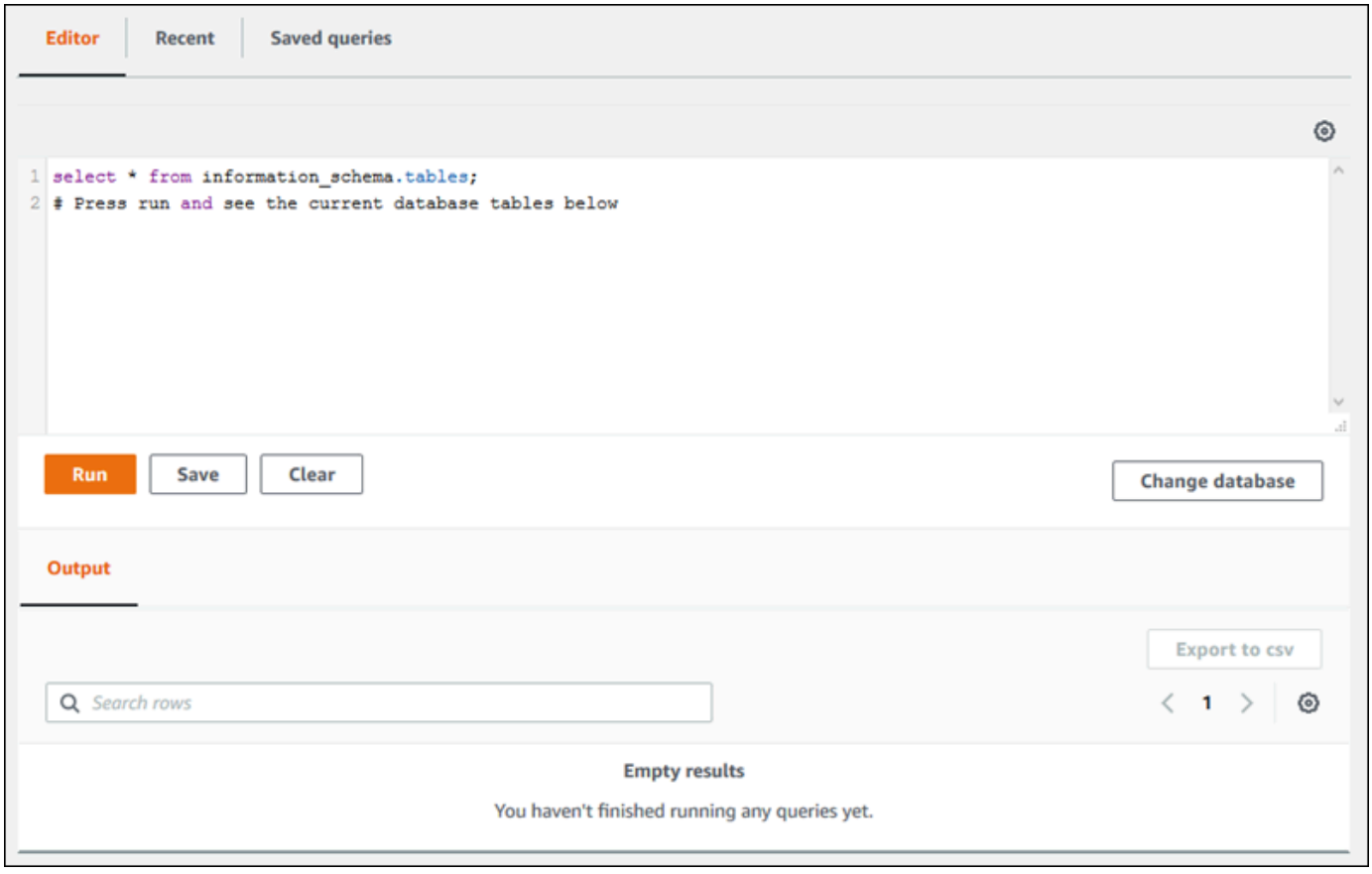
c. Geben Sie in Enter database password (Datenbankpasswort eingeben) das Passwort für den von Ihnen ausgewählten Datenbankbenutzer ein.

d. Geben Sie in das letzte Feld den Namen der Datenbank oder des Schemas ein, die bzw. das Sie für das Aurora-DB-Cluster verwenden möchten.

e. Wählen Sie Connect to database (Verbindung zur Datenbank herstellen).

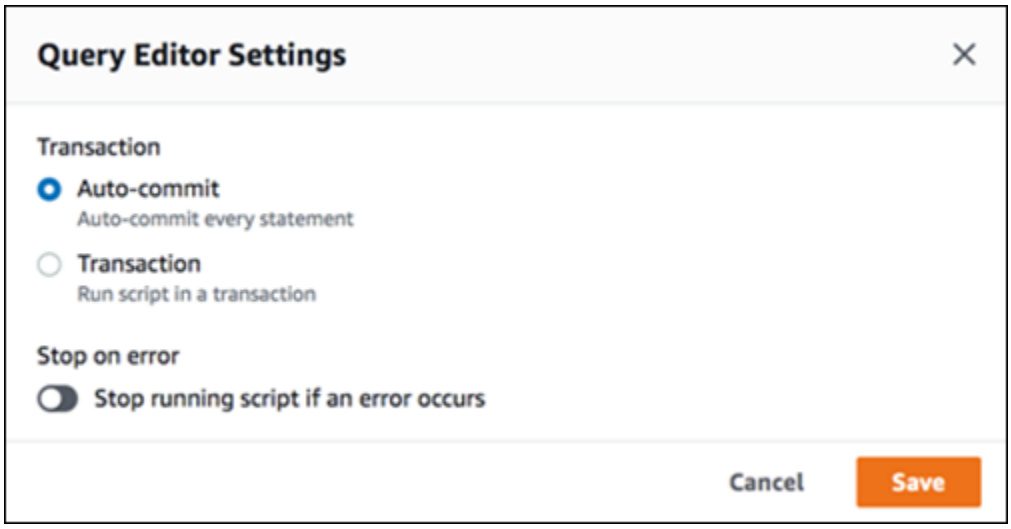
> **Note**
>
> Wenn Ihre Verbindung erfolgreich ist, werden Ihre Verbindungs- und Authentifizierungsinformationen in gespeicher AWS Secrets Manager. Sie müssen die Verbindungsinformationen nicht noch einmal eingeben.

7. Geben Sie im Abfrage-Editor die SQL-Abfrage ein, die Sie für die Datenbank ausführen möchten.



Jede SQL-Anweisung kann automatisch übergeben werden. Alternativ können Sie SQL-Anweisungen in einem Skript als Teil einer Transaktion ausführen. Um dieses Verhalten zu steuern, wählen Sie das Zahnradsymbol oberhalb des Abfragefensters aus.

Das Fenster Query Editor Settings (Einstellungen des Abfrage-Editors) wird angezeigt.



Wenn Sie Auto-commit (Automatisch übergeben) wählen, wird jede SQL-Anweisung automatisch übergeben. Wenn Sie Transaktion wählen, können Sie eine Gruppe von Anweisungen in einem Skript ausführen. Anweisungen werden automatisch am Ende des Skripts festgeschrieben, es sei denn, vorher wurde explizit festgeschrieben oder zurückgesetzt. Sie können ein Skript, das ausgeführt wird, bei Auftreten eines Fehlers auch anhalten, indem Sie Stop on error (Bei Fehler anhalten) aktivieren.

> **Note**
>
> In einer Gruppe von Anweisungen können Data Definition Language (DDL)-Anweisungen dazu führen, dass frühere Data Manipulation Language (DML)-Anweisungen übergeben werden. Sie können in einer Gruppe von Anweisungen in einem Skript auch `COMMIT`- und `ROLLBACK`-Anweisungen einfügen.

Wählen Sie nach Festlegen der Optionen im Fenster Query Editor Settings (Einstellungen des Abfrage-Editors) Save (Speichern) aus.

8. Wählen Sie Run (Ausführen) oder drücken Sie STRG+Eingabetaste. Der Abfrage-Editor zeigt dann die Ergebnisse Ihrer Abfrage an.

   Speichern Sie nach dem Ausführen der Abfrage diese in Saved Queries (Gespeicherte Abfragen) durch Auswahl von Save (Speichern).

   Sie können die Abfrageergebnisse durch Auswahl von Export to csv (Zu CSV exportieren) in ein Tabellenformat exportieren.

Sie können frühere Abfragen suchen, bearbeiten und erneut ausführen. Hierzu wählen Sie die Registerkarte Recent (Zuletzt) oder Saved queries (Gespeicherte Abfragen), den Abfragetext und anschließend Run (Ausführen) aus.

Um die Datenbank zu ändern, wählen Sie Change database (Datenbank ändern).

API-Referenz für Database Query Metadata Service (DBQMS)

Der Database Query Metadata Service (`dbqms`) ist ein reiner Dienst für interne Daten. Er stellt Ihre aktuellen und gespeicherten Abfragen für den Abfrage-Editor auf AWS-Managementkonsole für mehrere AWS-Services bereit, einschließlich Amazon RDS.

Unterstützte DBQMS-Aktionen

- CreateFavoriteQuery
- CreateQueryHistory
- CreateTab
- DeleteFavoriteQueries

- DeleteQueryHistory
- DeleteTab
- DescribeFavoriteQueries
- DescribeQueryHistory
- DescribeTabs
- GetQueryString
- UpdateFavoriteQuery
- UpdateQueryHistory
- UpdateTab

CreateFavoriteQuery

Speichern Sie eine neue bevorzugte Abfrage. Jeder -Benutzer kann bis zu 1000 gespeicherte Abfragen erstellen. Dieses Limit kann in Zukunft geändert werden.

CreateQueryHistory

Dient zum Speichern eines neuen Abfrageverlaufseintrags.

CreateTab

Speichern Sie eine neue Abfrage-Registerkarte. Jeder -Benutzer kann bis zu 10 Abfrage-Registerkarten erstellen.

DeleteFavoriteQueries

Löschen Sie eine oder mehrere gespeicherte Abfragen.

DeleteQueryHistory

Dient zum Löschen von Abfrageverlaufseinträgen.

DeleteTab

Löschen Sie Einträge der Abfrage-Registerkarte

DescribeFavoriteQueries

Listet gespeicherte Abfragen auf, die von einem Benutzer in einem bestimmten Konto erstellt wurden.

DescribeQueryHistory

Listet Abfrageverlaufseinträge auf.

DescribeTabs

Listet Abfrage-Registerkarten auf, die von einem Benutzer in einem bestimmten Konto erstellt wurden.

GetQueryString

Dient zum Abruf des vollständigen Abfragetexts über eine Abfrage-ID.

UpdateFavoriteQuery

Ermöglicht die Aktualisierung der Abfragezeichenfolge, der Beschreibung, des Namens oder des Ablaufdatums.

UpdateQueryHistory

Dient zum Aktualisieren des Status des Abfrageverlaufs.

UpdateTab

Aktualisieren Sie den Namen der Abfrage-Registerkarte und die Abfragezeichenfolge.

Verwenden von Amazon Aurora Machine Learning

Mit Amazon Aurora Machine Learning können Sie Ihre Aurora-DB-Cluster je nach Anforderungen in einen der folgenden Machine-Learning-Services von AWS integrieren. Sie alle unterstützen spezifische Anwendungsfälle für Machine Learning.

Amazon Bedrock

Amazon Bedrock ist ein vollständig verwalteter Dienst, der führende Basismodelle von KI-Unternehmen über eine API verfügbar macht und zudem Entwickler-Tools zum Erstellen und Skalieren von auf generativer KI basierenden Anwendungen bereitstellt. Bei Amazon Bedrock zahlen Sie für das Bilden von Inferenzen in allen Basismodellen von Drittanbietern. Die Preisgestaltung richtet sich nach der Menge der Ein- und Ausgabetoken und danach, ob Sie bereitgestellten Durchsatz für das Modell erworben haben. Weitere Informationen finden Sie unter Was ist Amazon Bedrock? im Amazon Bedrock-Benutzerhandbuch.

Amazon Comprehend

Amazon Comprehend ist ein verwalteter Dienst für die natürliche Sprachverarbeitung (NLP), der verwendet wird, um Informationen aus Dokumenten zu extrahieren. Mit Amazon Comprehend können Sie anhand des Inhalts von Dokumenten Stimmungen ableiten, indem Entitäten, Schlüsselphrasen, Sprache und andere Merkmale analysiert werden. Weitere Informationen finden Sie unter Was ist Amazon Comprehend? im Entwicklerhandbuch für Amazon Comprehend.

SageMaker AI

Amazon SageMaker AI ist ein vollständig verwalteter Machine-Learning-Service. Datenwissenschaftler und Entwickler verwenden Amazon SageMaker AI, um Machine-Learning-Modelle für eine Vielzahl von Inferenzaufgaben wie Betrugserkennung und Produktempfehlung zu erstellen, zu trainieren und zu testen. Wenn ein Machine-Learning-Modell für den Einsatz in der Produktion bereit ist, kann es in der von Amazon SageMaker AI gehosteten Umgebung bereitgestellt werden. Weitere Informationen finden Sie unter Was ist Amazon SageMaker AI? im Entwicklerhandbuch für Amazon SageMaker AI.

Die Verwendung von Amazon Comprehend mit Ihrem Aurora-DB-Cluster erfordert weniger Vorabeinstellungen als die Verwendung von SageMaker AI. Wenn Sie mit AWS-Machine-Learning nicht vertraut sind, empfehlen wir Ihnen, zunächst Amazon Comprehend zu erkunden.

Themen

- Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL
- Verwendung von Amazon Aurora Machine Learning mit Aurora PostgreSQL

Verwendung von Amazon Aurora Machine Learning mit Aurora MySQL

Wenn Sie Machine Learning von Amazon Aurora mit Ihrem DB-Cluster von Aurora MySQL verwenden, können Sie je nach Bedarf Amazon Bedrock, Amazon Comprehend oder Amazon SageMaker AI nutzen. Sie unterstützen jeweils unterschiedliche Anwendungsfälle für Machine Learning.

Inhalt

- Voraussetzungen für die Verwendung von Aurora Machine Learning mit Aurora MySQL
- Verfügbarkeit von Regionen und Versionen
- Unterstützte Funktionen und Einschränkungen von Aurora Machine Learning mit Aurora MySQL
- Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Aurora Machine Learning
  - Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon Bedrock
  - Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon Comprehend
  - Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von SageMaker AI
    - Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon S3 für SageMaker AI (optional)
  - Erteilen des Zugriffs auf Aurora Machine Learning für Datenbankbenutzer
    - Erteilen des Zugriffs auf Amazon-Bedrock-Funktionen
    - Erteilen des Zugriffs auf Amazon-Comprehend-Funktionen
    - Erteilen des Zugriffs auf SageMaker AI-Funktionen
- Verwenden von Amazon Bedrock mit Ihrem DB-Cluster von Aurora MySQL
- Verwenden von Amazon Comprehend mit Ihrem DB-Cluster von Aurora MySQL
- Verwenden von SageMaker AI mit Ihrem DB-Cluster von Aurora MySQL
  - Zeichensatzanforderungen für SageMaker AI-Funktionen, die Zeichenfolgen zurückgeben
  - Export von Daten nach Amazon S3 für das SageMaker AI-Modelltraining (erweitert)
- Leistungsaspekte zur Verwendung von Aurora Machine Learning mit Aurora MySQL
  - Modell und Prompt

  - Abfrage-Cache
  - Batch-Optimierung für Aurora-Machine-Learning-Funktionsaufrufe
- Überwachung von Aurora Machine Learning

Voraussetzungen für die Verwendung von Aurora Machine Learning mit Aurora MySQL

AWS Machine Learning Services sind verwaltete Services, die in ihren eigenen Produktionsumgebungen eingerichtet und ausgeführt werden. Aurora Machine Learning unterstützt die Integration in Amazon Bedrock, Amazon Comprehend, und SageMaker AI. Bevor Sie versuchen, Ihren DB-Cluster von Aurora MySQL für die Verwendung von Aurora Machine Learning einzurichten, stellen Sie sicher, dass Sie die folgenden Anforderungen und Voraussetzungen verstehen.

- Die Machine-Learning-Services müssen in derselben AWS-Region wie Ihr DB-Cluster von Aurora MySQL ausgeführt werden. Sie können keine Machine-Learning-Services aus einem DB-Cluster von Aurora MySQL in einer anderen Region verwenden.
- Wenn sich Ihr DB-Cluster von Aurora MySQL in einer anderen Virtual Public Cloud (VPC) als Ihr Amazon Bedrock-, Amazon Comprehend- oder SageMaker-KI-Service, muss die Sicherheitsgruppe der VPC ausgehende Verbindungen zum Ziel-Aurora-Machine-Learning-Service zulassen. Weitere Informationen finden Sie unter Steuern des Datenverkehrs zu den AWS-Ressourcen mithilfe von Sicherheitsgruppen im Amazon-VPC-Benutzerhandbuch.
- Sie können einen Aurora-Cluster, auf dem eine niedrigere Version von Aurora MySQL ausgeführt wird, auf eine unterstützte höhere Version aktualisieren, wenn Sie Aurora Machine Learning mit diesem Cluster verwenden möchten. Weitere Informationen finden Sie unter Datenbank-Engine-Updates für Amazon Aurora MySQL.
- Ihr DB-Cluster von Aurora MySQL muss eine benutzerdefinierte DB-Cluster-Parametergruppe verwenden. Am Ende des Einrichtungsvorgangs für jeden Aurora-Service für Machine Learning, den Sie verwenden möchten, fügen Sie den Amazon-Ressourcennamen (ARN) der zugehörigen IAM-Rolle hinzu, die für den Service erstellt wurde. Wir empfehlen Ihnen, vorab eine benutzerdefinierte DB-Cluster-Parametergruppe für Aurora MySQL zu erstellen und Ihren DB-Cluster von Aurora MySQL für ihre Verwendung zu konfigurieren, damit sie am Ende des Einrichtungsvorgangs von Ihnen geändert werden kann.
- Für SageMaker AI:
  - Die Machine-Learning-Komponenten, die Sie für Inferenzen verwenden möchten, müssen eingerichtet werden und einsatzbereit sein. Während des Konfigurationsprozesses für Ihren DB-

Cluster von Aurora MySQL muss der ARN des SageMaker AI-Endpunkts verfügbar sein. Die Datenwissenschaftler in Ihrem Team sind wahrscheinlich am besten in der Lage, mit SageMaker AI zu arbeiten, um die Modelle vorzubereiten und andere entsprechende Aufgaben zu erledigen. Informationen zu den ersten Schritten mit Amazon SageMaker AI finden Sie unter Erste Schritte mit Amazon SageMaker AI. Weitere Informationen zu Rückschlüssen und Endpunkten finden Sie unter Echtzeit-Inferenz.

- Wenn Sie SageMaker AI mit Ihren eigenen Trainingsdaten verwenden möchten, müssen Sie einen Amazon-S3-Bucket im Rahmen Ihrer Aurora-MySQL-Konfiguration für Aurora Machine Learning einrichten. Dazu folgen Sie dem gleichen allgemeinen Prozess wie beim Einrichten der SageMaker AI-Integration. Eine Zusammenfassung dieses optionalen Einrichtungsprozesses finden Sie unter Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon S3 für SageMaker AI (optional).

- Für globale Aurora-Datenbanken richten Sie die Aurora-Dienste für Machine Learning ein, die Sie in allen AWS-Regionen Ihrer globalen Aurora-Datenbank verwenden möchten. Wenn Sie beispielsweise Aurora Machine Learning mit SageMaker AI für Ihre globale Aurora-Datenbank verwenden möchten, gehen Sie für jeden DB-Cluster von Aurora MySQL in jeder AWS-Region wie folgt vor.
  - Richten Sie die Amazon-SageMaker AI-Services mit denselben SageMaker AI-Trainingsmodellen und -Endpunkten ein. Diese müssen auch dieselben Namen tragen.
  - Erstellen Sie die IAM-Rollen wie unter Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Aurora Machine Learning beschrieben.
  - Fügen Sie den ARN der IAM-Rolle der benutzerdefinierten DB-Cluster-Parametergruppe für jeden DB-Cluster von Aurora MySQL in jeder AWS-Region hinzu.

  Diese Aufgaben erfordern, dass Aurora Machine Learning für Ihre Version von Aurora MySQL in allen AWS-Regionen Ihrer globalen Aurora-Datenbank verfügbar ist.

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Aurora-Datenbank-Engines und in allen AWS-Regionen.

- Informationen zur Verfügbarkeit von Versionen und Regionen von Amazon Comprehend und Amazon SageMaker AI mit Aurora MySQL finden Sie unter Aurora Machine Learning mit Aurora MySQL.
- Amazon Bedrock wird nur auf Aurora MySQL Version 3.06 und höher unterstützt.

Informationen zur Verfügbarkeit von Regionen für Amazon Bedrock finden Sie unter Modell-Support nach AWS-Region im Amazon-Bedrock-Benutzerhandbuch.

Unterstützte Funktionen und Einschränkungen von Aurora Machine Learning mit Aurora MySQL

Bei der Verwendung von Aurora MySQL mit Aurora Machine Learning gelten die folgenden Einschränkungen:

- Die Aurora-Erweiterung für Machine Learning unterstützt keine Vektorschnittstellen.
- Integrationen von Aurora Machine Learning werden nicht unterstützt, wenn sie in einem Auslöser verwendet werden.
- Funktionen für Aurora Machine Learning sind mit der Binärprotokollreplikation (Binlog) nicht kompatibel.
  - Die Einstellung `--binlog-format=STATEMENT` führt beim Aufrufen von Aurora Machine Learning-Funktionen zu einer Ausnahme.
  - Aurora-Funktionen für Machine Learning sind nichtdeterministisch und nichtdeterministische gespeicherte Funktionen sind mit diesem Binlog-Format nicht kompatibel.

  Weitere Informationen finden Sie unter Gemischte Binärprotokollformate in der MySQL-Dokumentation.
- Gespeicherte Funktionen, die Tabellen mit „Generated-always“-Spalten aufrufen, werden nicht unterstützt. Dies gilt für jede gespeicherte Aurora-MySQL-Funktion. Weitere Informationen zu diesem Spaltentyp finden Sie unter CREATE TABLE und „Generated“-Spalten in der MySQL-Dokumentation.
- Amazon Bedrock-Funktionen unterstützen `RETURNS JSON` nicht. Sie können `CONVERT` oder `CAST` verwenden, um bei Bedarf von `TEXT` zu `JSON` zu konvertieren.
- Amazon Bedrock unterstützt keine Batch-Anforderungen.
- Aurora MySQL unterstützt jeden SageMaker AI-Endpunkt, der das CSV-Format (Comma Separated Value) lesen und schreiben kann, über den `ContentType text/csv`. Dieses Format wird von den folgenden integrierten SageMaker AI-Algorithmen akzeptiert:
  - Lineares Lernen
  - Random Cut Forest
  - XGBoost

Weitere Informationen zu diesen Algorithmen finden Sie unter Auswählen eines Algorithmus im Entwicklerhandbuch für Amazon SageMaker AI.

Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Aurora Machine Learning

In den folgenden Themen finden Sie separate Einrichtungsverfahren für jeden dieser Aurora-Dienste für Machine Learning.

Themen

- Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon Bedrock
- Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon Comprehend
- Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von SageMaker AI
  - Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon S3 für SageMaker AI (optional)
- Erteilen des Zugriffs auf Aurora Machine Learning für Datenbankbenutzer
  - Erteilen des Zugriffs auf Amazon-Bedrock-Funktionen
  - Erteilen des Zugriffs auf Amazon-Comprehend-Funktionen
  - Erteilen des Zugriffs auf SageMaker AI-Funktionen

Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon Bedrock

Aurora Machine Learning basiert auf Rollen und Richtlinien von AWS Identity and Access Management, damit Ihr DB-Cluster von Aurora MySQL auf die Amazon-Bedrock-Services zugreifen und diese nutzen kann. Die folgenden Verfahren erstellen eine IAM-Berechtigungsrichtlinie und -rolle, sodass Ihr DB-Cluster in Amazon Bedrock integriert werden kann.

So erstellen Sie die -IAM-Richtlinie

1. Melden Sie sich bei der AWS-Managementkonsole an, und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich Richtlinien aus.
3. Klicken Sie auf Create a policy (Richtlinie erstellen).

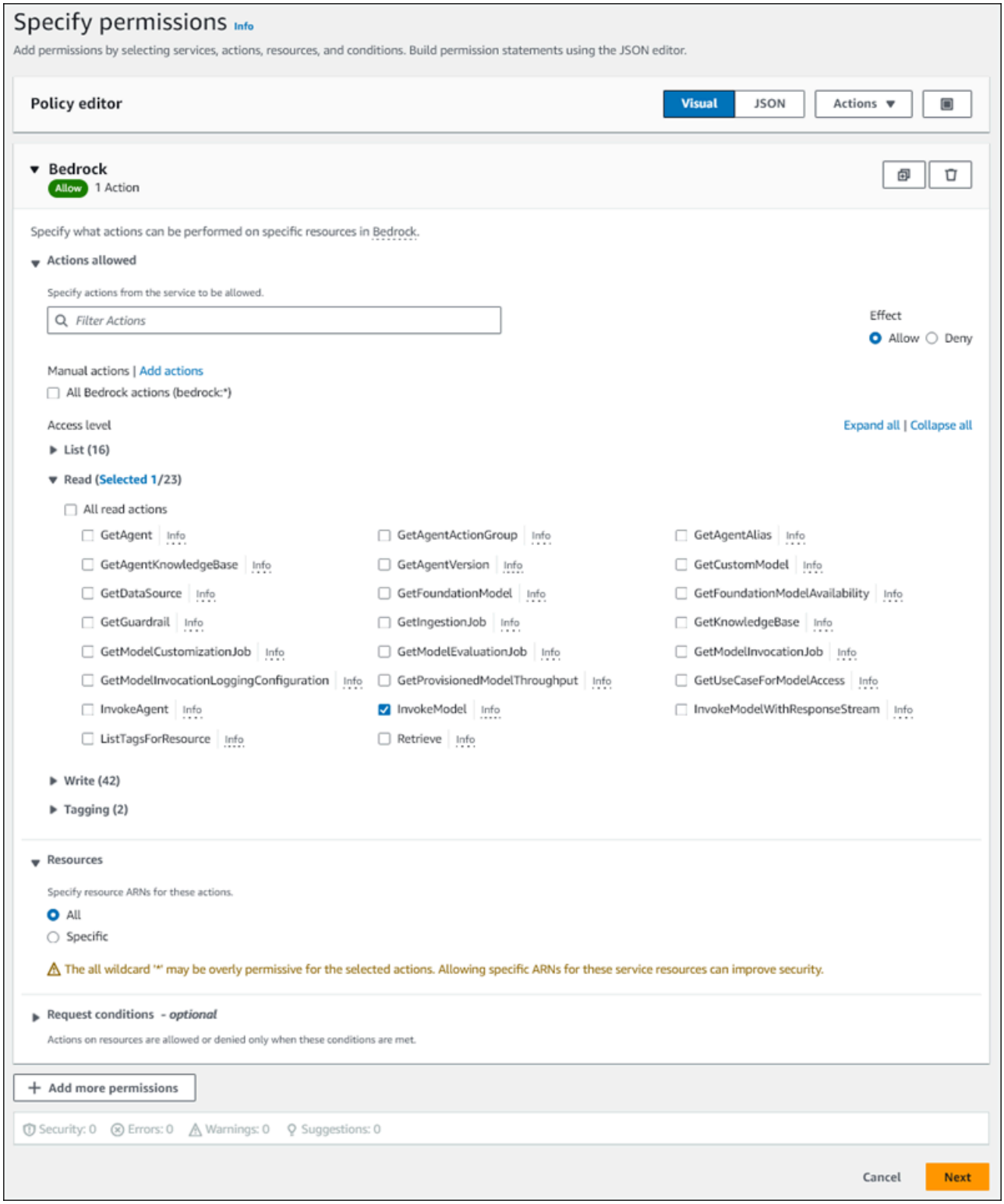
4. Wählen Sie auf der Seite Berechtigungen festlegen für Service auswählen die Option Bedrock aus.

   Die Amazon-Bedrock-Berechtigungen werden angezeigt.

5. Erweitern Sie Read und wählen Sie dann InvokeModel aus.

6. Wählen Sie für Ressourcen die Option Alle aus.

   Die Seite Berechtigungen angeben sollte wie die folgende Abbildung aussehen.



7. Wählen Sie Weiter aus.
8. Geben Sie auf der Seite Überprüfen und erstellen einen Namen für Ihre Richtlinie ein, zum Beispiel **`BedrockInvokeModel`**.

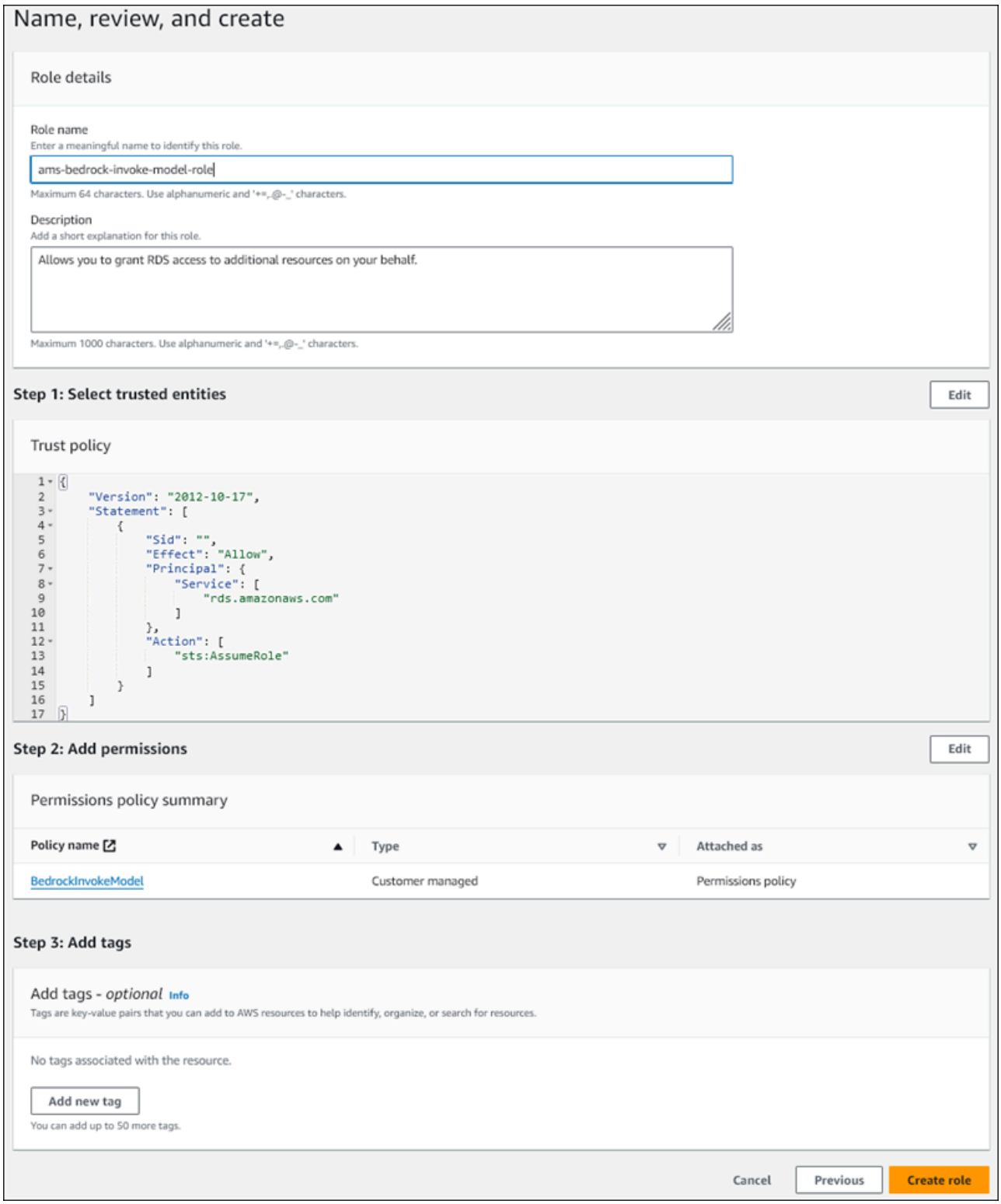
9. Überprüfen Sie Ihre Richtlinie und klicken Sie dann auf Richtlinie erstellen.

Als Nächstes erstellen Sie die IAM-Rolle, die die Amazon-Bedrock-Berechtigungsrichtlinie verwendet.

So erstellen Sie die IAM-Rolle

1. Melden Sie sich bei der AWS-Managementkonsole an, und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich Roles (Rollen) aus.
3. Wählen Sie Rolle erstellen aus.
4. Wählen Sie auf der Seite Vertrauenswürdige Entitäten auswählen für Anwendungsfall die Option RDS aus.
5. Wählen Sie RDS – Rolle zur Datenbank hinzufügen und dann Weiter aus.
6. Wählen Sie auf der Seite Berechtigungen hinzufügen für Berechtigungsrichtlinien die von Ihnen erstellte IAM-Richtlinie und dann Weiter aus.
7. Geben Sie auf der Seite Benennen, überprüfen und erstellen einen Namen für Ihre Rolle ein, zum Beispiel **`ams-bedrock-invoke-model-role`**.

   Die Rolle sollte in etwa wie die folgende Abbildung aussehen.



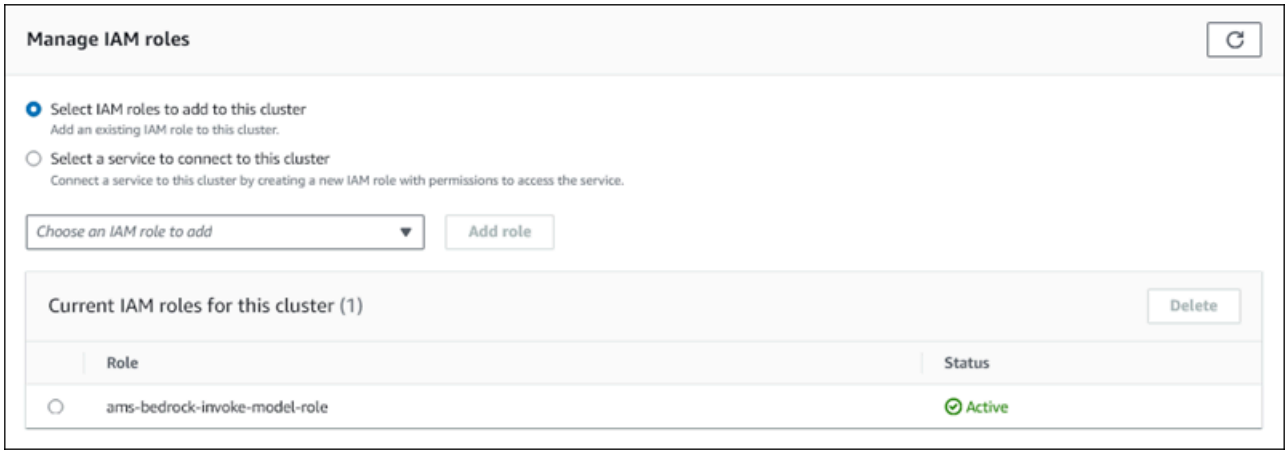
8. Überprüfen Sie die Rolle und wählen Sie dann Rolle erstellen aus.

Als Nächstes ordnen Sie Ihrem DB-Cluster die Amazon-Bedrock-IAM-Rolle zu.

So weisen Sie Ihrem Cluster die IAM-Rolle zu

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie aus dem Navigationsbereich Datenbanken aus.
3. Wählen Sie den DB-Cluster von Aurora MySQL aus, den Sie mit Amazon-Bedrock-Services verbinden möchten.
4. Wählen Sie die Registerkarte Connectivity & security (Konnektivität und Sicherheit).
5. Wählen Sie im Abschnitt IAM-Rollen verwalten die Option IAM-Rollen auswählen, die diesem Cluster hinzugefügt werden sollen aus.
6. Wählen Sie die von Ihnen erstellte IAM-Rolle und dann Rolle hinzufügen aus.

   Die IAM-Rolle ist Ihrem DB-Cluster zugeordnet, zunächst mit dem Status Ausstehend, dann mit dem Status Aktiv. Wenn der Vorgang abgeschlossen ist, wird die Rolle in der Liste Current IAM roles for this cluster (Aktuelle IAM-Rollen für diesen Cluster) angezeigt.



Sie müssen die ARN dieser IAM-Rolle dem Parameter `aws_default_bedrock_role` der benutzerdefinierten DB-Cluster-Parametergruppe hinzufügen, die mit dem DB-Cluster von Aurora MySQL verknüpft ist. Wenn Ihr DB-Cluster von Aurora MySQL keine benutzerdefinierte DB-Cluster-Parametergruppe verwendet, müssen Sie eine erstellen, die Sie mit Ihrem DB-Cluster von Aurora MySQL nutzen können, um die Integration abzuschließen. Weitere Informationen finden Sie unter DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster.

So konfigurieren Sie den DB-Cluster-Parameter

1. Öffnen Sie in der Amazon-RDS-Konsole die Registerkarte Configuration (Konfiguration) Ihres DB-Clusters von Aurora MySQL.

2. Suchen Sie die DB-Cluster-Parametergruppe, die für Ihren Cluster konfiguriert ist. Wählen Sie den Link zum Öffnen Ihrer benutzerdefinierten DB-Cluster-Parametergruppe und dann Bearbeiten aus.
3. Suchen Sie den `aws_default_bedrock_role`-Parameter in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe.
4. Geben Sie im Feld Wert den ARN der IAM-Rolle ein.
5. Wählen Sie zum Speichern der Einstellung Save Changes (Änderungen speichern) aus.
6. Starten Sie die primäre Instance Ihres DB-Clusters von Aurora MySQL neu, damit diese Parametereinstellung wirksam wird.

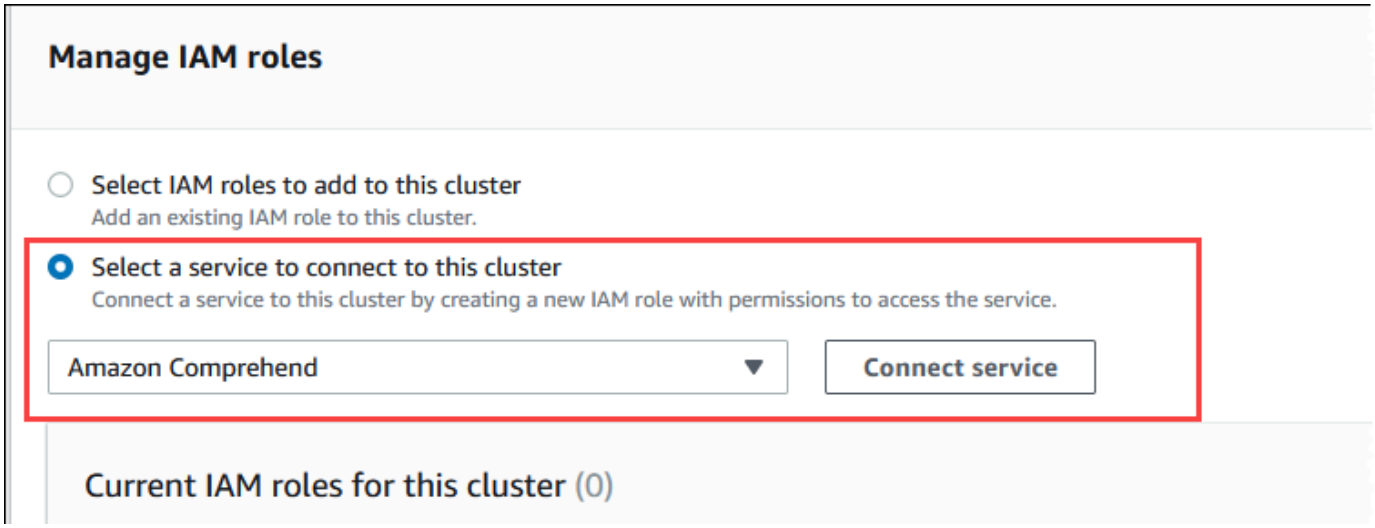
Die IAM-Integration für Amazon Bedrock ist abgeschlossen. Fahren Sie mit der Einrichtung Ihres DB-Clusters von Aurora MySQL für den Einsatz von Amazon Bedrock durch Erteilen des Zugriffs auf Aurora Machine Learning für Datenbankbenutzer fort.

Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon Comprehend

Aurora Machine Learning basiert auf Rollen und Richtlinien von AWS Identity and Access Management, damit Ihr DB-Cluster von Aurora MySQL auf die Amazon-Comprehend-Services zugreifen und diese nutzen kann. Mit dem folgenden Verfahren werden automatisch eine IAM-Rolle und -Richtlinie für Ihren Cluster erstellt, sodass dieser Amazon Comprehend verwenden kann.

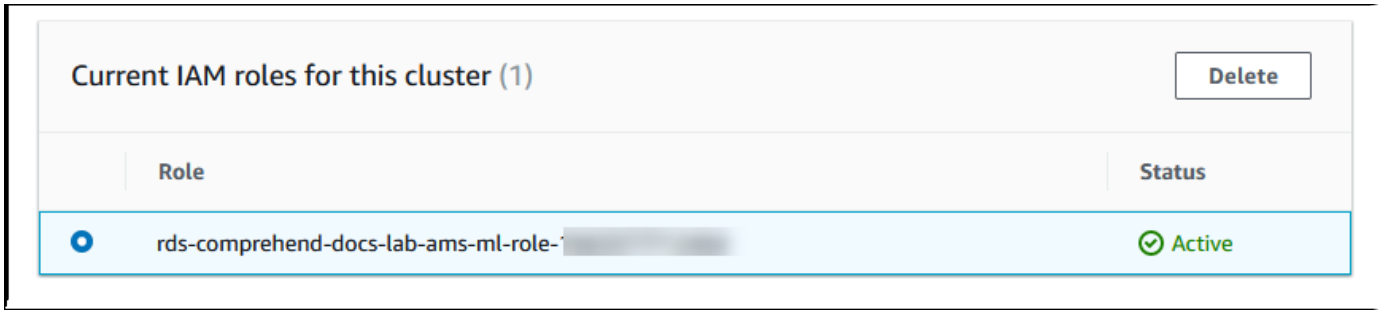
So richten Sie Ihren DB-Cluster von Aurora MySQL zur Verwendung von Amazon Comprehend ein

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie aus dem Navigationsbereich Datenbanken aus.
3. Wählen Sie den DB-Cluster von Aurora MySQL aus, den Sie mit Amazon-Comprehend-Services verbinden möchten.
4. Wählen Sie die Registerkarte Connectivity & security (Konnektivität und Sicherheit).
5. Wählen im Abschnitt IAM-Rollen verwalten die Option Einen Service zum Herstellen einer Verbindung mit diesem Cluster auswählen aus.
6. Wählen Sie Amazon Comprehend aus dem Menü und dann Service verbinden aus.



7. Für das Dialogfeld Connect cluster to Amazon Comprehend (Cluster mit Amazon Comprehend verbinden) sind keine zusätzlichen Informationen erforderlich. Möglicherweise wird jedoch eine Meldung mit dem Hinweis angezeigt, dass sich die Integration von Aurora und Amazon Comprehend derzeit in der Vorschau befindet. Lesen Sie die Nachricht unbedingt, bevor Sie fortfahren. Sie können Abbrechen auswählen, wenn Sie nicht fortfahren möchten.
8. Wählen Sie Connect service (Service verbinden) aus, um den Integrationsvorgang abzuschließen.

   Aurora erstellt die IAM-Rolle. Außerdem wird die Richtlinie erstellt, die dem Aurora-MySQL-DB-Cluster die Nutzung der Amazon-Comprehend-Services ermöglicht, und die Richtlinie der Rolle angefügt. Wenn der Vorgang abgeschlossen ist, wird die Rolle in der Liste Current IAM roles for this cluster (Aktuelle IAM-Rollen für diesen Cluster) angezeigt, wie in der folgenden Abbildung dargestellt.



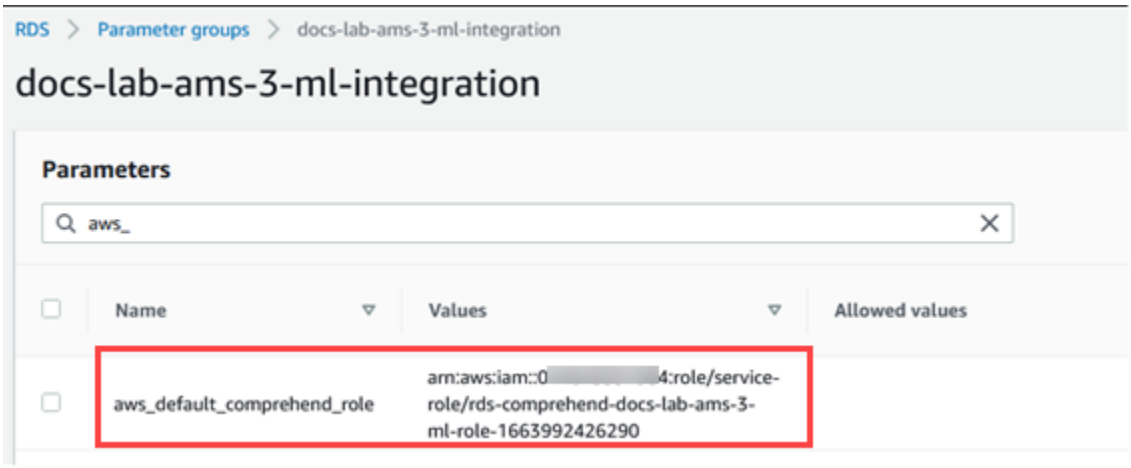
   Sie müssen die ARN dieser IAM-Rolle dem Parameter `aws_default_comprehend_role` der benutzerdefinierten DB-Cluster-Parametergruppe hinzufügen, die mit dem DB-Cluster von Aurora MySQL verknüpft ist. Wenn Ihr DB-Cluster von Aurora MySQL keine benutzerdefinierte DB-Cluster-Parametergruppe verwendet, müssen Sie eine erstellen, die Sie mit Ihrem DB-

Cluster von Aurora MySQL nutzen können, um die Integration abzuschließen. Weitere Informationen finden Sie unter DB-Cluster-Parametergruppen für Amazon-Aurora-DB-Cluster.

Nachdem Sie Ihre benutzerdefinierte DB-Cluster-Parametergruppe erstellt und mit Ihrem DB-Cluster von Aurora MySQL verknüpft haben, können Sie mit den folgenden Schritten fortfahren.

Wenn Ihr Cluster eine benutzerdefinierte DB-Cluster-Parametergruppe verwendet, gehen Sie wie folgt vor.

a. Öffnen Sie in der Amazon-RDS-Konsole die Registerkarte Configuration (Konfiguration) Ihres DB-Clusters von Aurora MySQL.
b. Suchen Sie die DB-Cluster-Parametergruppe, die für Ihren Cluster konfiguriert ist. Wählen Sie den Link zum Öffnen Ihrer benutzerdefinierten DB-Cluster-Parametergruppe und dann Bearbeiten aus.
c. Suchen Sie den `aws_default_comprehend_role`-Parameter in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe.
d. Geben Sie im Feld Wert den ARN der IAM-Rolle ein.
e. Wählen Sie zum Speichern der Einstellung Save Changes (Änderungen speichern) aus. Die folgende Abbildung zeigt ein Beispiel.

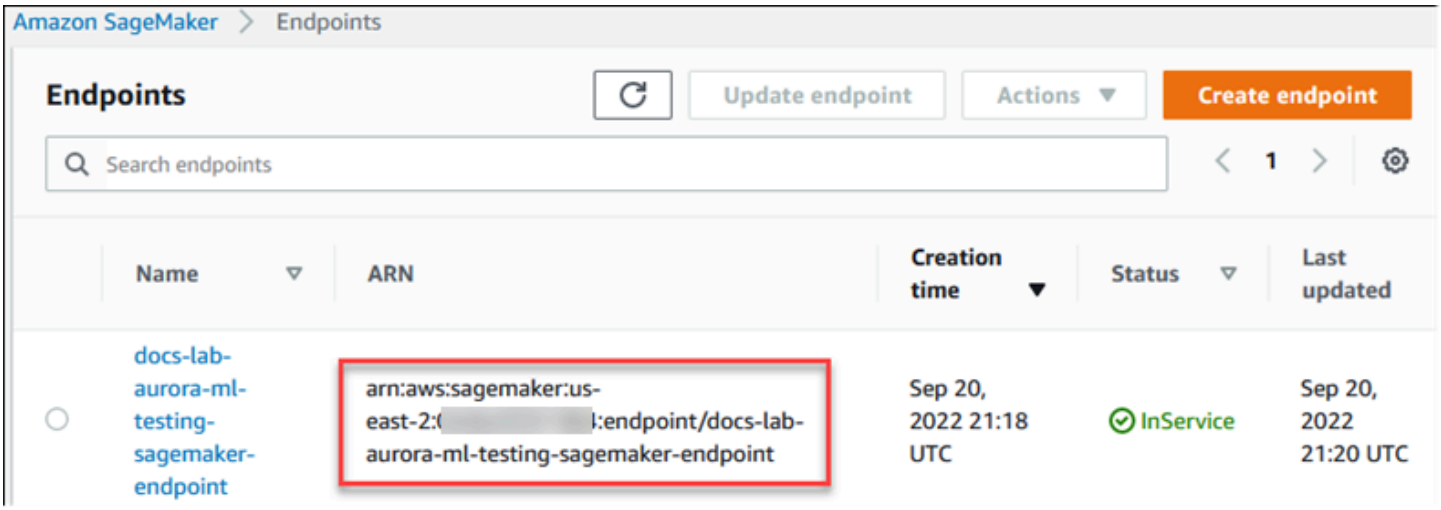


Starten Sie die primäre Instance Ihres DB-Clusters von Aurora MySQL neu, damit diese Parametereinstellung wirksam wird.

Die IAM-Integration für Amazon Comprehend ist abgeschlossen. Fahren Sie mit der Einrichtung Ihres DB-Clusters von Aurora MySQL zur Zusammenarbeit mit Amazon Comprehend fort, indem Sie den entsprechenden Datenbankbenutzern Zugriff gewähren.

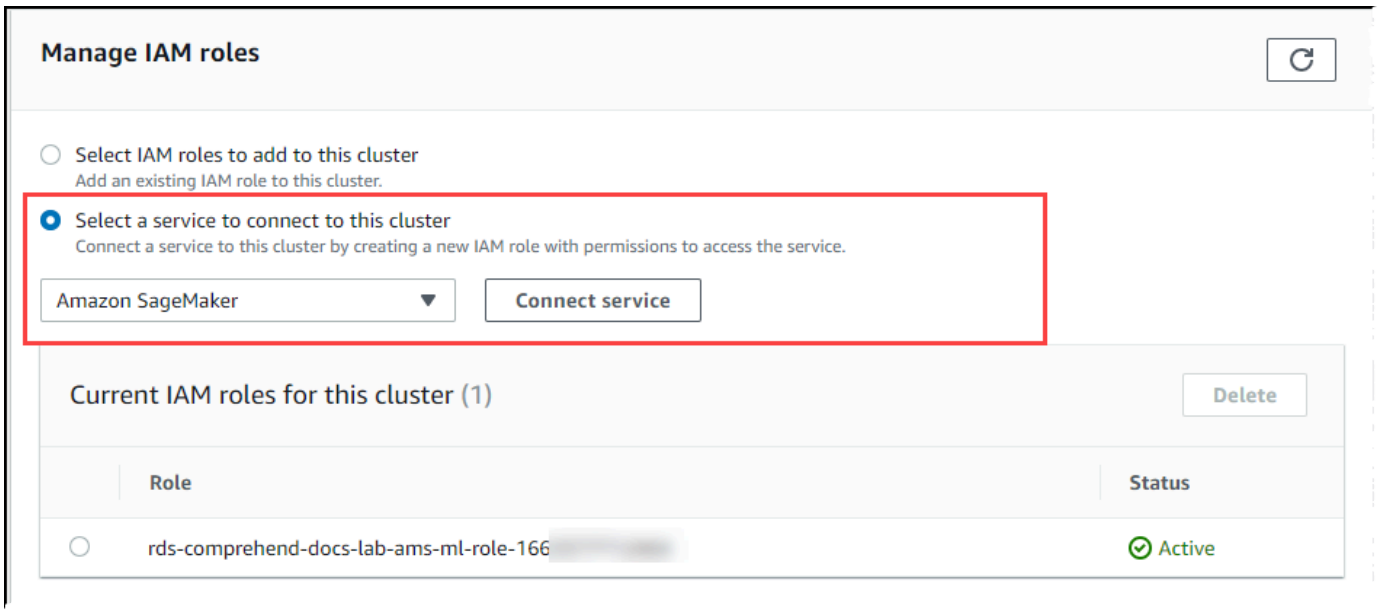
Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von SageMaker AI

Mit dem folgenden Verfahren werden automatisch eine IAM-Rolle und -Richtlinie für Ihren DB-Cluster von Aurora MySQL erstellt, sodass dieser SageMaker AI verwenden kann. Bevor Sie versuchen, dieses Verfahren auszuführen, stellen Sie sicher, dass der SageMaker AI-Endpunkt verfügbar ist, damit Sie ihn bei Bedarf eingeben können. In der Regel ist es Aufgabe der Datenwissenschaftler in Ihrem Team, einen Endpunkt zu erstellen, den Sie von Ihrem DB-Cluster von Aurora MySQL aus verwenden können. Sie finden solche Endpunkte in der SageMaker AI-Konsole. Öffnen Sie im Navigationsbereich das Menü Inference (Inferenz) und wählen Sie Endpoints (Endpunkte) aus. Die folgende Abbildung zeigt ein Beispiel.

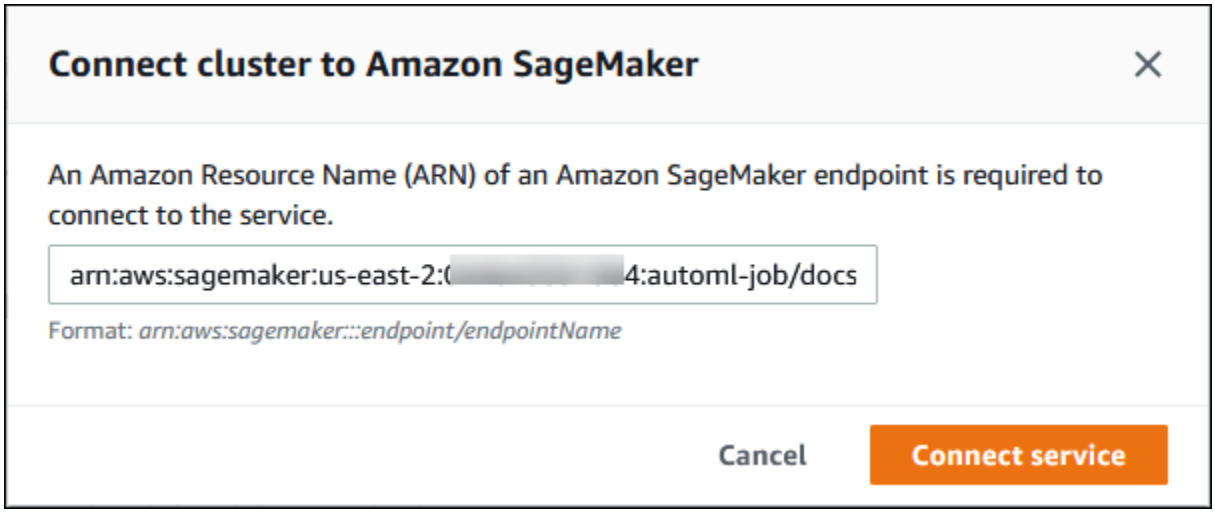


So richten Sie Ihren DB-Cluster von Aurora MySQL zur Verwendung von SageMaker AI ein

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie Databases (Datenbanken) aus dem Amazon-RDS-Navigationsmenü und dann den DB-Cluster von Aurora MySQL aus, den Sie mit SageMaker AI-Services verbinden möchten.
3. Wählen Sie die Registerkarte Connectivity & security (Konnektivität und Sicherheit).
4. Wählen Sie Select a service to connect to this cluster (Auswahl eines Services zur Verbindung mit diesem Cluster) im Abschnitt Manage IAM roles (IAM-Rollen verwalten) aus. Wählen Sie SageMaker AI in der Auswahl aus.



5. Wählen Sie Connect Service (Service verbinden).
6. Geben Sie im Dialogfeld Cluster mit SageMaker AI verbinden den ARN des SageMaker AI-Endpunkts ein.



7. Aurora erstellt die IAM-Rolle. Außerdem wird die Richtlinie erstellt, die dem DB-Cluster von Aurora MySQL die Nutzung der SageMaker AI-Services ermöglicht, und die Richtlinie der Rolle angefügt. Wenn der Vorgang abgeschlossen ist, wird die Rolle in der Liste Current IAM roles for this cluster (Aktuelle IAM-Rollen für diesen Cluster) angezeigt.
8. Öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
9. Wählen Sie Roles (Rollen) aus dem Abschnitt „Access management“ (Zugriffsverwaltung) des Navigationsmenüs von AWS Identity and Access Management aus.
10. Suchen Sie die Rolle in der Liste. Der Name verwendet das folgende Format.

```
rds-sagemaker-your-cluster-name-role-auto-generated-digits
```

11. Öffnen Sie die Seite „Summary“ (Zusammenfassung) der Rolle und suchen Sie den ARN. Notieren Sie sich den ARN oder kopieren Sie ihn mit dem Widget „Copy“ (Kopieren).
12. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
13. Wählen Sie Ihren DB-Cluster von Aurora MySQL und dann die Registerkarte Configuration (Konfiguration) aus.
14. Suchen Sie die DB-Cluster-Parametergruppe und wählen Sie den Link zum Öffnen Ihrer benutzerdefinierten DB-Cluster-Parametergruppe aus. Suchen Sie den `aws_default_sagemaker_role`-Parameter und geben Sie den ARN der IAM-Rolle in das Feld „Value“ (Wert) ein. Speichern Sie die Einstellung mit „Save“ (Speichern).
15. Starten Sie die primäre Instance Ihres DB-Clusters von Aurora MySQL neu, damit diese Parametereinstellung wirksam wird.

Die IAM-Einrichtung ist damit abgeschlossen. Fahren Sie mit der Einrichtung Ihres DB-Clusters von Aurora MySQL zur Zusammenarbeit mit SageMaker AI fort, indem Sie den entsprechenden Datenbankbenutzern Zugriff gewähren.

Wenn Sie Ihre SageMaker AI-Modelle für das Training verwenden möchten, anstatt vordefinierte SageMaker AI-Komponenten zu nutzen, müssen Sie Ihrem DB-Cluster von Aurora MySQL auch den Amazon-S3-Bucket hinzufügen, wie im folgenden Abschnitt Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon S3 für SageMaker AI (optional) beschrieben.

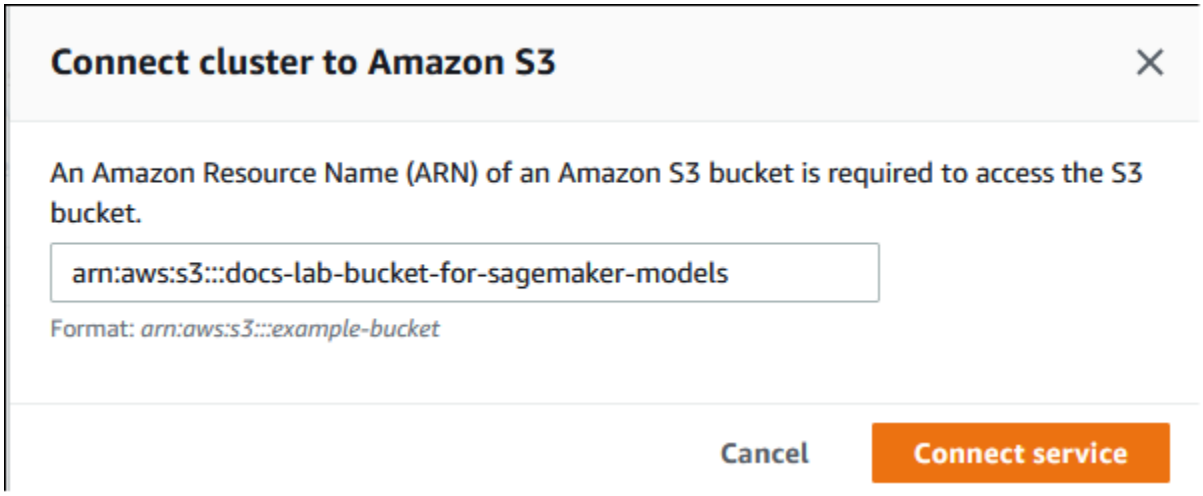
Einrichten Ihres DB-Clusters von Aurora MySQL zur Verwendung von Amazon S3 für SageMaker AI (optional)

Wenn Sie SageMaker AI mit Ihren eigenen Modellen verwenden möchten, anstatt die von SageMaker AI bereitgestellten vordefinierten Komponenten zu verwenden, müssen Sie einen Amazon-S3-Bucket für den DB-Cluster von Aurora MySQL einrichten. Weitere Informationen zum Erstellen eines Amazon-S3-Buckets finden Sie unter Erstellen von Buckets im Handbuch für Amazon Simple Storage Service.

So richten Sie Ihren DB-Cluster von Aurora MySQL zur Verwendung eines Amazon-S3-Buckets für SageMaker AI ein

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie Databases (Datenbanken) aus dem Amazon-RDS-Navigationsmenü und dann den DB-Cluster von Aurora MySQL aus, den Sie mit SageMaker AI-Services verbinden möchten.
3. Wählen Sie die Registerkarte Connectivity & security (Konnektivität und Sicherheit).
4. Wählen Sie Select a service to connect to this cluster (Auswahl eines Services zur Verbindung mit diesem Cluster) im Abschnitt Manage IAM roles (IAM-Rollen verwalten) aus. Wählen Sie Amazon S3 in der Auswahl aus.
5. Wählen Sie Connect Service (Service verbinden).
6. Geben Sie im Dialogfeld Cluster mit Amazon S3 verbinden den ARN des Amazon-S3-Buckets ein, wie in der folgenden Abbildung dargestellt.



7. Wählen Sie Connect service (Service verbinden) aus, um den Integrationsvorgang abzuschließen.

Weitere Informationen zur Verwendung von Amazon-S3-Buckets mit SageMaker AI finden Sie unter Angeben eines Amazon-S3-Buckets zum Hochladen von Trainingsdatensätzen und Speichern von Ausgabedaten im Entwicklerhandbuch für Amazon SageMaker AI. Weitere Informationen zum Arbeiten mit SageMaker AI finden Sie unter Erste Schritte mit Notebook-Instances von Amazon SageMaker AI im Entwicklerhandbuch für Amazon SageMaker AI.

Erteilen des Zugriffs auf Aurora Machine Learning für Datenbankbenutzer

Datenbankbenutzern muss die Berechtigung zum Aufrufen von Aurora-Machine-Learning-Funktionen erteilt werden. Wie Sie Berechtigungen erteilen, hängt von der MySQL-Version ab, die Sie für Ihren DB-Cluster von Aurora MySQL verwenden, wie im Folgenden beschrieben. Wie Sie diesen Vorgang ausführen, hängt von der MySQL-Version ab, die Ihr DB-Cluster von Aurora MySQL verwendet.

- Für Aurora-MySQL-Version 3 (mit MySQL 8.0 kompatibel) muss Datenbankbenutzern die entsprechende Datenbankrolle zugewiesen werden. Weitere Informationen finden Sie unter Verwenden von Rollen im MySQL 8.0-Referenzhandbuch.
- Für Aurora-MySQL-Version 2 (mit MySQL 5.7 kompatibel) werden Datenbankbenutzern Berechtigungen erteilt. Weitere Informationen finden Sie unter Access Control and Account Management im MySQL-5.7-Referenzhandbuch.

Die folgende Tabelle zeigt die Rollen und Berechtigungen, die Datenbankbenutzer benötigen, um mit Funktionen für Machine Learning zu arbeiten.

Aurora-MySQL-Version 3 (Rolle)	Aurora MySQL Version 2 (Berechtigung)
AWS_BEDROCK_ACCESS	–
AWS_COMPREHEND_ACCESS	INVOKE COMPREHEND
AWS_SAGEMAKER_ACCESS	INVOKE SAGEMAKER

Erteilen des Zugriffs auf Amazon-Bedrock-Funktionen

Wenn Sie Datenbankbenutzern Zugriff auf Amazon-Bedrock-Funktionen erteilen möchten, verwenden Sie die folgende SQL-Anweisung:

```
GRANT AWS_BEDROCK_ACCESS TO user@domain-or-ip-address;
```

Datenbankbenutzern müssen außerdem EXECUTE-Berechtigungen für die Funktionen erteilt werden, die Sie für die Arbeit mit Amazon Bedrock erstellen.

```
GRANT EXECUTE ON FUNCTION database_name.function_name TO user@domain-or-ip-address;
```

Schließlich müssen die Rollen der Datenbankbenutzer auf AWS_BEDROCK_ACCESS festgelegt sein:

```
SET ROLE AWS_BEDROCK_ACCESS;
```

Die Amazon-Bedrock-Funktionen sind jetzt verfügbar.

Erteilen des Zugriffs auf Amazon-Comprehend-Funktionen

Wenn Sie Datenbankbenutzern Zugriff auf Amazon-Comprehend-Funktionen erteilen möchten, verwenden Sie die entsprechende Anweisung für Ihre Aurora-MySQL-Version.

- Aurora-MySQL-Version 3 (mit MySQL 8.0 kompatibel)

```
GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address;
```

- Aurora-MySQL-Version 2 (mit MySQL 5.7 kompatibel)

```
GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address;
```

Die Amazon-Comprehend-Funktionen sind jetzt verfügbar. Anwendungsbeispiele finden Sie unter Verwenden von Amazon Comprehend mit Ihrem DB-Cluster von Aurora MySQL.

Erteilen des Zugriffs auf SageMaker AI-Funktionen

Wenn Sie Datenbankbenutzern Zugriff auf SageMaker AI-Funktionen erteilen möchten, verwenden Sie die entsprechende Anweisung für Ihre Aurora-MySQL-Version.

- Aurora-MySQL-Version 3 (mit MySQL 8.0 kompatibel)

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address;
```

- Aurora-MySQL-Version 2 (mit MySQL 5.7 kompatibel)

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address;
```

Datenbankbenutzern müssen außerdem EXECUTE-Berechtigungen für die Funktionen erteilt werden, die Sie für die Arbeit mit SageMaker AI erstellen. Angenommen, Sie haben zwei Funktionen erstellt, `db1.anomoly_score` und `db2.company_forecasts`, um die Dienste Ihres SageMaker AI-Endpunkts aufzurufen. Sie gewähren Ausführungsberechtigungen wie im folgenden Beispiel dargestellt.

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1;
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2;
```

Die SageMaker AI-Funktionen sind jetzt verfügbar. Anwendungsbeispiele finden Sie unter Verwenden von SageMaker AI mit Ihrem DB-Cluster von Aurora MySQL.

Verwenden von Amazon Bedrock mit Ihrem DB-Cluster von Aurora MySQL

Um Amazon Bedrock zu verwenden, erstellen Sie eine benutzerdefinierte Funktion (UDF) in Ihrer Aurora-MySQL-Datenbank, die ein Modell aufruft. Weitere Informationen finden Sie unter Was ist Amazon Bedrock? im Amazon Bedrock-Benutzerhandbuch.

Eine UDF verwendet die folgende Syntax:

```
CREATE FUNCTION function_name (argument type)
        [DEFINER = user]
    RETURNS mysql_data_type
    [SQL SECURITY {DEFINER | INVOKER}]
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'model_id'
    [CONTENT_TYPE 'content_type']
    [ACCEPT 'content_type']
    [TIMEOUT_MS timeout_in_milliseconds];
```

- Amazon Bedrock-Funktionen unterstützen `RETURNS JSON` nicht. Sie können `CONVERT` oder `CAST` verwenden, um bei Bedarf von `TEXT` zu `JSON` zu konvertieren.
- Wenn Sie weder `CONTENT_TYPE` noch `ACCEPT` angeben, wird standardmäßig `application/json` verwendet.
- Wenn Sie kein `TIMEOUT_MS` angeben, wird der Standardwert `aurora_ml_inference_timeout` verwendet.

Die folgende UDF ruft beispielsweise das Modell „Amazon Titan Text Express“ auf:

```
CREATE FUNCTION invoke_titan (request_body TEXT)
    RETURNS TEXT
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'amazon.titan-text-express-v1'
    CONTENT_TYPE 'application/json'
    ACCEPT 'application/json';
```

Verwenden Sie den folgenden SQL-Befehl, um einem DB-Benutzer die Verwendung dieser Funktion zu ermöglichen:

```
GRANT EXECUTE ON FUNCTION database_name.invoke_titan TO user@domain-or-ip-address;
```

Dann kann der Benutzer `invoke_titan` wie jede andere Funktion aufrufen, wie im folgenden Beispiel gezeigt. Achten Sie darauf, den Text der Anforderung gemäß den Textmodellen von Amazon Titan zu formatieren.

```
CREATE TABLE prompts (request varchar(1024));
INSERT INTO prompts VALUES (
'{
    "inputText": "Generate synthetic data for daily product sales in various categories
 - include row number, product name, category, date of sale and price. Produce output
 in JSON format. Count records and ensure there are no more than 5.",
    "textGenerationConfig": {
        "maxTokenCount": 1024,
        "stopSequences": [],
        "temperature":0,
        "topP":1
    }
}');

SELECT invoke_titan(request) FROM prompts;

{"inputTextTokenCount":44,"results":[{"tokenCount":296,"outputText":"
```tabular-data-json
{
    "rows": [
        {
            "Row Number": "1",
            "Product Name": "T-Shirt",
            "Category": "Clothing",
            "Date of Sale": "2024-01-01",
            "Price": "$20"
        },
        {
            "Row Number": "2",
            "Product Name": "Jeans",
            "Category": "Clothing",
            "Date of Sale": "2024-01-02",
            "Price": "$30"
        },
        {
            "Row Number": "3",
```
```

```
            "Product Name": "Hat",
            "Category": "Accessories",
            "Date of Sale": "2024-01-03",
            "Price": "$15"
        },
        {
            "Row Number": "4",
            "Product Name": "Watch",
            "Category": "Accessories",
            "Date of Sale": "2024-01-04",
            "Price": "$40"
        },
        {
            "Row Number": "5",
            "Product Name": "Phone Case",
            "Category": "Accessories",
            "Date of Sale": "2024-01-05",
            "Price": "$25"
        }
    ]
}
```","completionReason":"FINISH"}]}

Achten Sie bei anderen Modellen, die Sie verwenden, darauf, den Anforderungstext entsprechend zu formatieren. Weitere Informationen finden Sie unter Inferenzparameter für Fundamentmodelle im Amazon Bedrock-Benutzerhandbuch.

Verwenden von Amazon Comprehend mit Ihrem DB-Cluster von Aurora MySQL

Für Aurora MySQL bietet Aurora Machine Learning die folgenden zwei integrierten Funktionen für die Arbeit mit Amazon Comprehend und Ihren Textdaten. Sie geben den zu analysierenden Text (`input_data`) und die Sprache (`language_code`) an.

aws_comprehend_detect_sentiment

Diese Funktion erkennt, dass der Text eine positive, negative, neutrale oder gemischte emotionale Haltung hat. Die Referenzdokumentation dieser Funktion lautet wie folgt.

```
aws_comprehend_detect_sentiment(
  input_text,
  language_code
```

```
  [,max_batch_size]
)
```

Weitere Informationen finden Sie unter Stimmung im Entwicklerhandbuch für Amazon Comprehend.

aws_comprehend_detect_sentiment_confidence

Diese Funktion misst den Wahrscheinlichkeitsgrad der für einen bestimmten Text erkannten Stimmung. Sie gibt einen Wert (Typ, `double`) zurück, der die Wahrscheinlichkeit der Stimmung angibt, die dem Text von der aws_comprehend_detect_sentiment-Funktion zugewiesen wurde. Die Wahrscheinlichkeit ist eine statistische Metrik zwischen 0 und 1. Je höher der Wahrscheinlichkeitsgrad ist, desto stärker können Sie das Ergebnis gewichten. Eine Zusammenfassung der Dokumentation der Funktion lautet wie folgt.

```
aws_comprehend_detect_sentiment_confidence(
  input_text,
  language_code
  [,max_batch_size]
)
```

In beiden Funktionen (aws_comprehend_detect_sentiment_confidence, aws_comprehend_detect_sentiment) verwendet `max_batch_size` den Standardwert 25, wenn kein Wert angegeben ist. Die Batch-Größe sollte immer größer als 0 sein. Sie können `max_batch_size` zur Optimierung der Leistung der Amazon-Comprehend-Funktionsaufrufe verwenden. Große Stapel bieten schnellere Leistung bei gleichzeitig höherer Speichernutzung auf dem DB-Cluster von Aurora MySQL. Weitere Informationen finden Sie unter Leistungsaspekte zur Verwendung von Aurora Machine Learning mit Aurora MySQL.

Informationen zu den Parametern und Rückgabetypen für die Stimmungserkennungsfunktionen in Amazon Comprehend finden Sie unter DetectSentiment.

Example Beispiel: Eine einfache Abfrage mit Amazon-Comprehend-Funktionen

Hier ist ein Beispiel für eine einfache Abfrage, die diese beiden Funktionen aufruft, um festzustellen, wie zufrieden Ihre Kunden mit Ihrem Support-Team sind. Angenommen, Sie haben eine Datenbanktabelle (`support`), in der Kundenfeedback nach jeder Supportanfrage gespeichert wird. Diese Beispielabfrage wendet beide integrierten Funktionen auf den Text in der `feedback`-Spalte der Tabelle an und gibt die Ergebnisse aus. Die von der Funktion zurückgegebenen Wahrscheinlichkeitswerte sind Doppelwerte zwischen 0,0 und 1,0. Für eine besser lesbare Ausgabe

rundet diese Abfrage die Ergebnisse auf 6 Dezimalstellen auf. Diese Abfrage sortiert außerdem die Ergebnisse in absteigender Reihenfolge, angefangen mit dem Ergebnis mit dem höchsten Wahrscheinlichkeitsgrad, um Vergleiche einfacher zu gestalten.

```
SELECT feedback AS 'Customer feedback',
    aws_comprehend_detect_sentiment(feedback, 'en') AS Sentiment,
    ROUND(aws_comprehend_detect_sentiment_confidence(feedback, 'en'), 6)
    AS Confidence FROM support
    ORDER BY Confidence DESC;
+-------------------------------------------------------+-----------+------------+
| Customer feedback                                     | Sentiment | Confidence |
+-------------------------------------------------------+-----------+------------+
| Thank you for the excellent customer support!         | POSITIVE  |   0.999771 |
| The latest version of this product stinks!            | NEGATIVE  |   0.999184 |
| Your support team is just awesome! I am blown away.   | POSITIVE  |   0.997774 |
| Your product is too complex, but your support is great.  | MIXED     |   0.957958 |
| Your support tech helped me in fifteen minutes.       | POSITIVE  |   0.949491 |
| My problem was never resolved!                        | NEGATIVE  |   0.920644 |
| When will the new version of this product be released?   | NEUTRAL   |   0.902706 |
| I cannot stand that chatbot.                          | NEGATIVE  |   0.895219 |
| Your support tech talked down to me.                  | NEGATIVE  |   0.868598 |
| It took me way too long to get a real person.         | NEGATIVE  |   0.481805 |
+-------------------------------------------------------+-----------+------------+
 10 rows in set (0.1898 sec)
```

Example Beispiel: Ermitteln der durchschnittlichen Stimmung für Text oberhalb eines bestimmten Wahrscheinlichkeitsgrads

Eine typische Amazon Comprehend-Abfrage sucht nach Zeilen, auf denen die Stimmung einen bestimmten Wert hat, mit einem Zuverlässigkeitsniveau, das über einem bestimmten numerischen Wert liegt. Beispielsweise zeigt die folgende Abfrage, wie Sie die durchschnittliche Stimmung in Dokumenten in Ihrer Datenbank feststellen können. Die Abfrage berücksichtigt nur Dokumente mit einer Feststellungszuverlässigkeit von mindestens 80 %.

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')
  WHEN 'POSITIVE' THEN 1.0
  WHEN 'NEGATIVE' THEN -1.0
  ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total
FROM productTable
WHERE productTable.productCode = 1302 AND
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

Verwenden von SageMaker AI mit Ihrem DB-Cluster von Aurora MySQL

Damit Sie die SageMaker AI-Funktionalität von Ihrem DB-Cluster von Aurora MySQL aus verwenden können, müssen Sie gespeicherte Funktionen erstellen, die Ihre Aufrufe an den SageMaker AI-Endpunkt und seine Inferenzfunktionen einbetten. Verwenden Sie dazu `CREATE FUNCTION` von MySQL im Prinzip auf die gleiche Weise wie für andere Verarbeitungsaufgaben auf Ihrem DB-Cluster von Aurora MySQL.

Um in SageMaker AI bereitgestellte Modelle für die Inferenz zu verwenden, erstellen Sie benutzerdefinierte Funktionen mithilfe von MySQL-Datendefinitionssprachenanweisungen (DDL) für gespeicherte Funktionen. Jede gespeicherte Funktion stelle den SageMaker AI-Endpunkt das, der das Modell hostet. Wenn Sie eine solche Funktion definieren, geben Sie die Eingabeparameter für das Modell, den spezifischen aufzurufenden SageMaker AI-Endpunkt und den Rückgabetyp an. Die Funktion gibt die vom SageMaker AI-Endpunkt nach dem Anwenden des Modells auf den Eingabeparametern berechnete Inferenz an.

Alle gespeicherten Aurora Machine Learning-Funktionen geben numerische Typen oder zurüc `VARCHAR`. Sie können alle numerischen Typen außer verwende `BIT`. Andere Typen wie etwa `JSON`, `BLOB`, `TEXT` oder `DATE` sind nicht zulässig.

Das folgende Beispiel zeigt die Syntax `CREATE FUNCTION` für das Arbeiten mit SageMaker AI.

```
CREATE FUNCTION function_name (
     arg1 type1,
     arg2 type2, ...)
     [DEFINER = user]
  RETURNS mysql_type
  [SQL SECURITY { DEFINER | INVOKER } ]
  ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT
  ENDPOINT NAME 'endpoint_name'
  [MAX_BATCH_SIZE max_batch_size];
```

Dies ist eine Erweiterung der regulären `CREATE FUNCTION`-DDL-Anweisung. In der `CREATE FUNCTION`-Anweisung, die die SageMaker AI-Funktion definiert, geben Sie keinen Funktionstext an. Stattdessen geben Sie das Schlüsselwort `ALIAS` dort an, wo normalerweise der Funktionstext steht. Derzeit unterstützt Aurora Machine Learning nur `aws_sagemaker_invoke_endpoint` für diese erweiterte Syntax. Sie müssen den Parameter `endpoint_name` angeben. Ein SageMaker AI-Endpunkt kann verschiedene Eigenschaften für jedes Modell haben.

> **Note**
>
> Weitere Informationen zu `CREATE FUNCTION` finden Sie unter CREATE PROCEDURE- und CREATE FUNCTION-Anweisungen im MySQL-8.0-Referenzhandbuch.

Der Parameter `max_batch_size` ist optional. Standardmäßig beträgt die maximale Stapelgröße 10 000. Sie können diesen Parameter in Ihrer Funktion verwenden, um die maximale Anzahl der in einer Stapelanfrage an SageMaker AI verarbeiteten Eingaben zu beschränken. Der `max_batch_size`-Parameter kann dabei helfen, Fehler durch zu große Eingaben zu vermeiden oder unterstützen, dass SageMaker AI schneller eine Antwort zurückgibt. Dieser Parameter wirkt sich auf die Größe eines internen Puffers für die Verarbeitung von SageMaker AI-Anfragen aus. Die Angabe eines zu großen Werts für `max_batch_size` kann zu erheblicher Speicherüberlastung auf Ihrer DB-Instance führen.

Wir empfehlen Ihnen, für die Einstellung `MANIFEST` den Standardwert `OFF` zu belassen. Obwohl Sie die `MANIFEST ON`-Option verwenden können, können einige SageMaker AI-Funktionen die mit dieser Option exportierte CSV-Datei nicht direkt verwenden. Das Manifestformat ist nicht mit dem erweiterten Manifestformat von SageMaker AI kompatibel.

Sie erstellen eine separate gespeicherte Funktion für jedes Ihrer SageMaker AI-Modelle. Dieses Mapping von Funktionen zu Modellen ist erforderlich, da ein Endpunkt mit einem spezifischen Modell verbunden ist und jedes Modell unterschiedliche Parameter akzeptiert. Die Verwendung von SQL-Typen für die Modelleingaben und den Modellausgabetyp hilft bei der Vermeidung von Konvertierungsfehlern, bei denen Daten zwischen den AWS-Services hin- und herübertragen werden. Sie können steuern, wer das Modell anwenden kann. Sie können auch die Laufzeiteigenschaften steuern, indem Sie einen Parameter angeben, der für die maximale Stapelgröße steht.

Derzeit haben alle Aurora Machine Learning-Funktionen die `NOT DETERMINISTIC`-Eigenschaft. Wenn Sie diese Eigenschaft nicht explizit angeben, richtet Aurora `NOT DETERMINISTIC` automatisch ein. Dies ist erforderlich, weil das SageMaker AI-Modell ohne Benachrichtigung der Datenbank geändert werden kann. Wenn dies geschieht, führen Aufrufe an eine Aurora Machine Learning-Funktion möglicherweise zu unterschiedlichen Ergebnissen für gleiche Eingaben innerhalb einer einzigen Transaktion.

Sie können die Merkmale `CONTAINS SQL`, `NO SQL`, `READS SQL DATA` oder `MODIFIES SQL DATA` in Ihrer `CREATE FUNCTION`-Anweisung nicht verwenden.

Nachfolgend sehen Sie ein Beispiel für den Aufruf eines SageMaker AI-Endpunkts zum Erkennen von Anomalien. Es gibt einen SageMaker AI-Endpunkt `random-cut-forest-model`. Das entsprechende Modell wurde bereits durch den `random-cut-forest`-Algorithmus trainiert. Für jede Eingabe gibt das Modell einen Anomaliewert aus. Dieses Beispiel zeigt die Datenpunkte, deren Wert über 3 Standardabweichungen (etwa das 99,9-te Perzentil) vom Durchschnittsergebnis liegt.

```
CREATE FUNCTION anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value))
 from nyc_taxi);

select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

Zeichensatzanforderungen für SageMaker AI-Funktionen, die Zeichenfolgen zurückgeben

Wir empfehlen die Angabe eines Zeichensatzes von `utf8mb4` als Rückgabetyp für Ihre SageMaker AI-Funktionen, die Zeichenfolgenwerte zurückgeben. Wenn dies nicht sinnvoll ist, verwenden Sie eine ausreichende Zeichenfolgenlänge, damit der Rückgabetyp einen im `utf8mb4`-Zeichensatz ausgedrückten Wert aufnehmen kann. Das folgende Beispiel illustriert die Angabe des `utf8mb4`-Zeichensatzes für Ihre Funktion.

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

Derzeit verwendet jede SageMaker AI-Funktion, die eine Zeichenfolge zurückgibt, den Zeichensatz `utf8mb4` für den Rückgabewert. Der Rückgabewert verwendet diesen Zeichensatz auch dann, wenn Ihre SageMaker AI-Funktion implizit oder explizit für ihren Rückgabetyp einen anderen Zeichensatz angibt. Wenn Ihre SageMaker AI-Funktion einen anderen Zeichensatz für den Rückgabewert angibt, können die zurückgegebenen Daten leicht gekürzt sein, wenn Sie sie in einer Tabellenspalte speichern, die nicht lang genug ist. Beispielsweise erstellt eine Abfrage mit einer `DISTINCT`-Klausel eine temporäre Tabelle. Daher kann das Ergebnis der SageMaker AI-Funktion durch die Art und Weise, in der Zeichenfolgen bei einer Abfrage intern behandelt werden, gekürzt werden.

Export von Daten nach Amazon S3 für das SageMaker AI-Modelltraining (erweitert)

Wir empfehlen Ihnen, mit Aurora Machine Learning und SageMaker AI zu beginnen, indem Sie einige der bereitgestellten Algorithmen verwenden. Die Datenwissenschaftler in Ihrem Team sollten Ihnen

die SageMaker AI-Endpunkte bereitstellen, die Sie mit Ihrem SQL-Code verwenden können. Im Folgenden finden Sie minimale Informationen zur Verwendung Ihres eigenen Amazon-S3-Buckets mit Ihren eigenen SageMaker AI-Modellen und Ihrem DB-Cluster von Aurora MySQL.

Machine Learning besteht aus zwei Hauptschritten: Training und Inferenz. Für das Training von SageMaker AI-Modellen exportieren Sie Daten in einen Amazon S3-Bucket. Der Amazon S3-Bucket wird von einer Jupyter SageMaker AI-Notebook-Instance für das Training Ihres Modells vor seiner Bereitstellung verwendet. Sie können die Anweisung `SELECT INTO OUTFILE S3` verwenden, um Daten aus einem Aurora MySQL-DB-Cluster abzufragen und sie direkt in Textdateien in einem Amazon S3-Bucket zu speichern. Anschließend verwendet die Notebook-Instance die Daten aus dem Amazon S3-Bucket für das Training.

Aurora Machine Learning erweitert die bestehende `SELECT INTO OUTFILE`-Syntax in Aurora MySQL zum Export von Daten im CSV-Format. Die generierte CSV-Datei kann direkt von Modellen verwendet werden, die dieses Format für Trainingszwecke benötigen.

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

Die Erweiterung unterstützt das Standard-CSV-Format.

- Format `TEXT` ist identisch mit dem bestehenden MySQL-Exportformat. Dies ist das Standardformat.
- Format `CSV` ist ein neu eingeführtes Format, das der Spezifikation in RFC-4180 folgt.
- Wenn Sie das optionale Schlüsselwort `HEADER` angeben, enthält die Ausgabedatei eine Kopfzeile. Die Beschriftungen in der Kopfzeile entsprechen den Spaltennamen aus der `SELECT`-Anweisung.
- Sie können die Schlüsselwörter `CSV` und `HEADER` weiterhin als Kennungen verwenden.

Die erweiterte Syntax und Grammatik von `SELECT INTO` sind jetzt wie folgt.

```
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
[FORMAT {CSV|TEXT} [HEADER]]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
```

```
  [TERMINATED BY 'string']
]
```

Leistungsaspekte zur Verwendung von Aurora Machine Learning mit Aurora MySQL

Die Dienste Amazon Bedrock, Amazon Comprehend und SageMaker AI erledigen den größten Teil der Arbeit, wenn sie von einer Aurora-Funktion für Machine Learning aufgerufen werden. Das bedeutet, dass Sie diese Ressourcen nach Bedarf unabhängig voneinander skalieren können. Für Ihren DB-Cluster von Aurora MySQL können Sie Ihre Funktionsaufrufe so effizient wie möglich gestalten. Im Folgenden finden Sie einige Leistungsaspekte, die Sie bei der Arbeit mit Aurora Machine Learning beachten sollten.

Modell und Prompt

Die Leistung bei der Verwendung von Amazon Bedrock hängt stark vom Modell und vom verwendeten Prompt ab. Wählen Sie ein Modell und einen Prompt aus, die für Ihren Anwendungsfall optimal geeignet sind.

Abfrage-Cache

Der Abfrage-Cache von Aurora MySQL funktioniert nicht für Aurora-Funktionen für Machine Learning. Aurora MySQL speichert keine Abfrageergebnisse im Abfrage-Cache für SQL-Anweisungen, die Aurora-Funktionen für Machine Learning aufrufen.

Batch-Optimierung für Aurora-Machine-Learning-Funktionsaufrufe

Der wichtigste Aurora Machine Learning-Performanceaspekt, den Sie von Ihrem Aurora-Cluster aus beeinflussen können, ist die Batch-Moduseinstellung für Aufrufe der gespeicherten Aurora-Machine-Learning-Funktion. Machine-Learning-Funktionen erfordern in der Regel einen erheblichen Overhead, was es unpraktisch macht, einen externen Service für jede Zeile separat aufzurufen. Aurora Machine Learning kann diesen Overhead minimieren, indem die Aufrufe des externen Aurora-Machine-Learning-Services für viele Zeilen in einem einzigen Batch zusammengefasst werden. Aurora Machine Learning empfängt die Antworten für alle Eingabezeilen und liefert die Antworten Zeile für Zeile an die Abfrage, während sie ausgeführt wird. Diese Optimierung verbessert den Durchsatz und die Latenz für Ihre Aurora-Abfragen ohne Änderungen bei den Ergebnissen.

Wenn Sie eine gespeicherte Aurora-Funktion erstellen, die mit einem SageMaker AI-Endpunkt verbunden ist, legen Sie den Parameter für die Stapelgröße fest. Dieser Parameter legt fest, wie

viele Zeilen für jeden zugrunde liegenden Aufruf an SageMaker AI übertragen werden. Bei Abfragen, die sehr viele Zeilen verarbeiten, kann der Overhead für separate SageMaker AI-Aufrufe für jede Zeile erheblich sein. Je größer der von der gespeicherten Prozedur verarbeitete Datensatz ist, umso größer kann die Stapelgröße gewählt werden.

Ob die Stapelmodusoptimierung für eine SageMaker AI-Funktion verwendet werden kann, erkennen Sie, wenn Sie den von der `EXPLAIN PLAN`-Anweisung erstellten Abfrageplan prüfen. In diesem Fall enthält die `extra`-Spalte in dem Ausführungsplan `Batched machine learning`. Das folgende Beispiel zeigt einen Aufruf an eine SageMaker AI-Funktion im Stapelmodus.

```
mysql> CREATE FUNCTION anomaly_score(val real) returns real alias
 aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)

mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows | filtered | Extra                   |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
|  1 | SIMPLE      | nyc_taxi | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |   48 |   100.00 | Batched machine learning |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
1 row in set, 1 warning (0.01 sec)
```

Wenn Sie eine der integrierten Amazon Comprehend-Funktionen aufrufen, können Sie die Stapelgröße steuern, indem Sie den optionalen `max_batch_size`-Parameter angeben. Dieser Parameter schränkt die maximale Anzahl der in jedem Stapel verarbeiteten `input_text`-Werte ein. Wenn mehrere Elemente gleichzeitig gesendet werden, reduziert dies die Anzahl der Wiederholungen zwischen Aurora und Amazon Comprehend. Die Begrenzung der Stapelgröße ist in verschiedenen Situationen sinnvoll, etwa bei einer Abfrage mit einer `LIMIT`-Klausel. Durch die Verwendung eines kleineren Wertes für `max_batch_size` vermeiden Sie, dass Amazon Comprehend häufiger aufgerufen wird als Eingabetexte vorhanden sind.

Die Batchoptimierung für die Evaluierung von Aurora-Machine-Learning-Funktionen gilt in den folgenden Fällen:

- Funktionsaufrufe innerhalb der ausgewählten Liste oder der WHERE-Klausel von SELECT-Anweisungen.
- Funktionsaufrufe in der VALUES-Liste von INSERT- und REPLACE-Anweisungen.
- SageMaker AI-Funktionen in SET-Werten von UPDATE-Anweisungen.

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
  (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
  (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

Überwachung von Aurora Machine Learning

Sie können Batchoperationen von Aurora Machine Learning überwachen, indem Sie mehrere globale Variablen abfragen, wie im folgenden Beispiel gezeigt.

```
show status like 'Aurora_ml%';
```

Sie können die Statusvariablen mit einer FLUSH STATUS-Anweisung zurücksetzen. So stehen alle Zahlen für Gesamtwerte, Durchschnittswerte usw. seit der letzten Zurücksetzung der Variablen.

`Aurora_ml_logical_request_cnt`

Die Anzahl der logischen Anfragen, die die DB-Instance seit dem letzten Status-Reset zum Senden an die Aurora-Machine-Learning-Services ausgewertet hat. Je nachdem, ob Stapelverarbeitung verwendet wurde, kann dieser Wert höher sein als `Aurora_ml_actual_request_cnt`.

`Aurora_ml_logical_response_cnt`

Die aggregierte Zahl der Antworten, die Aurora MySQL von den Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg erhält.

`Aurora_ml_actual_request_cnt`

Die aggregierte Zahl der Anfragen, die Aurora MySQL an die Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg stellt.

`Aurora_ml_actual_response_cnt`

Die aggregierte Zahl der Antworten, die Aurora MySQL von den Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg erhält.

`Aurora_ml_cache_hit_cnt`

Die aggregierte Zahl der internen Cache-Treffer, die Aurora MySQL von den Machine-Learning-Services von Aurora über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg erhält.

`Aurora_ml_retry_request_cnt`

Die Anzahl der wiederholten Anfragen, die die DB-Instance seit dem letzten Status-Reset an die Aurora-Machine-Learning-Services gesendet hat.

`Aurora_ml_single_request_cnt`

Die aggregierte Zahl der Machine-Learning-Funktionen von Aurora, die im Nicht-Batch-Modus über alle von Benutzern der DB-Instance durchgeführten Abfragen hinweg evaluiert werden.

Informationen zur Überwachung der Leistung der von Aurora Machine Learning-Funktionen aufgerufenen SageMaker AI-Vorgänge finden Sie unter Überwachung von Amazon SageMaker AI.

Verwendung von Amazon Aurora Machine Learning mit Aurora PostgreSQL

Wenn Sie Amazon Aurora Machine Learning mit Ihrem Aurora PostgreSQL-DB-Cluster verwenden, können Sie je nach Bedarf Amazon Comprehend oder Amazon SageMaker AI oder Amazon Bedrock verwenden. Diese Services unterstützen jeweils spezifische Anwendungsfälle für Machine Learning.

Aurora Machine Learning wird nur in bestimmten AWS-Regionen und für bestimmte Versionen von Aurora PostgreSQL unterstützt. Bevor Sie versuchen, Aurora Machine Learning einzurichten, überprüfen Sie die Verfügbarkeit Ihrer Aurora-PostgreSQL-Version und Ihrer Region. Details hierzu finden Sie unter Aurora Machine Learning mit Aurora PostgreSQL.

Themen

- Voraussetzungen für die Verwendung von Aurora Machine Learning mit Aurora PostgreSQL
- Unterstützte Funktionen und Einschränkungen von Aurora Machine Learning mit Aurora PostgreSQL
- Einrichten Ihres DB-Clusters von Aurora PostgreSQL zur Verwendung von Aurora Machine Learning
- Verwenden von Amazon Bedrock mit Ihrem DB-Cluster von Aurora PostgreSQL

- Verwenden von Amazon Comprehend mit Ihrem DB-Cluster von Aurora PostgreSQL
- SageMaker KI mit Ihrem Aurora PostgreSQL-DB-Cluster verwenden
- Exportieren von Daten nach Amazon S3 für das SageMaker KI-Modelltraining (Fortgeschritten)
- Leistungsaspekte zur Verwendung von Aurora Machine Learning mit Aurora PostgreSQL
- Überwachung von Aurora Machine Learning

Voraussetzungen für die Verwendung von Aurora Machine Learning mit Aurora PostgreSQL

AWS Dienste für maschinelles Lernen sind verwaltete Dienste, die in ihren eigenen Produktionsumgebungen eingerichtet und ausgeführt werden. Aurora Machine Learning unterstützt die Integration mit Amazon Comprehend, SageMaker AI und Amazon Bedrock. Bevor Sie versuchen, Ihren DB-Cluster von Aurora PostgreSQL für die Verwendung von Aurora Machine Learning einzurichten, stellen Sie sicher, dass Sie die folgenden Anforderungen und Voraussetzungen verstehen.

- Die Amazon Comprehend-, SageMaker AI- und Amazon Bedrock-Services müssen in derselben Weise ausgeführt werden AWS-Region wie Ihr Aurora PostgreSQL-DB-Cluster. Sie können Amazon Comprehend- oder SageMaker AI- oder Amazon Bedrock-Services nicht von einem Aurora PostgreSQL-DB-Cluster in einer anderen Region aus verwenden.
- Wenn sich Ihr Aurora PostgreSQL-DB-Cluster in einer anderen Virtual Public Cloud (VPC) befindet, die auf dem Amazon VPC-Service basiert, als Ihre Amazon Comprehend- und SageMaker AI-Services, muss die Sicherheitsgruppe der VPC ausgehende Verbindungen zum Aurora-Zielservice für maschinelles Lernen zulassen. Weitere Informationen finden Sie unter Aktivieren der Netzwerkkommunikation von Amazon Aurora zu anderen AWS-Services.
- Für SageMaker KI müssen die Komponenten des maschinellen Lernens, die Sie für Inferenzen verwenden möchten, eingerichtet und einsatzbereit sein. Während des Konfigurationsprozesses für Ihren Aurora PostgreSQL-DB-Cluster müssen Sie den Amazon-Ressourcennamen (ARN) des SageMaker KI-Endpunkts verfügbar haben. Die Datenwissenschaftler in Ihrem Team sind wahrscheinlich am besten in der Lage, mit SageMaker KI zu arbeiten, um die Modelle vorzubereiten und die anderen derartigen Aufgaben zu erledigen. Informationen zu den ersten Schritten mit Amazon SageMaker AI finden Sie unter Erste Schritte mit Amazon SageMaker AI. Weitere Informationen zu Rückschlüssen und Endpunkten finden Sie unter Echtzeit-Inferenz.
- Für Amazon Bedrock muss die Modell-ID der Bedrock-Modelle, die Sie für Inferenzen verwenden möchten, während des Konfigurationsprozesses Ihres DB-Clusters von Aurora PostgreSQL

verfügbar sein. Die Datenwissenschaftler in Ihrem Team sind wahrscheinlich am besten in der Lage, mit Bedrock zu arbeiten, um zu entscheiden, welche Modelle verwendet werden sollen, sie bei Bedarf zu optimieren und andere entsprechende Aufgaben zu erledigen. Informationen zu den ersten Schritten mit Amazon Bedrock finden Sie unter So richten Sie Bedrock ein.

- Amazon-Bedrock-Benutzer müssen den Zugriff auf Modelle anfordern, bevor sie verwendet werden können. Wenn Sie zusätzliche Modelle für die Text-, Chat- und Bildgenerierung hinzufügen möchten, müssen Sie den Zugriff auf Modelle in Amazon Bedrock anfordern. Weitere Informationen finden Sie unter Modellzugriff.

Unterstützte Funktionen und Einschränkungen von Aurora Machine Learning mit Aurora PostgreSQL

Aurora Machine Learning unterstützt jeden SageMaker KI-Endpunkt, der das CSV-Format (Comma Separated Value) mit einem `ContentType` Wert von lesen und schreiben kann. `text/csv` Die integrierten SageMaker KI-Algorithmen, die dieses Format derzeit akzeptieren, sind die folgenden.

- Lineares Lernen
- Random Cut Forest
- XGBoost

  Weitere Informationen zu diesen Algorithmen finden Sie unter Choose an Algorithm im Amazon SageMaker AI Developer Guide.

Bei der Verwendung von Amazon Bedrock mit Aurora Machine Learning gelten die folgenden Einschränkungen:

- Die benutzerdefinierten Funktionen (UDFs) bieten eine native Möglichkeit, mit Amazon Bedrock zu interagieren. Sie haben UDFs keine spezifischen Anforderungen an Anfragen oder Antworten, sodass sie jedes Modell verwenden können.
- Sie können UDFs damit jeden gewünschten Workflow erstellen. Sie können beispielsweise Basis-Primitive wie `pg_cron` kombinieren, um eine Abfrage auszuführen, Daten abzurufen, Schlussfolgerungen zu generieren und in Tabellen zu schreiben, um Abfragen direkt zu bearbeiten.
- UDFs unterstützt keine Batch- oder Parallelanrufe.
- Die Aurora Machine Learning-Erweiterung unterstützt keine Vektorschnittstellen. Als Teil der Erweiterung ist eine Funktion verfügbar, mit der die Einbettungen der Modellantwort im `float8[]`-

Format ausgegeben werden können, um diese Einbettungen in Aurora zu speichern. Weitere Informationen zur Verwendung von `float8[]` finden Sie unter Verwenden von Amazon Bedrock mit Ihrem DB-Cluster von Aurora PostgreSQL.

Einrichten Ihres DB-Clusters von Aurora PostgreSQL zur Verwendung von Aurora Machine Learning

Damit Aurora Machine Learning mit Ihrem Aurora PostgreSQL-DB-Cluster funktioniert, müssen Sie für jeden der Dienste, die Sie verwenden möchten, eine AWS Identity and Access Management (IAM-) Rolle erstellen. Die IAM-Rolle ermöglicht es Ihrem DB-Cluster von Aurora PostgreSQL, den Aurora-Dienst für Machine Learning im Namen des Clusters zu verwenden. Sie müssen auch die Aurora-Erweiterung für Machine Learning installieren. In den folgenden Themen finden Sie Einrichtungsverfahren für jeden dieser Aurora-Dienste für Machine Learning.

Topics

- Einrichten von Aurora PostgreSQL für die Verwendung von Amazon Bedrock
- Einrichten von Aurora PostgreSQL für die Verwendung von Amazon Comprehend
- Aurora PostgreSQL für die Verwendung von Amazon AI einrichten SageMaker
  - Aurora PostgreSQL für die Verwendung von Amazon S3 für SageMaker KI einrichten (Fortgeschritten)
- Installieren der Aurora-Erweiterung für Machine Learning

Einrichten von Aurora PostgreSQL für die Verwendung von Amazon Bedrock

Im folgenden Verfahren erstellen Sie zunächst die IAM-Rolle und -Richtlinie, die Aurora PostgreSQL die Berechtigung erteilt, Amazon Bedrock im Namen des Clusters zu verwenden. Anschließend fügen Sie die Richtlinie einer IAM-Rolle an, die Ihren DB-Cluster von Aurora PostgreSQL für die Arbeit mit Amazon Bedrock verwendet. Der Einfachheit halber verwendet dieses Verfahren die AWS-Managementkonsole , um alle Aufgaben abzuschließen.

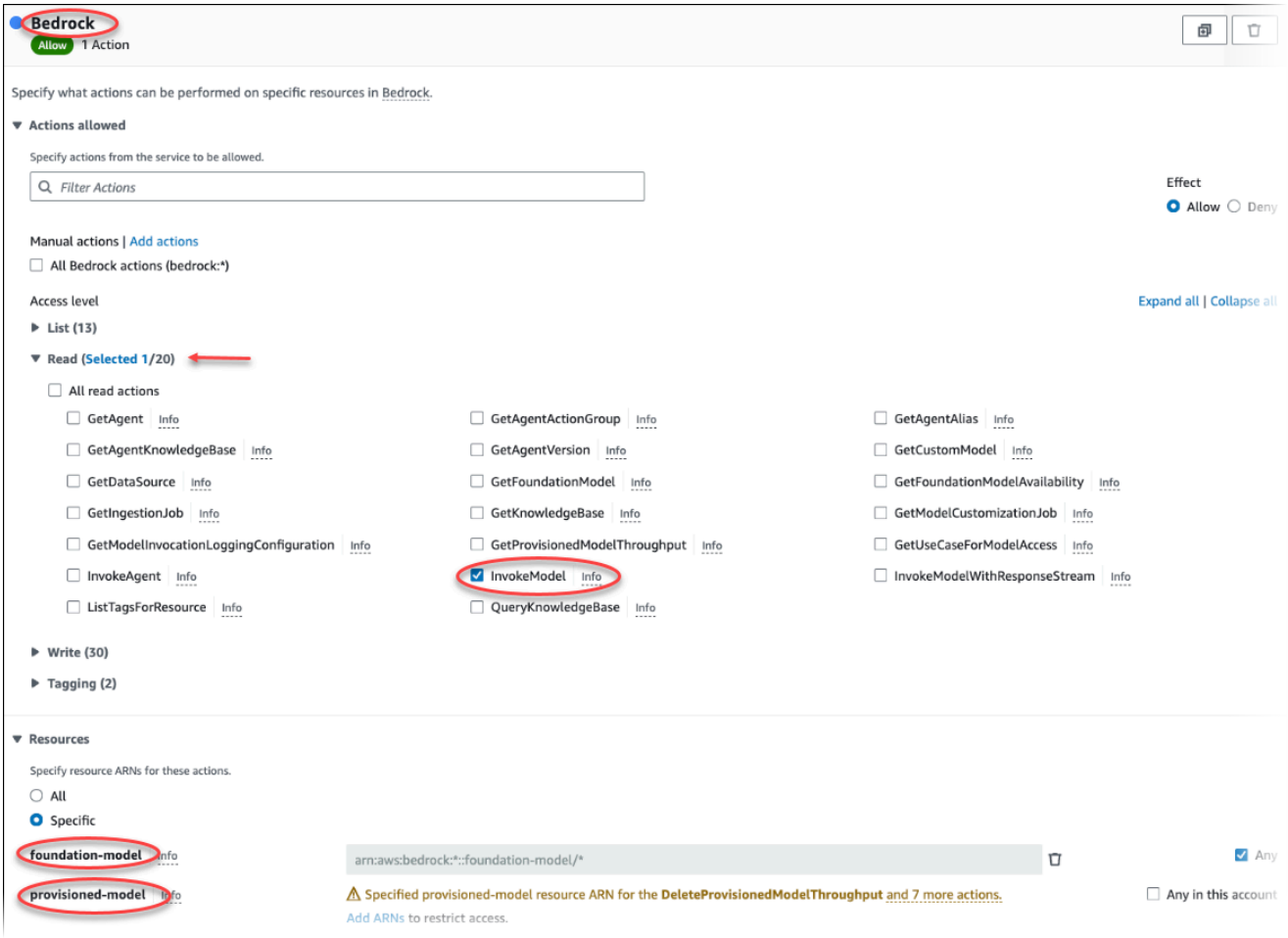
So richten Sie Ihren DB-Cluster von Aurora PostgreSQL zur Verwendung von Amazon Bedrock ein

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.

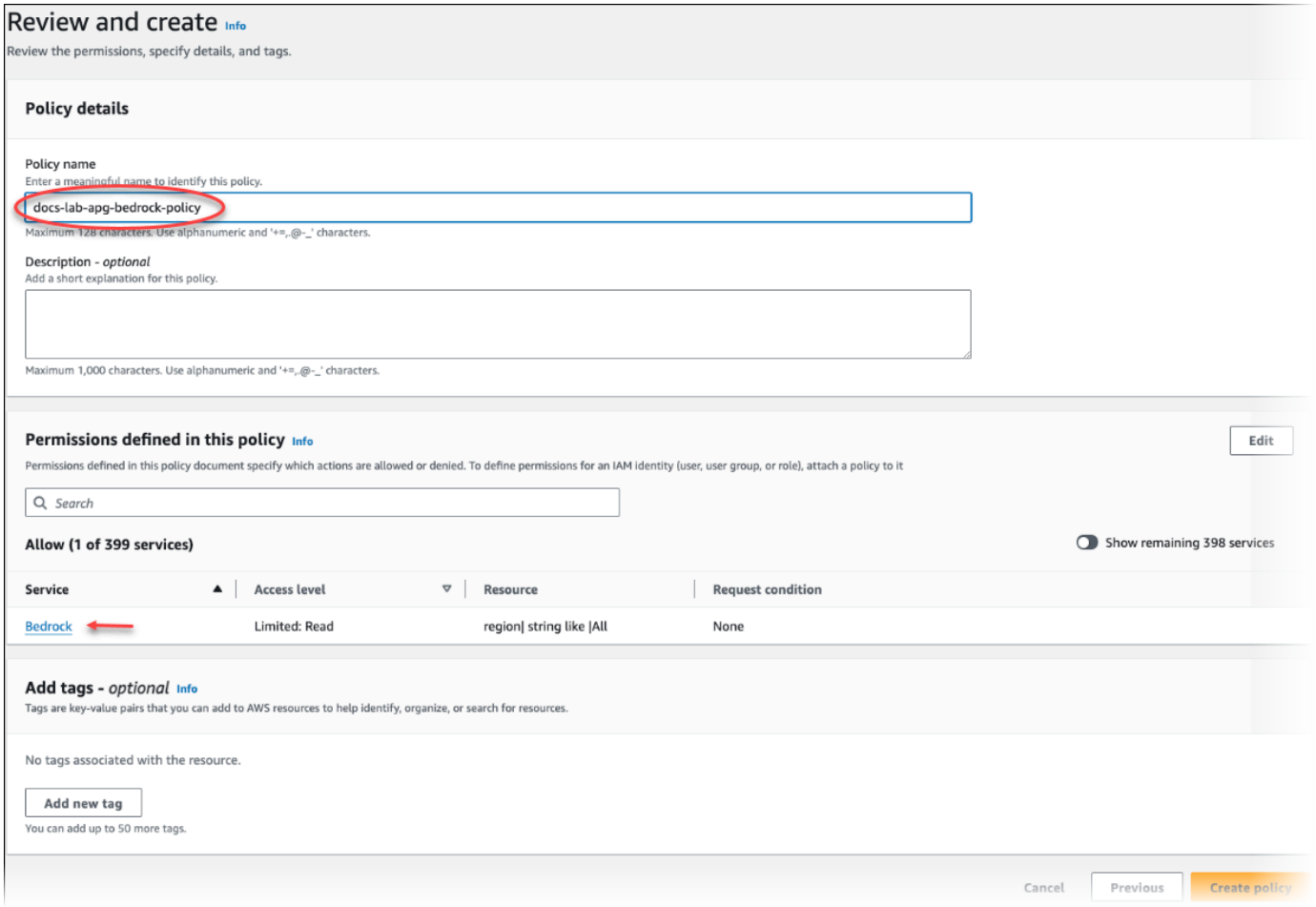
3. Wählen Sie im Konsolenmenü (IAM) die Option Richtlinien AWS Identity and Access Management (unter Zugriffsverwaltung) aus.

   a. Wählen Sie Richtlinie erstellen aus. Wählen Sie auf der Seite „Visueller Editor“ die Option Service aus und geben Sie dann im Feld „Service auswählen“ Bedrock ein. Erweitern Sie die Lesezugriffsebene. Wählen Sie InvokeModelaus den Amazon Bedrock-Leseeinstellungen.

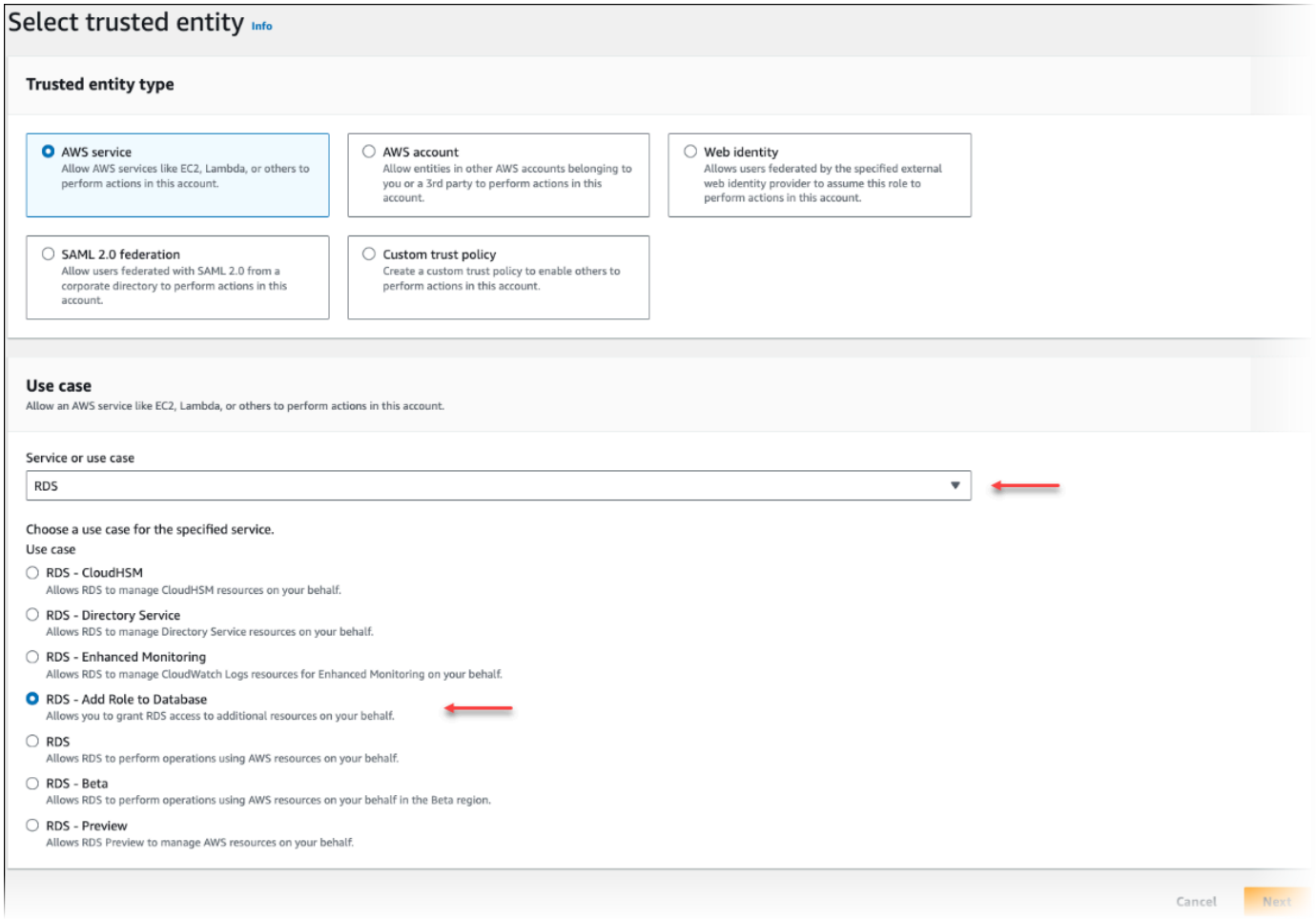
   b. Wählen Sie über die Richtlinie das Foundation/Provisioned Modell aus, dem Sie Lesezugriff gewähren möchten.



4. Wählen Sie Next: Tags (Weiter: Tags) aus und definieren Sie Tags (dies ist optional). Wählen Sie Weiter: Prüfen aus. Geben Sie einen Namen und eine Beschreibung für die Richtlinie ein, wie in der Abbildung gezeigt.

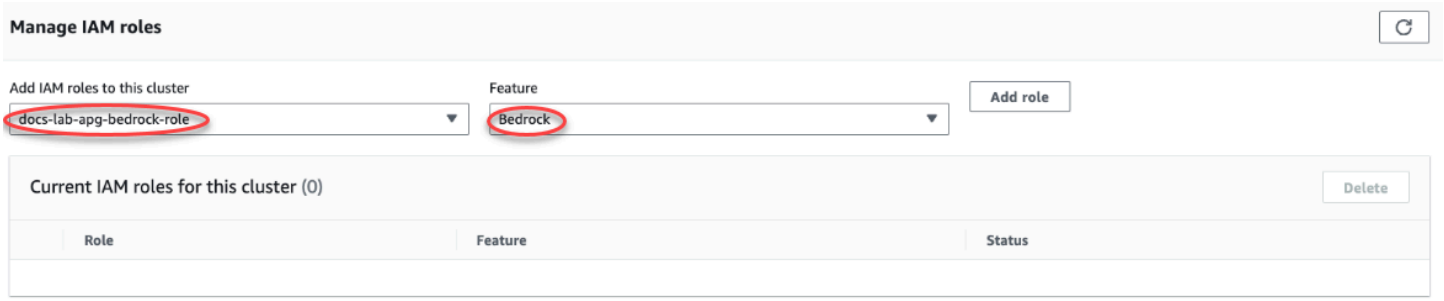


5. Wählen Sie Richtlinie erstellen aus. Die Konsole zeigt eine Warnung an, wenn die Richtlinie gespeichert wurde. Sie finden sie in der Liste der Richtlinien.
6. Wählen Sie in der IAM-Konsole die Option Roles (Rollen) (unter „Access management“ (Zugriffsverwaltung)) aus.
7. Wählen Sie Rolle erstellen aus.
8. Wählen Sie auf der Seite „Vertrauenswürdige Entitäten auswählen“ die Kachel AWS -Service und dann RDS aus, um die Auswahl zu öffnen.
9. Wählen Sie RDS – Add Role to Database (RDS – Rolle zu Datenbank hinzufügen) aus.



10. Wählen Sie Weiter aus. Suchen Sie auf der Seite „Add permissions“ (Berechtigungen hinzufügen) nach der im vorherigen Schritt erstellten Richtlinie und wählen Sie sie aus den aufgelisteten Richtlinien aus. Wählen Sie Weiter aus.
11. Next: Review (Weiter: Überprüfung). Geben Sie einen Namen und eine Beschreibung für die IAM-Rolle ein.
12. Öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
13. Navigieren Sie zu dem AWS-Region Ort, an dem sich Ihr Aurora PostgreSQL-DB-Cluster befindet.
14. Wählen Sie im Navigationsbereich Datenbanken und dann den Aurora PostgreSQL-DB-Cluster aus, den Sie mit Bedrock verwenden möchten.
15. Wählen Sie die Registerkarte Connectivity & security (Konnektivität und Sicherheit) aus und scrollen Sie zum Abschnitt Manage IAM roles (IAM-Rollen verwalten) auf der Seite. Wählen Sie in der Auswahl Add IAM roles to this cluster (Diesem Cluster IAM-Rollen hinzufügen) die Rolle aus, die Sie in den vorherigen Schritten erstellt haben. Wählen Sie in der Auswahl Funktion „Bedrock“ und dann Rolle hinzufügen aus.

Die Rolle ist (mit ihrer Richtlinie) dem DB-Cluster von Aurora PostgreSQL zugeordnet. Wenn der Vorgang abgeschlossen ist, wird die Rolle in der Liste „Current IAM roles for this cluster“ (Aktuelle IAM-Rollen für diesen Cluster) angezeigt, wie in der folgenden Abbildung dargestellt.



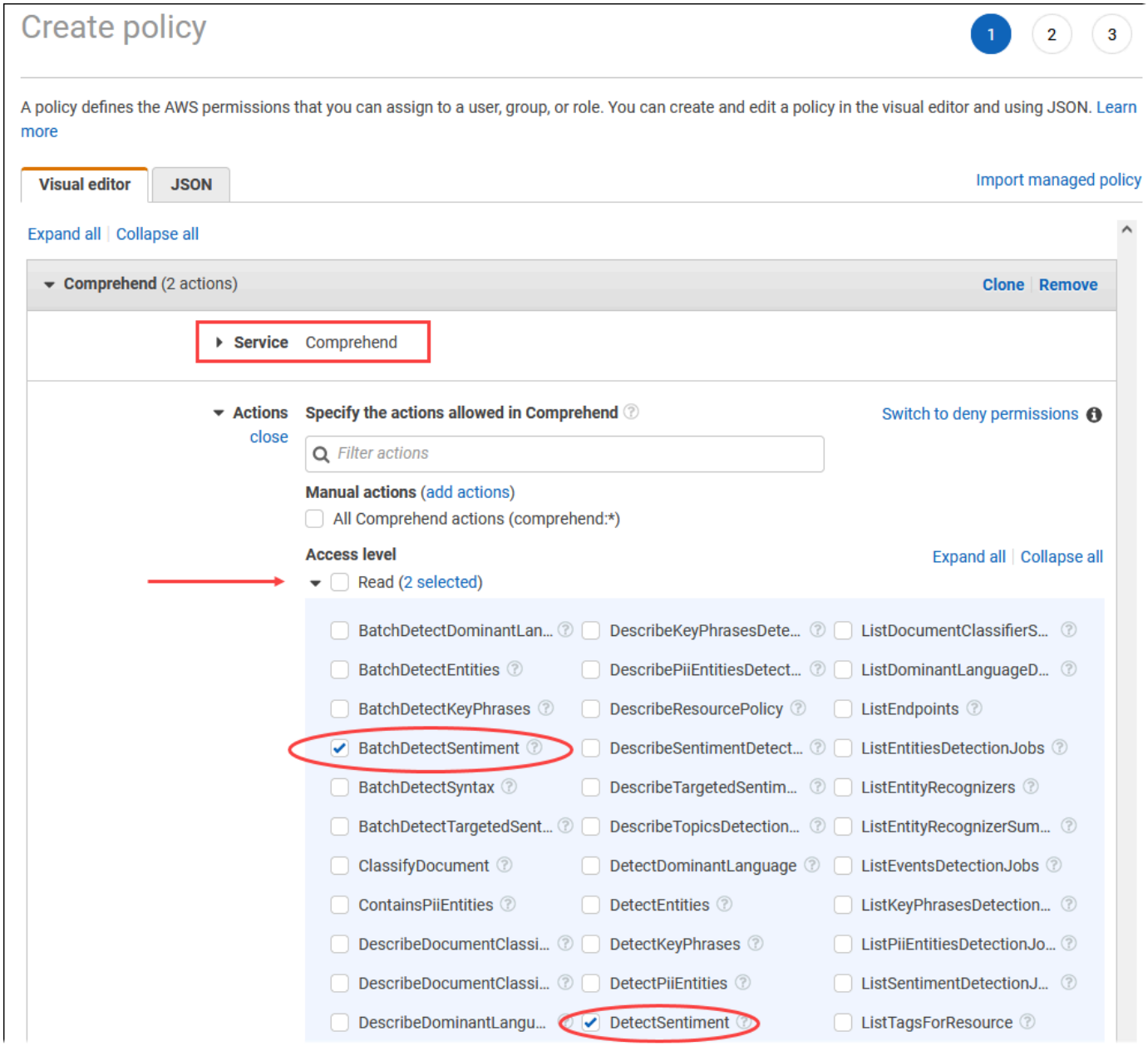
Die IAM-Einrichtung für Amazon Bedrock ist abgeschlossen. Fahren Sie mit der Einrichtung von Aurora PostgreSQL für Aurora Machine Learning fort, indem Sie die Erweiterung wie unter Installieren der Aurora-Erweiterung für Machine Learning beschrieben installieren.

Einrichten von Aurora PostgreSQL für die Verwendung von Amazon Comprehend

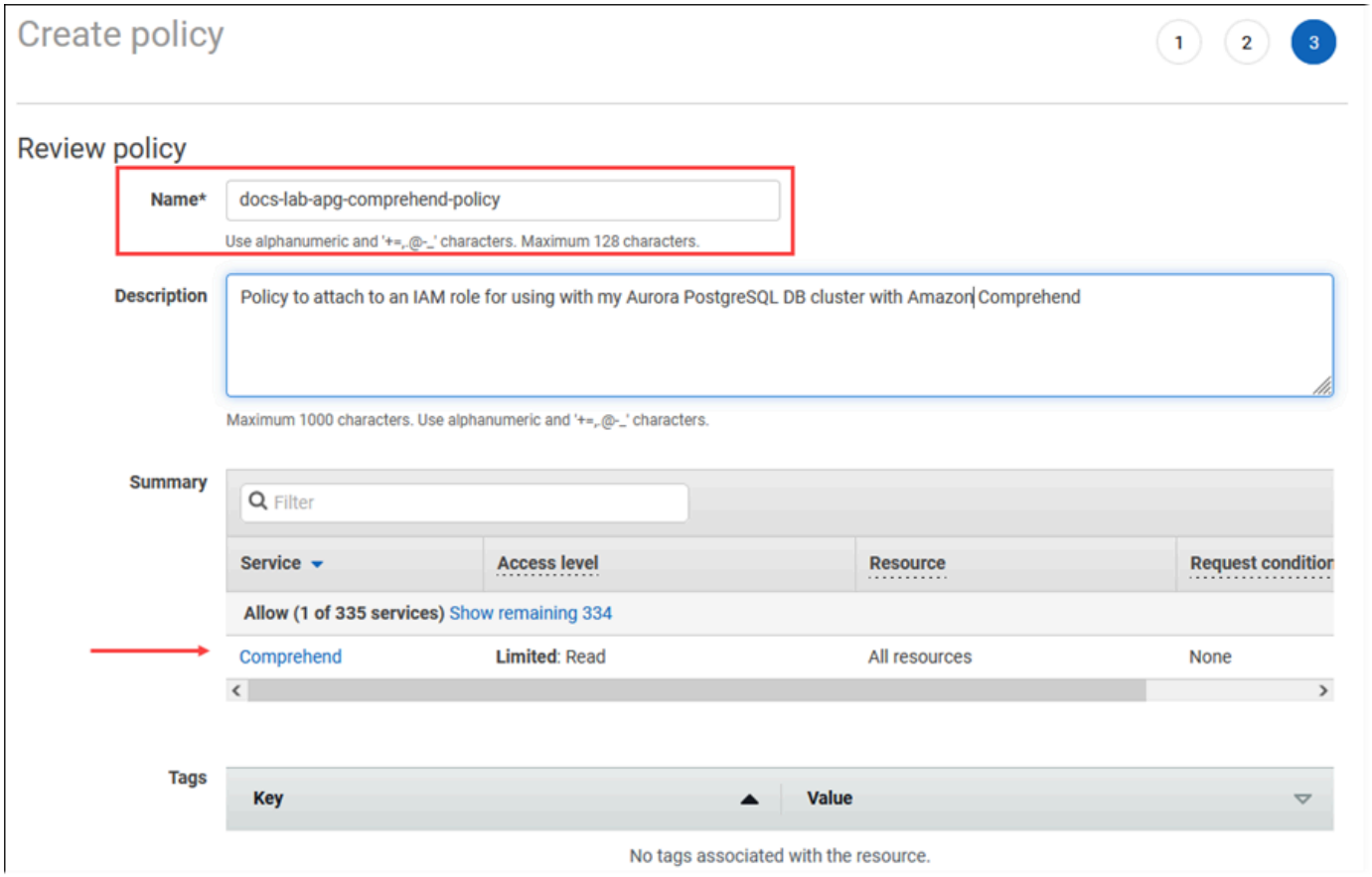
Im folgenden Verfahren erstellen Sie zunächst die IAM-Rolle und -Richtlinie, die Aurora PostgreSQL die Berechtigung erteilt, Amazon Comprehend im Namen des Clusters zu verwenden. Anschließend fügen Sie die Richtlinie einer IAM-Rolle an, die Ihr DB-Cluster von Aurora PostgreSQL für die Arbeit mit Amazon Comprehend verwendet. Der Einfachheit halber nutzt dieses Verfahren die AWS-Managementkonsole , um alle Aufgaben abzuschließen.

So richten Sie Ihren DB-Cluster von Aurora PostgreSQL zur Verwendung von Amazon Comprehend ein

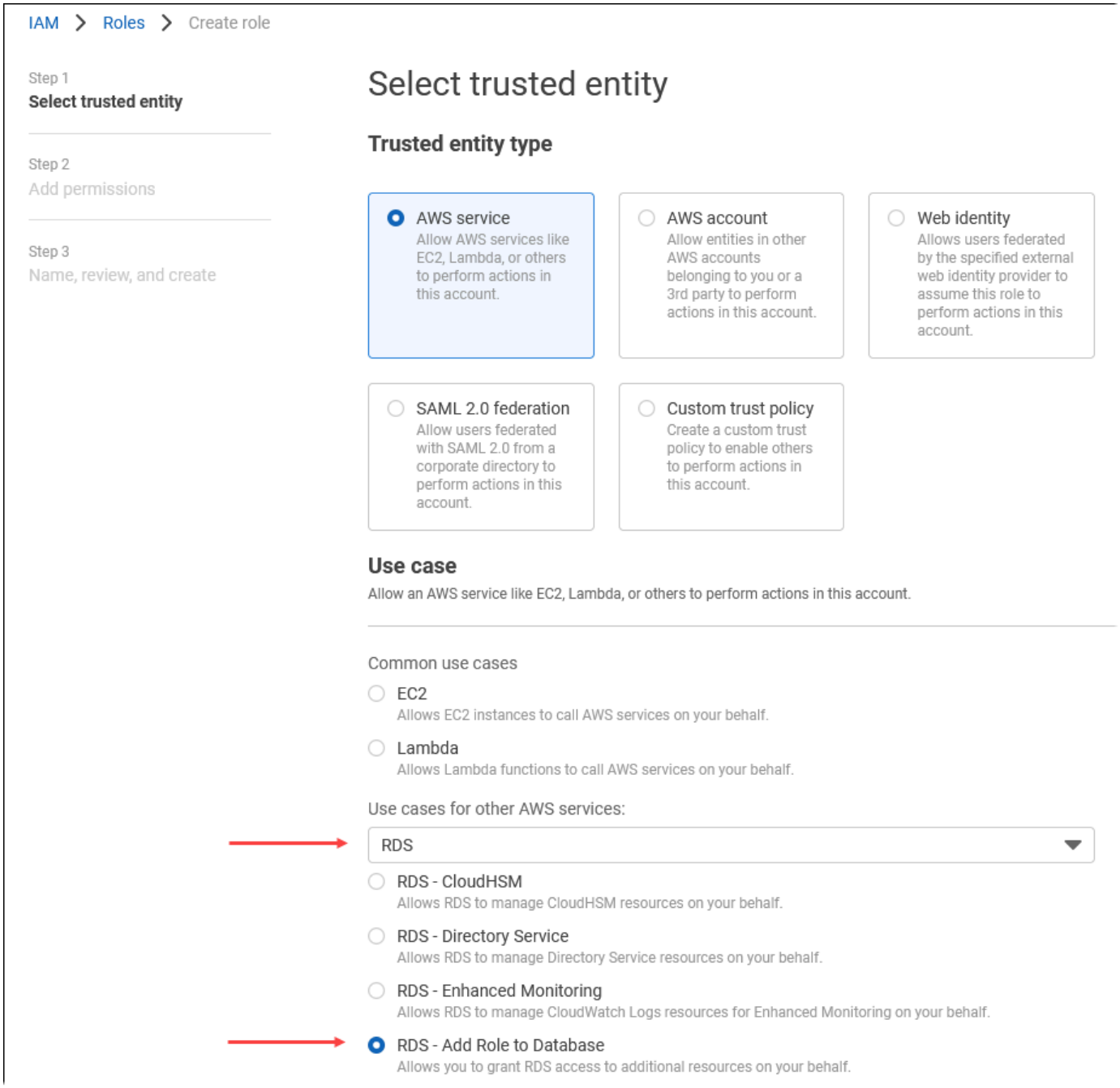
1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
3. Wählen Sie im Konsolenmenü (IAM) die Option Richtlinien AWS Identity and Access Management (unter Zugriffsverwaltung) aus.



4. Wählen Sie Richtlinie erstellen aus. Wählen Sie auf der Seite „Visual editor“ (Visueller Editor) die Option Service (Service) aus und geben Sie dann im Feld „Select a service“ (Service auswählen) Comprehend ein. Erweitern Sie die Lesezugriffsebene. Wählen Sie BatchDetectSentimentund DetectSentimentaus den Amazon Comprehend Leseeinstellungen.
5. Wählen Sie Next: Tags (Weiter: Tags) aus und definieren Sie Tags (dies ist optional). Wählen Sie Weiter: Prüfen aus. Geben Sie einen Namen und eine Beschreibung für die Richtlinie ein, wie in der Abbildung gezeigt.



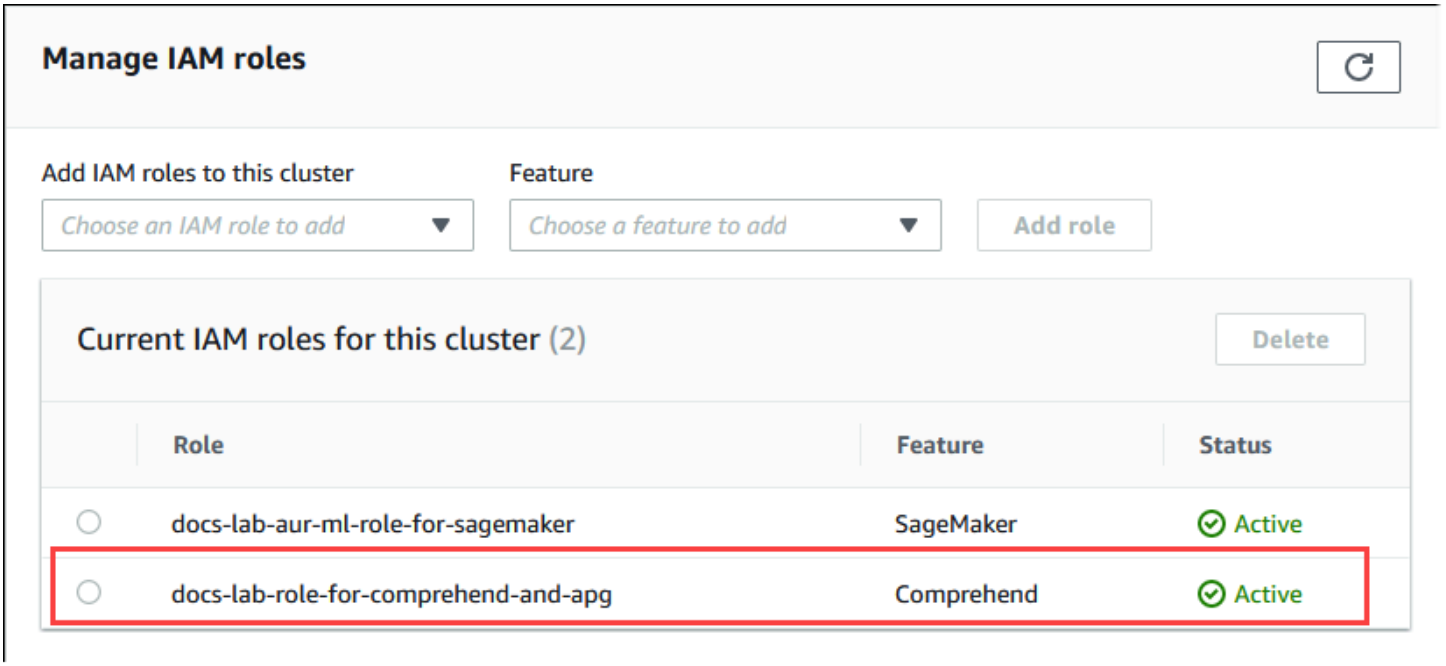
6. Wählen Sie Richtlinie erstellen aus. Die Konsole zeigt eine Warnung an, wenn die Richtlinie gespeichert wurde. Sie finden sie in der Liste der Richtlinien.
7. Wählen Sie in der IAM-Konsole die Option Roles (Rollen) (unter „Access management“ (Zugriffsverwaltung)) aus.
8. Wählen Sie Rolle erstellen aus.
9. Wählen Sie auf der Seite „Vertrauenswürdige Entitäten auswählen“ die Kachel AWS -Service und dann RDS aus, um die Auswahl zu öffnen.
10. Wählen Sie RDS – Add Role to Database (RDS – Rolle zu Datenbank hinzufügen) aus.



11. Wählen Sie Weiter aus. Suchen Sie auf der Seite „Add permissions“ (Berechtigungen hinzufügen) nach der im vorherigen Schritt erstellten Richtlinie und wählen Sie sie aus den aufgelisteten Richtlinien aus. Wählen Sie Next (Weiter) aus.
12. Next: Review (Weiter: Überprüfung). Geben Sie einen Namen und eine Beschreibung für die IAM-Rolle ein.
13. Öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.

14. Navigieren Sie zu dem AWS-Region Ort, an dem sich Ihr Aurora PostgreSQL-DB-Cluster befindet.
15. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den DB-Cluster von Aurora PostgreSQL aus, den Sie mit Amazon Comprehend verwenden möchten.
16. Wählen Sie die Registerkarte Connectivity & security (Konnektivität und Sicherheit) aus und scrollen Sie zum Abschnitt Manage IAM roles (IAM-Rollen verwalten) auf der Seite. Wählen Sie in der Auswahl Add IAM roles to this cluster (Diesem Cluster IAM-Rollen hinzufügen) die Rolle aus, die Sie in den vorherigen Schritten erstellt haben. Wählen Sie in der Auswahl Funktion „Comprehend“ und dann Rolle hinzufügen aus.

Die Rolle ist (mit ihrer Richtlinie) dem DB-Cluster von Aurora PostgreSQL zugeordnet. Wenn der Vorgang abgeschlossen ist, wird die Rolle in der Liste „Current IAM roles for this cluster“ (Aktuelle IAM-Rollen für diesen Cluster) angezeigt, wie in der folgenden Abbildung dargestellt.



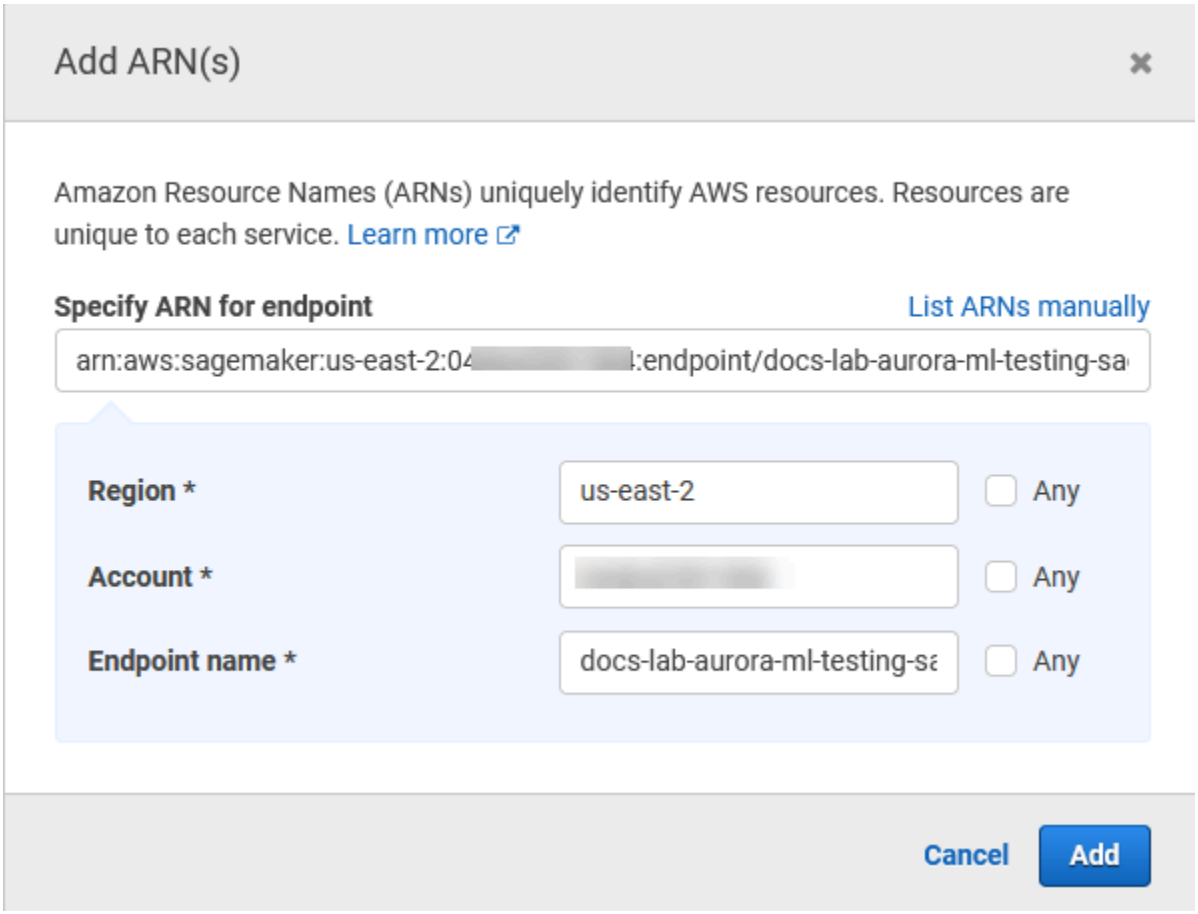
Die IAM-Einrichtung für Amazon Comprehend ist abgeschlossen. Fahren Sie mit der Einrichtung von Aurora PostgreSQL für Aurora Machine Learning fort, indem Sie die Erweiterung wie unter Installieren der Aurora-Erweiterung für Machine Learning beschrieben installieren.

Aurora PostgreSQL für die Verwendung von Amazon AI einrichten SageMaker

Bevor Sie die IAM-Richtlinie und -Rolle für Ihren Aurora PostgreSQL-DB-Cluster erstellen können, müssen Sie Ihr SageMaker KI-Modell einrichten und Ihren Endpunkt verfügbar haben.

So richten Sie Ihren Aurora PostgreSQL-DB-Cluster für die Verwendung von KI ein SageMaker

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. https://console.aws.amazon.com/iam/
2. Wählen Sie im Konsolenmenü (IAM) die Option Richtlinien AWS Identity and Access Management (unter Zugriffsverwaltung) und wählen Sie dann Richtlinie erstellen aus. Wählen Sie im Visual Editor SageMakerden Service aus. Öffnen Sie für Aktionen die Leseauswahl (unter Zugriffsebene) und wählen Sie InvokeEndpoint. Daraufhin wird ein Warnsymbol angezeigt.
3. Öffnen Sie die Ressourcenauswahl und wählen Sie unter Endpunktressourcen-ARN angeben für die InvokeEndpoint Aktion den Link ARN hinzufügen, um den Zugriff einzuschränken.
4. Geben Sie Ihre SageMaker KI-Ressourcen und den Namen Ihres Endpunkts ein. AWS-Region Ihr AWS Konto ist vorausgefüllt.



5. Wählen Sie zum Speichern Add (Hinzufügen) aus. Wählen Sie Next: Tags (Weiter: Tags) und Next: Review (Weiter: Überprüfen) aus, um zur letzten Seite des Richtlinienerstellungsprozesses zu gelangen.

6. Geben Sie einen Namen und eine Beschreibung für die Richtlinie ein und wählen Sie dann Create policy (Richtlinie erstellen) aus. Die Richtlinie wird erstellt und der Liste „Policies“ (Richtlinien) hinzugefügt. Dabei wird in der Konsole eine Warnung angezeigt.
7. Wählen Sie in der IAM-Konsole Roles (Rollen) aus.
8. Wählen Sie Rolle erstellen aus.
9. Wählen Sie auf der Seite „Vertrauenswürdige Entitäten auswählen“ die Kachel AWS -Service und dann RDS aus, um die Auswahl zu öffnen.
10. Wählen Sie RDS – Add Role to Database (RDS – Rolle zu Datenbank hinzufügen) aus.
11. Wählen Sie Weiter aus. Suchen Sie auf der Seite „Add permissions“ (Berechtigungen hinzufügen) nach der im vorherigen Schritt erstellten Richtlinie und wählen Sie sie aus den aufgelisteten Richtlinien aus. Wählen Sie Next (Weiter) aus.
12. Next: Review (Weiter: Überprüfung). Geben Sie einen Namen und eine Beschreibung für die IAM-Rolle ein.
13. Öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
14. Navigieren Sie zu dem AWS-Region Ort, an dem sich Ihr Aurora PostgreSQL-DB-Cluster befindet.
15. Wählen Sie im Navigationsbereich Datenbanken und dann den Aurora PostgreSQL-DB-Cluster aus, den Sie mit SageMaker KI verwenden möchten.
16. Wählen Sie die Registerkarte Connectivity & security (Konnektivität und Sicherheit) aus und scrollen Sie zum Abschnitt Manage IAM roles (IAM-Rollen verwalten) auf der Seite. Wählen Sie in der Auswahl Add IAM roles to this cluster (Diesem Cluster IAM-Rollen hinzufügen) die Rolle aus, die Sie in den vorherigen Schritten erstellt haben. Wählen Sie in der Funktionsauswahl SageMaker AI und dann Rolle hinzufügen aus.

Die Rolle ist (mit ihrer Richtlinie) dem DB-Cluster von Aurora PostgreSQL zugeordnet. Wenn der Vorgang abgeschlossen ist, wird die Rolle in der Liste „Current IAM roles for this cluster“ (Aktuelle IAM-Rollen für diesen Cluster) angezeigt.

Das IAM-Setup für SageMaker KI ist abgeschlossen. Fahren Sie mit der Einrichtung von Aurora PostgreSQL für Aurora Machine Learning fort, indem Sie die Erweiterung wie unter Installieren der Aurora-Erweiterung für Machine Learning beschrieben installieren.

Aurora PostgreSQL für die Verwendung von Amazon S3 für SageMaker KI einrichten (Fortgeschritten)

Um SageMaker KI mit Ihren eigenen Modellen zu verwenden, anstatt die von SageMaker AI bereitgestellten vorgefertigten Komponenten zu verwenden, müssen Sie einen Amazon Simple Storage Service (Amazon S3) -Bucket einrichten, den der Aurora PostgreSQL-DB-Cluster verwenden kann. Dies ist ein Thema für Fortgeschrittene und wird in diesem Benutzerhandbuch für Amazon Aurora nicht vollständig dokumentiert. Der allgemeine Prozess ist derselbe wie bei der Integration der SageMaker KI-Unterstützung, und zwar wie folgt.

1. Erstellen Sie die IAM-Richtlinie und -Rolle für Amazon S3.
2. Fügen Sie die IAM-Rolle und den Amazon-S3-Import oder -Export als Funktion auf der Registerkarte „Connectivity & security“ (Konnektivität und Sicherheit) Ihres DB-Clusters von Aurora PostgreSQL hinzu.
3. Fügen Sie den ARN der Rolle der benutzerdefinierten DB-Cluster-Parametergruppe für jeden Aurora-DB-Cluster hinzu.

Grundlegende Informationen zur Nutzung finden Sie unter Exportieren von Daten nach Amazon S3 für das SageMaker KI-Modelltraining (Fortgeschritten).

Installieren der Aurora-Erweiterung für Machine Learning

Die Aurora-Erweiterungen `aws_ml 1.0` für maschinelles Lernen bieten zwei Funktionen, mit denen Sie Amazon Comprehend, SageMaker AI-Services, aufrufen können, und `aws_ml 2.0` bieten zwei zusätzliche Funktionen, mit denen Sie Amazon Bedrock-Dienste aufrufen können. Durch die Installation der Erweiterung auf Ihrem DB-Cluster von Aurora PostgreSQL wird auch eine administrative Rolle für das Feature erstellt.

> **Note**
>
> Die Verwendung dieser Funktionen hängt davon ab, ob die IAM-Einrichtung für den Aurora-Service für maschinelles Lernen (Amazon Comprehend, SageMaker AI, Amazon Bedrock) abgeschlossen ist, wie unter beschrieben. Einrichten Ihres DB-Clusters von Aurora PostgreSQL zur Verwendung von Aurora Machine Learning

- aws_comprehend.detect_sentiment – Sie verwenden diese Funktion, um Stimmungsanalysen auf Text anzuwenden, der in der Datenbank Ihres DB-Clusters von Aurora PostgreSQL gespeichert ist.

- aws_sagemaker.invoke_endpoint — Sie verwenden diese Funktion in Ihrem SQL-Code, um mit dem KI-Endpunkt von Ihrem Cluster aus zu kommunizieren. SageMaker
- aws_bedrock.invoke_model – Sie verwenden diese Funktion in Ihrem SQL-Code, um mit den Bedrock-Modellen aus Ihrem Cluster zu kommunizieren. Die Antwort dieser Funktion erfolgt im TEXT-Format. Wenn ein Modell also im JSON-Textformat antwortet, wird die Ausgabe dieser Funktion im Zeichenfolgenformat an den Endbenutzer weitergeleitet.
- aws_bedrock.invoke_model_get_embeddings – Sie verwenden diese Funktion in Ihrem SQL-Code, um Bedrock-Modelle aufzurufen, die Ausgabeeinbettungen in einer JSON-Antwort zurückgeben. Dies kann genutzt werden, wenn Sie die direkt mit dem JSON-Key verknüpften Einbettungen extrahieren möchten, um die Antwort mit selbstverwalteten Workflows zu optimieren.

So richten Sie die Aurora-Erweiterung für Machine Learning in Ihrem DB-Cluster von Aurora PostgreSQL ein

- Verwenden Sie `psql`, um eine Verbindung mit der Writer-Instance Ihres DB-Clusters von Aurora PostgreSQL herzustellen. Stellen Sie eine Verbindung mit der entsprechenden Datenbank her, in der die `aws_ml`-Erweiterung installiert werden soll.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=labdb
```

```
labdb=> CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
labdb=>
```

Durch das Installieren der `aws_ml`-Erweiterungen werden auch die administrative `aws_ml`-Rolle und drei neue Schemas wie folgt erstellt.

- `aws_comprehend` – Schema für den Amazon-Comprehend-Service und die Quelle der `detect_sentiment`-Funktion (`aws_comprehend.detect_sentiment`).
- `aws_sagemaker`— Schema für den SageMaker AI-Service und Quelle der Funktion (). `invoke_endpoint` `aws_sagemaker.invoke_endpoint`
- `aws_bedrock` – Schema für den Amazon-Comprehend-Service und die Quelle der Funktionen `invoke_model(aws_bedrock.invoke_model)` und `invoke_model_get_embeddings(aws_bedrock.invoke_model_get_embeddings)`.

Der Rolle `rds_superuser` wird die administrative Rolle `aws_ml` zugewiesen und wird damit zum `OWNER` dieser beiden Aurora-Schemas für Machine Learning. Damit andere Datenbankbenutzer auf die Aurora-Funktionen für Machine Learning zugreifen können, muss der `rds_superuser` `EXECUTE`-Berechtigungen für die Aurora-Funktionen für Machine Learning erteilen. Standardmäßig werden `EXECUTE`-Berechtigungen von `PUBLIC` für die Funktionen in den beiden Aurora-Schemata für Machine Learning widerrufen.

In einer Datenbankkonfiguration mit mehreren Mandanten können Sie verhindern, dass Mandanten auf Aurora-Funktionen für Machine Learning zugreifen, indem Sie `REVOKE USAGE` für das spezifische Aurora-Schema für Machine Learning verwenden, das Sie schützen möchten.

Verwenden von Amazon Bedrock mit Ihrem DB-Cluster von Aurora PostgreSQL

Für Aurora PostgreSQL bietet Aurora Machine Learning die folgende Amazon-Bedrock-Funktion für die Arbeit mit Ihren Textdaten. Diese Funktion ist erst verfügbar, nachdem Sie die Erweiterung `aws_ml` 2.0 installiert und alle Einrichtungsvorgänge abgeschlossen haben. Weitere Informationen finden Sie unter Einrichten Ihres DB-Clusters von Aurora PostgreSQL zur Verwendung von Aurora Machine Learning.

aws_bedrock.invoke_model

Diese Funktion verwendet Text im JSON-Format als Eingabe und verarbeitet ihn für eine Vielzahl von Modellen, die auf Amazon Bedrock gehostet werden, und gibt die JSON-Textantwort vom Modell zurück. Diese Antwort kann Texte, Bilder oder Einbettungen enthalten. Eine Zusammenfassung der Dokumentation der Funktion lautet wie folgt.

```
aws_bedrock.invoke_model(
   IN model_id      varchar,
   IN content_type  text,
   IN accept_type   text,
   IN model_input   text,
   OUT model_output varchar)
```

Die Ein- und Ausgaben dieser Funktion sind wie folgt.

- `model_id` – ID des Modells.
- `content_type` – Der Typ der Anforderung an das Bedrock-Modell.

- `accept_type` – Der vom Modell von Bedrock zu erwartende Antworttyp. Normalerweise application/JSON für die meisten Modelle.
- `model_input` – Eingabeaufforderungen; ein bestimmter Satz von Eingaben für das Modell in dem von content_type angegebenen Format. Weitere Informationen zu der Anfrage, die format/structure das Modell akzeptiert, finden Sie unter Inferenzparameter für Basismodelle.
- `model_output` – Die Ausgabe des Bedrock-Modells als Text.

Das folgende Beispiel zeigt, wie Sie mit invoke_model ein Modell von Anthropic Claude 2 für Bedrock aufrufen.

Example Beispiel: Eine einfache Abfrage mit Amazon-Bedrock-Funktionen

```
SELECT aws_bedrock.invoke_model (
    model_id    := 'anthropic.claude-v2',
    content_type:= 'application/json',
    accept_type := 'application/json',
    model_input := '{"prompt": "\n\nHuman: You are a helpful assistant that answers
 questions directly and only using the information provided in the context below.
\nDescribe the answer
    in detail.\n\nContext: %s \n\nQuestion: %s \n
\nAssistant:","max_tokens_to_sample":4096,"temperature":0.5,"top_k":250,"top_p":0.5,"stop_seque
[]}'
);
```

aws_bedrock.invoke_model_get_embeddings

Die Modellausgabe kann in einigen Fällen auf Vektoreinbettungen verweisen. Da die Antwort je nach Modell unterschiedlich ist, kann eine weitere Funktion namens „invoke_model_get_embeddings“ genutzt werden, die genau wie „invoke_model“ funktioniert, die Einbettungen jedoch durch Angabe des entsprechenden JSON-Schlüssels ausgibt.

```
aws_bedrock.invoke_model_get_embeddings(
   IN model_id      varchar,
   IN content_type  text,
   IN json_key      text,
   IN model_input   text,
   OUT model_output float8[])
```

Die Ein- und Ausgaben dieser Funktion sind wie folgt.

- `model_id` – ID des Modells.
- `content_type` – Der Typ der Anforderung an das Bedrock-Modell. Hier ist „accept_type“ auf den Standardwert `application/json` festgelegt.
- `model_input` – Eingabeaufforderungen; ein bestimmter Satz von Eingaben für das Modell in dem von content_type angegebenen Format. Weitere Informationen zu der Anforderung, die format/structure das Modell akzeptiert, finden Sie unter Inferenzparameter für Fundamentmodelle.
- `json_key` – Verweis auf das Feld, aus dem die Einbettung extrahiert werden soll. Dies kann variieren, wenn sich das Einbettungsmodell ändert.
- `model_output` – Die Ausgabe des Bedrock-Modells als Array von Einbettungen mit 16-Bit-Dezimalstellen.

Das folgende Beispiel zeigt, wie eine Einbettung mit dem Modell Titan Embeddings G1 — Text Embedding für den Begriff PostgreSQL-Überwachungsansichten generiert wird. I/O

Example Beispiel: Eine einfache Abfrage mit Amazon-Bedrock-Funktionen

```
SELECT aws_bedrock.invoke_model_get_embeddings(
   model_id      := 'amazon.titan-embed-text-v1',
   content_type  := 'application/json',
   json_key      := 'embedding',
   model_input   := '{ "inputText": "PostgreSQL I/O monitoring views"}') AS embedding;
```

Verwenden von Amazon Comprehend mit Ihrem DB-Cluster von Aurora PostgreSQL

Für Aurora PostgreSQL bietet Aurora Machine Learning die folgende Amazon-Comprehend-Funktion für die Arbeit mit Ihren Textdaten. Diese Funktion ist erst verfügbar, nachdem Sie die `aws_ml`-Erweiterung installiert und alle Einrichtungsvorgänge abgeschlossen haben. Weitere Informationen finden Sie unter Einrichten Ihres DB-Clusters von Aurora PostgreSQL zur Verwendung von Aurora Machine Learning.

aws_comprehend.detect_sentiment

> Diese Funktion nimmt Text als Eingabe und wertet aus, ob der Text eine positive, negative, neutrale oder gemischte emotionale Haltung hat. Sie gibt diese Stimmung zusammen mit einem Wahrscheinlichkeitsgrad für ihre Bewertung aus. Eine Zusammenfassung der Dokumentation der Funktion lautet wie folgt.

```
aws_comprehend.detect_sentiment(
   IN input_text varchar,
   IN language_code varchar,
   IN max_rows_per_batch int,
   OUT sentiment varchar,
   OUT confidence real)
```

Die Ein- und Ausgaben dieser Funktion sind wie folgt.

- `input_text` – Der Text zur Bewertung und Zuordnung der Stimmung (negativ, positiv, neutral, gemischt).
- `language_code` – Die Sprache des identifizierten `input_text` unter Verwendung der aus zwei Buchstaben bestehenden ISO-639-1-Kennung mit regionalem Untertag (nach Bedarf) bzw. dem aus drei Buchstaben bestehenden ISO-639-2-Code. Zum Beispiel ist `en` der Code für Englisch, `zh` der Code für vereinfachtes Chinesisch. Weitere Informationen finden Sie unter Unterstützte Sprachen im Amazon-Comprehend-Entwicklerhandbuch.
- `max_rows_per_batch` – Die maximale Anzahl von Zeilen pro Batch für die Verarbeitung im Batch-Modus. Weitere Informationen finden Sie unter Grundlagen zum Batch-Modus und zu den Aurora-Funktionen für Machine Learning.
- `sentiment` – Die Stimmung des Eingabetexts, die als POSITIV, NEGATIV, NEUTRAL oder GEMISCHT identifiziert wird.
- `confidence` – Der Grad der Wahrscheinlichkeit für die Genauigkeit der angegebenen `sentiment`. Die Werte können zwischen 0,0 und 1,0 liegen.

Nachstehend finden Sie Beispiele zur Verwendung dieser Funktion.

Example Beispiel: Eine einfache Abfrage mit Amazon-Comprehend-Funktionen

Hier ist ein Beispiel für eine einfache Abfrage, die diese Funktion aufruft, um die Kundenzufriedenheit mit Ihrem Support-Team zu bewerten. Angenommen, Sie haben eine Datenbanktabelle (`support`), in der Kundenfeedback nach jeder Supportanfrage gespeichert wird. Diese Beispielabfrage wendet die `aws_comprehend.detect_sentiment`-Funktion auf den Text in der `feedback`-Spalte der Tabelle an und gibt die Stimmung und den Wahrscheinlichkeitsgrad für diese Stimmung aus. Diese Abfrage gibt Ergebnisse außerdem in absteigender Reihenfolge aus.

```
SELECT feedback, s.sentiment,s.confidence
```

```
    FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
    ORDER BY s.confidence DESC;
 feedback                          | sentiment | confidence
 ------------------------------------------------------------+-----------+------------
 Thank you for the excellent customer support!               | POSITIVE  |   0.999771
 The latest version of this product stinks!                  | NEGATIVE  |   0.999184
 Your support team is just awesome! I am blown away.         | POSITIVE  |   0.997774
 Your product is too complex, but your support is great.     | MIXED     |   0.957958
 Your support tech helped me in fifteen minutes.             | POSITIVE  |   0.949491
 My problem was never resolved!                              | NEGATIVE  |   0.920644
 When will the new version of this product be released?      | NEUTRAL   |   0.902706
 I cannot stand that chatbot.                                | NEGATIVE  |   0.895219
 Your support tech talked down to me.                        | NEGATIVE  |   0.868598
 It took me way too long to get a real person.               | NEGATIVE  |   0.481805

 (10 rows)
```

Damit vermieden wird, dass Sie für die Stimmungserkennung mehr als einmal pro Tabellenzeile belastet werden, können Sie die Ergebnisse materialisieren. Tun Sie dies in den relevanten Zeilen. Beispielsweise werden die Notizen des Klinikers aktualisiert, sodass nur Personen auf Französisch (`fr`) die Stimmungserkennungsfunktion verwenden.

```
UPDATE clinician_notes
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
    LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

Weitere Informationen zur Optimierung Ihrer Funktionsaufrufe finden Sie unter Leistungsaspekte zur Verwendung von Aurora Machine Learning mit Aurora PostgreSQL.

SageMaker KI mit Ihrem Aurora PostgreSQL-DB-Cluster verwenden

Nachdem Sie Ihre SageMaker KI-Umgebung eingerichtet und wie unter beschrieben in Aurora PostgreSQL integriert habenAurora PostgreSQL für die Verwendung von Amazon AI einrichten SageMaker , können Sie mithilfe der Funktion Operationen aufrufen. `aws_sagemaker.invoke_endpoint` Die `aws_sagemaker.invoke_endpoint`-Funktion stellt nur eine Verbindung mit einem Modellendpunkt in derselben AWS-Region her. Wenn Ihre

Datenbank-Instance über mehrere Replikate verfügt, stellen Sie AWS-Regionen sicher, dass Sie jedes SageMaker KI-Modell einrichten und für alle bereitstellen. AWS-Region

Aufrufe von `aws_sagemaker.invoke_endpoint` werden mithilfe der IAM-Rolle authentifiziert, die Sie eingerichtet haben, um Ihren Aurora PostgreSQL-DB-Cluster mit dem SageMaker AI-Service und dem Endpunkt zu verknüpfen, den Sie während des Einrichtungsvorgangs angegeben haben. SageMaker Die Endpunkte des KI-Modells sind auf ein einzelnes Konto beschränkt und nicht öffentlich. Die `endpoint_name` URL enthält nicht die Konto-ID. SageMaker AI bestimmt die Konto-ID anhand des Authentifizierungstoken, das von der SageMaker AI-IAM-Rolle der Datenbankinstanz bereitgestellt wird.

aws_sagemaker.invoke_endpoint

Diese Funktion verwendet den SageMaker AI-Endpunkt als Eingabe und die Anzahl der Zeilen, die als Stapel verarbeitet werden sollen. Sie verwendet auch die verschiedenen Parameter, die vom Endpunkt des SageMaker KI-Modells erwartet werden, als Eingabe. Die Referenzdokumentation dieser Funktion lautet wie folgt.

```
aws_sagemaker.invoke_endpoint(
  IN endpoint_name varchar,
  IN max_rows_per_batch int,
  VARIADIC model_input "any",
  OUT model_output varchar
  )
```

Die Ein- und Ausgaben dieser Funktion sind wie folgt.

- `endpoint_name`— Eine Endpunkt-URL, die AWS-Region—unabhängig ist.
- `max_rows_per_batch` – Die maximale Anzahl von Zeilen pro Batch für die Verarbeitung im Batch-Modus. Weitere Informationen finden Sie unter Grundlagen zum Batch-Modus und zu den Aurora-Funktionen für Machine Learning.
- `model_input` – Ein oder mehrere Eingabeparameter für das Modell. Dabei kann es sich um jeden Datentyp handeln, der vom SageMaker KI-Modell benötigt wird. Mit PostgreSQL können Sie bis zu 100 Eingabeparameter für eine Funktion angeben. Array-Datentypen müssen eindimensional sein, können aber so viele Elemente enthalten, wie vom SageMaker KI-Modell erwartet werden. Die Anzahl der Eingaben für ein SageMaker KI-Modell ist nur durch die SageMaker AI-Nachrichtengrößenbeschränkung von 6 MB begrenzt.
- `model_output`— Die Ausgabe des SageMaker KI-Modells als Text.

Erstellen einer benutzerdefinierten Funktion zum Aufrufen eines KI-Modells SageMaker

Erstellen Sie eine separate benutzerdefinierte Funktion, die `aws_sagemaker.invoke_endpoint` für jedes Ihrer SageMaker KI-Modelle aufgerufen wird. Ihre benutzerdefinierte Funktion stellt den SageMaker KI-Endpunkt dar, der das Modell hostet. Die `aws_sagemaker.invoke_endpoint`-Funktion wird innerhalb der benutzerdefinierten Funktion ausgeführt. Benutzerdefinierte Funktionen bieten viele Vorteile:

- Sie können Ihrem SageMaker KI-Modell einen eigenen Namen geben, anstatt nur alle Ihre SageMaker KI-Modelle aufzurufen`aws_sagemaker.invoke_endpoint`.
- Sie können die Modellendpunkt-URL an nur einer Stelle im SQL-Anwendungscode angeben.
- Sie können `EXECUTE`-Berechtigungen für jede Aurora-Funktion für Machine Learning unabhängig steuern.
- Sie können die Modellein- und Ausgabetypen mit SQL-Typen deklarieren. SQL erzwingt die Anzahl und Art der Argumente, die an Ihr SageMaker KI-Modell übergeben werden, und führt bei Bedarf eine Typkonvertierung durch. Die Verwendung von SQL-Typen führt auch `SQL NULL` zu dem entsprechenden Standardwert, der von Ihrem SageMaker KI-Modell erwartet wird.
- Sie können die maximale Stapelgröße reduzieren, wenn Sie die ersten paar Zeilen etwas schneller zurückgeben möchten.

Um eine benutzerdefinierte Funktion anzugeben, verwenden Sie die SQL Data Definition Language (DDL)-Anweisung `CREATE FUNCTION`. Beim Definieren der Funktion geben Sie Folgendes an:

- Die Eingabeparameter für das Modell.
- Der spezifische SageMaker KI-Endpunkt, der aufgerufen werden soll.
- Den Rückgabetyp.

Die benutzerdefinierte Funktion gibt die vom SageMaker KI-Endpunkt berechnete Inferenz zurück, nachdem das Modell mit den Eingabeparametern ausgeführt wurde. Das folgende Beispiel erstellt eine benutzerdefinierte Funktion für ein SageMaker KI-Modell mit zwei Eingabeparametern.

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
```

```
        arg1, arg2                          -- model inputs are separate arguments
        )::INT                              -- cast the output to INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Beachten Sie Folgendes:

- Die `aws_sagemaker.invoke_endpoint`-Funktionseingabe kann ein oder mehrere Parameter eines beliebigen Datentyps sein.
- In diesem Beispiel wird ein INT-Ausgabetyp verwendet. Wenn Sie die Ausgabe von einem `varchar`-Typ in einen anderen Typ umwandeln, muss sie in einen von PostgreSQL eingebauten skalaren Typ wie `INTEGER`, `REAL`, `FLOAT` oder `NUMERIC` umgewandelt werden. Weitere Informationen zu diesen Typen finden Sie unter Datentypen in der PostgreSQL-Dokumentation.
- Geben Sie `PARALLEL SAFE` an, um die parallele Abfrageausführung zu aktivieren. Weitere Informationen finden Sie unter Verbessern von Antwortzeiten durch parallele Abfrageverarbeitung.
- Geben Sie `COST 5000` an, um die Kosten für die Ausführung der Funktion zu schätzen. Verwenden Sie eine positive Zahl, welche die geschätzten Ausführungskosten für die Funktion in -Einheiten angib `cpu_operator_cost`.

Übergabe eines Arrays als Eingabe an ein SageMaker KI-Modell

Die `aws_sagemaker.invoke_endpoint`-Funktion kann bis zu 100 Eingabeparameter haben, was die Grenze für PostgreSQL-Funktionen ist. Wenn das SageMaker KI-Modell mehr als 100 Parameter desselben Typs benötigt, übergeben Sie die Modellparameter als Array.

Das folgende Beispiel definiert eine Funktion, die ein Array als Eingabe an das SageMaker KI-Regressionsmodell übergibt. Die Ausgabe wird in einen `REAL`-Wert umgewandelt.

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
      'sagemaker_model_endpoint_name',
      NULL,
      params
      )::REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Angabe der Batchgröße beim Aufrufen eines KI-Modells SageMaker

Im folgenden Beispiel wird eine benutzerdefinierte Funktion für ein SageMaker KI-Modell erstellt, die den Standardwert für die Batchgröße auf NULL setzt. Mit der Funktion können Sie auch eine andere Stapelgröße angeben, wenn Sie sie aufrufen.

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL,  -- optional batch size limit
    OUT category INT)                     -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT              -- casts output to type INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Beachten Sie Folgendes:

- Verwenden Sie den optionalen `max_rows_per_batch`-Parameter, um die Anzahl der Zeilen für einen Stapelmodus-Funktionsaufruf zu steuern. Wenn Sie den Wert NULL verwenden, wählt der Abfrageoptimierer automatisch die maximale Stapelgröße aus. Weitere Informationen finden Sie unter Grundlagen zum Batch-Modus und zu den Aurora-Funktionen für Machine Learning.
- Standardmäßig wird die Übergabe von NULL als Parameterwert in eine leere Zeichenfolge übersetzt, bevor sie an SageMaker AI übergeben wird. Für dieses Beispiel haben die Eingaben unterschiedliche Typen.
- Wenn Sie über eine Nicht-Texteingabe oder eine Texteingabe verfügen, die standardmäßig auf einen anderen Wert als eine leere Zeichenfolge gesetzt werden muss, verwenden Sie die `COALESCE`-Anweisung. Verwenden Sie `COALESCE`, um NULL in den gewünschten Nullersetzungswert im Aufruf von `aws_sagemaker.invoke_endpoint` zu übersetzen. Für den `amount`-Parameter in diesem Beispiel wird ein NULL-Wert in 0,0 konvertiert.

Aufrufen eines SageMaker KI-Modells mit mehreren Ausgaben

Das folgende Beispiel erstellt eine benutzerdefinierte Funktion für ein SageMaker KI-Modell, das mehrere Ausgaben zurückgibt. Ihre Funktion muss die Ausgabe der `aws_sagemaker.invoke_endpoint`-Funktion in einen entsprechenden Datentyp umwandeln.

Beispielsweise können Sie den integrierten PostgreSQL-Punkttyp für (x,y)-Paare oder einen benutzerdefinierten zusammengesetzten Typ verwenden.

Diese benutzerdefinierte Funktion gibt Werte aus einem Modell zurück, das mehrere Ausgaben zurückgibt, indem ein zusammengesetzter Typ für die Ausgaben verwendet wird.

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
    IN free_cash_flow NUMERIC(18, 6),
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint('endpt_name',
       max_rows_per_batch,free_cash_flow, debt))::company_forecasts;


$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Verwenden Sie für den zusammengesetzten Typ Felder in der gleichen Reihenfolge wie sie in der Modellausgabe angezeigt werden und wandeln Sie die Ausgabe von `aws_sagemaker.invoke_endpoint` in den zusammengesetzten Typ um. Der Aufrufer kann die einzelnen Felder entweder nach Namen oder mit der PostgreSQL-".*"-Notation extrahieren.

Exportieren von Daten nach Amazon S3 für das SageMaker KI-Modelltraining (Fortgeschritten)

Wir empfehlen Ihnen, sich anhand der bereitgestellten Algorithmen und Beispiele mit Aurora Machine Learning und SageMaker KI vertraut zu machen, anstatt zu versuchen, Ihre eigenen Modelle zu trainieren. Weitere Informationen finden Sie unter Erste Schritte mit Amazon SageMaker AI

Um SageMaker KI-Modelle zu trainieren, exportieren Sie Daten in einen Amazon S3 S3-Bucket. Der Amazon S3 S3-Bucket wird von SageMaker KI verwendet, um Ihr Modell vor der Bereitstellung zu trainieren. Sie können Daten aus einem Aurora PostgreSQL-DB-Cluster abfragen und direkt in Textdateien speichern, die in einem Amazon S3-Bucket gespeichert sind. Dann SageMaker verwendet KI die Daten aus dem Amazon S3 S3-Bucket für das Training. Weitere Informationen zum Training von SageMaker KI-Modellen finden Sie unter Trainieren eines Modells mit Amazon SageMaker AI.

> **Note**
>
> Wenn Sie einen Amazon S3 S3-Bucket für SageMaker KI-Modelltraining oder Batch-Scoring erstellen, verwenden Sie `sagemaker` den Namen des Amazon S3 S3-Buckets. Weitere Informationen finden Sie unter Spezifizieren eines Amazon S3 S3-Buckets zum Hochladen von Trainingsdatensätzen und Speichern von Ausgabedaten im Amazon SageMaker AI Developer Guide.

Weitere Informationen zum Exportieren der Daten finden Sie unter Exportieren von Daten aus einem/einer Aurora PostgreSQL-DB-Cluster zu Amazon S3.

Leistungsaspekte zur Verwendung von Aurora Machine Learning mit Aurora PostgreSQL

Die Amazon Comprehend- und SageMaker KI-Services erledigen die meiste Arbeit, wenn sie von einer Aurora-Funktion für maschinelles Lernen aufgerufen werden. Das bedeutet, dass Sie diese Ressourcen nach Bedarf unabhängig voneinander skalieren können. Für Ihren DB-Cluster von Aurora PostgreSQL können Sie Ihre Funktionsaufrufe so effizient wie möglich gestalten. Im Folgenden finden Sie einige Leistungsaspekte, die Sie bei der Arbeit mit Aurora Machine Learning und Aurora PostgreSQL beachten sollten.

Themen

- Grundlagen zum Batch-Modus und zu den Aurora-Funktionen für Machine Learning
- Verbessern von Antwortzeiten durch parallele Abfrageverarbeitung
- Verwenden von materialisierten Ansichten und materialisierten Spalten

Grundlagen zum Batch-Modus und zu den Aurora-Funktionen für Machine Learning

Normalerweise führt PostgreSQL Funktionen zeilenweise aus. Aurora Machine Learning kann diesen Overhead reduzieren, indem die Aufrufe an den externen Aurora-Service für Machine Learning für viele Zeilen in Batches mit einem Ansatz kombiniert werden, der als Ausführung im Batch-Modus bezeichnet wird. Im Batch-Modus empfängt Aurora Machine Learning die Antworten für einen Batch von Eingabezeilen und sendet dann die Antworten eine Zeile nach der anderen an die laufende Abfrage zurück. Diese Optimierung verbessert den Durchsatz Ihrer Aurora-Abfragen, ohne den PostgreSQL-Abfrageoptimierer zu beschränken.

Aurora verwendet automatisch den Stapelmodus, wenn die Funktion aus der SELECT-Liste, einer WHERE-Klausel oder einer HAVING-Klausel referenziert wird. Beachten Sie, dass einfache CASE-Ausdrücke der obersten Ebene für die Ausführung im Stapelmodus geeignet sind. Gesuchte CASE-Ausdrücke der obersten Ebene sind auch für die Ausführung im Stapelmodus berechtigt, vorausgesetzt, die erste WHEN-Klausel ist ein einfaches Prädikat mit einem Stapelmodus-Funktionsaufruf.

Ihre benutzerdefinierte Funktion muss eine LANGUAGE SQL-Funktion sein und sollte PARALLEL SAFE und COST 5000 angeben.

Funktionsmigration von der SELECT-Anweisung zur FROM-Klausel

Normalerweise wird eine aws_ml-Funktion, die für die Ausführung im Stapelmodus berechtigt ist, automatisch von Aurora in die FROM-Klausel migriert.

Die Migration berechtigter Funktionen im Stapelmodus in die FROM-Klausel kann manuell auf Abfrageebene untersucht werden. Dazu verwenden Sie EXPLAIN-Anweisungen (und ANALYZE und VERBOSE) und finden unter jedem im Stapelmodus die "Stapelverarbeitungs"-Informatione Function Scan. Sie können EXPLAIN (mit VERBOSE) auch verwenden, ohne die Abfrage auszuführen. Sie beobachten dann, ob die Aufrufe der Funktion als Function Scan unter einem eingebetteten Loop-Join erscheinen, der nicht in der ursprünglichen Anweisung angegeben wurde.

Im folgenden Beispiel zeigt der eingebettete Loop-Join-Operator im Plan, dass Aurora die anomaly_score-Funktion migriert hat. Sie migrierte diese Funktion von der SELECT-Liste in die FROM-Klausel, wo sie für die Ausführung im Stapelmodus berechtigt ist.

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
                         QUERY PLAN
-------------------------------------------------------------
 Nested Loop
   Output: anomaly_score((r.description)::text)
   ->  Seq Scan on ts.r
         Output: r.id, r.description, r.score
   ->  Function Scan on public.anomaly_score
         Output: anomaly_score.anomaly_score
         Function Call: anomaly_score((r.description)::text)
```

Um die Ausführung im Stapelmodus zu deaktivieren, setzen Sie den apg_enable_function_migration-Parameter auf false. Dies verhindert die Migration von

aws_ml-Funktionen von der SELECT in die FROM-Klausel. Nachfolgend wird gezeigt, wie Sie dies tun.

```
SET apg_enable_function_migration = false;
```

Der apg_enable_function_migration-Parameter ist ein GUC-Parameter (Grand Unified Configuration), der von der Aurora PostgreSQL apg_plan_mgmt-Erweiterung für die Abfrageplanverwaltung erkannt wird. Um die Funktionsmigration in einer Sitzung zu deaktivieren, speichern Sie den resultierenden Plan mithilfe der Abfrageplanverwaltung als approved-Plan. Zur Laufzeit erzwingt die Abfrageplanverwaltung den approved-Plan mit seiner apg_enable_function_migration-Einstellung. Diese Erzwingung erfolgt unabhängig von der Einstellung des apg_enable_function_migration-GUC-Parameters. Weitere Informationen finden Sie unter Verwalten von Abfrageausführungsplänen für Aurora PostgreSQL.

Verwenden des max_rows_per_batch-Parameters

Sowohl die aws_comprehend.detect_sentiment- als auch die aws_sagemaker.invoke_endpoint-Funktion verfügen über einen max_rows_per_batch-Parameter. Dieser Parameter gibt die Anzahl der Zeilen an, die an den Aurora-Service für Machine Learning gesendet werden können. Je größer der von Ihrer Funktion verarbeitete Datensatz ist, umso größer kann die Batch-Größe gewählt werden.

Funktionen im Batch-Modus verbessern die Effizienz, indem sie Batches von Zeilen erstellen, welche die Kosten der Aurora Machine Learning-Funktionsaufrufe über eine große Anzahl von Zeilen verteilen. Wenn jedoch eine SELECT-Anweisung aufgrund einer LIMIT-Klausel früh beendet wird, kann der Stapel über mehr Zeilen konstruiert werden, als die Abfrage verwendet. Dieser Ansatz kann zu zusätzlichen Gebühren für Ihr Konto führen. AWS Um die Vorteile der Ausführung im Stapelmodus zu nutzen, aber das Erstellen zu großer Stapel zu vermeiden, verwenden Sie einen kleineren Wert für den max_rows_per_batch-Parameter in Ihren Funktionsaufrufen.

Wenn Sie eine EXPLAIN (VERBOSE, ANALYZE) einer Abfrage ausführen, welche die Ausführung im Stapelmodus verwendet, sehen Sie einen FunctionScan-Operator, der unterhalb eines eingebetteten Loop-Joins ist. Die Anzahl der von EXPLAIN gemeldeten Schleifen entspricht der Angabe, wie oft eine Zeile vom FunctionScan-Operator abgerufen wurde. Wenn eine Anweisung eine LIMIT-Klausel verwendet, ist die Anzahl der Abrufe konsistent. Um die Größe des Stapels zu optimieren, setzen Sie den max_rows_per_batch-Parameter auf diesen Wert. Wenn jedoch die Funktion im Stapelmodus in einem Prädikat in der WHERE-Klausel oder HAVING-Klausel referenziert wird, können Sie die Anzahl der Abrufe wahrscheinlich nicht im Voraus kennen. Verwenden Sie in

diesem Fall die Schleifen als Richtlinie und experimentieren Sie mit `max_rows_per_batch`, um eine Einstellung zu finden, welche die Leistung optimiert.

Überprüfen der Ausführung im Stapelmodus

Um festzustellen, ob eine Funktion im Batch-Modus ausgeführt wurde, verwenden Sie `EXPLAIN ANALYZE`. Wenn die Ausführung im Stapelmodus verwendet wurde, enthält der Abfrageplan die Informationen in einem Abschnitt "Stapelverarbeitung".

```
EXPLAIN ANALYZE SELECT user-defined-function();
 Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                                 avg/min/max batch call time=146.273/146.273/146.273
```

In diesem Beispiel gab es 1 Stapel, der 3.333 Zeilen enthielt, deren Verarbeitung 146,273 ms dauerte. Der Abschnitt "Stapelverarbeitung" zeigt Folgendes:

- Wie viele Stapel es für diese Scan-Operation der Funktion gab
- Die durchschnittliche, minimale und maximale Stapelgröße
- Die durchschnittliche, minimale und maximale Stapel-Ausführungszeit

In der Regel ist der letzte Stapel kleiner als der Rest, was häufig zu einer minimalen Stapelgröße führt, die viel kleiner als der Durchschnitt ist.

Um die ersten Zeilen schneller zurückzugeben, setzen Sie den `max_rows_per_batch`-Parameter auf einen kleineren Wert.

Um die Anzahl der Aufrufe im Stapelmodus an den ML-Service zu reduzieren, wenn Sie ein `LIMIT` in Ihrer benutzerdefinierten Funktion verwenden, setzen Sie den `max_rows_per_batch`-Parameter auf einen kleineren Wert.

Verbessern von Antwortzeiten durch parallele Abfrageverarbeitung

Damit Sie möglichst schnell Ergebnisse aus einer großen Anzahl von Zeilen erhalten, können Sie die parallele Abfrageverarbeitung mit der Verarbeitung im Batch-Modus kombinieren. Sie können die parallele Abfrageverarbeitung für `SELECT`-, `CREATE TABLE AS SELECT`- und `CREATE MATERIALIZED VIEW`-Anweisungen verwenden.

> **Note**
>
> PostgreSQL unterstützt noch keine parallele Abfrage für DML-Anweisungen (Data Manipulation Language).

Die parallele Abfrageverarbeitung erfolgt sowohl innerhalb der Datenbank als auch innerhalb des ML-Services. Die Anzahl der Kerne in der Instance-Klasse der Datenbank begrenzt den Parallelitätsgrad, der während der Abfrageausführung verwendet werden kann. Der Datenbankserver kann einen Ausführungsplan für parallele Abfragen erstellen, der die Aufgabe unter einer Gruppe von parallelen Workern partitioniert. Dann kann jeder dieser Worker Stapelanforderungen erstellen, die Zehntausende von Zeilen enthalten (oder so viele, wie es von jedem Service erlaubt ist).

Die gebündelten Anfragen aller parallel Worker werden an den SageMaker KI-Endpunkt gesendet. Der Grad der Parallelität, den der Endpunkt unterstützen kann, hängt von der Anzahl und Art der Instances ab, die ihn unterstützen. Für K Grade der Parallelität benötigen Sie eine Datenbank-Instance-Klasse mit mindestens K Kernen. Außerdem müssen Sie den SageMaker KI-Endpunkt für Ihr Modell so konfigurieren, dass er über K Anfangsinstanzen einer ausreichend leistungsstarken Instanzklasse verfügt.

Zur Nutzung der parallelen Abfrageverarbeitung können Sie den `parallel_workers`-Speicherparameter der Tabelle festlegen, welche die Daten enthält, die Sie übergeben möchten. Sie setzen `parallel_workers` auf eine Funktion im Stapelmodus wie `aws_comprehend.detect_sentiment`. Wenn der Optimierer einen parallelen Abfrageplan wählt, können die AWS ML-Dienste sowohl im Batch als auch parallel aufgerufen werden.

Sie können die folgenden Parameter mit der `aws_comprehend.detect_sentiment`-Funktion verwenden, um einen Plan mit 4-Wege-Parallelität zu erhalten. Wenn Sie einen der beiden folgenden Parameter ändern, müssen Sie die Datenbank-Instance neu starten, damit die Änderungen wirksam werden.

```
-- SET max_worker_processes to 8;  -- default value is 8
-- SET max_parallel_workers to 8;  -- not greater than max_worker_processes
SET max_parallel_workers_per_gather to 4;  -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
-- for the Aurora machine learning function is coming from in order to manually
 override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
```

```
--
ALTER TABLE yourTable SET (parallel_workers = 4);

-- Example query to exploit both batch-mode execution and parallel query
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

Weitere Informationen zum Steuern paralleler Abfragen finden Sie unter Parallele Pläne in der PostgreSQL-Dokumentation.

Verwenden von materialisierten Ansichten und materialisierten Spalten

Wenn Sie einen AWS Service wie SageMaker AI oder Amazon Comprehend aus Ihrer Datenbank aufrufen, wird Ihr Konto gemäß der Preispolitik dieses Dienstes belastet. Um die Gebühren für Ihr Konto zu minimieren, können Sie das Ergebnis des Aufrufs des AWS Dienstes in einer materialisierten Spalte materialisieren, sodass der AWS Service nicht mehr als einmal pro Eingabezeile aufgerufen wird. Wenn gewünscht, können Sie eine `materializedAt`-Zeitstempelspalte hinzufügen, um die Zeit aufzuzeichnen, zu der die Spalten materialisiert wurden.

Die Latenz einer gewöhnlichen einzeiligen `INSERT`-Anweisung ist in der Regel viel geringer als die Latenz beim Aufrufen einer Funktion im Stapelmodus. Daher können Sie möglicherweise die Latenzanforderungen Ihrer Anwendung nicht erfüllen, wenn Sie die Funktion im Stapelmodus für jede einzeilige `INSERT` aufrufen, die Ihre Anwendung ausführt. Um das Ergebnis des Aufrufs eines AWS Dienstes in einer materialisierten Spalte zu materialisieren, müssen Hochleistungsanwendungen in der Regel die materialisierten Spalten auffüllen. Dazu geben sie regelmäßig eine `UPDATE`-Anweisung aus, die gleichzeitig für einen großen Stapel von Zeilen ausgeführt wird.

`UPDATE` führt eine Sperre auf Zeilenebene durch, die sich auf eine laufende Anwendung auswirken kann. Sie müssen also möglicherweise `SELECT ... FOR UPDATE SKIP LOCKED` oder `MATERIALIZED VIEW` verwenden.

Analyseabfragen, die mit einer großen Anzahl von Zeilen in Echtzeit arbeiten, können die Materialisierung im Batch-Modus mit der Echtzeitverarbeitung kombinieren. Dazu fügen diese Abfragen eine `UNION ALL` der vormaterialisierten Ergebnisse mit einer Abfrage über die Zeilen zusammen, die noch keine materialisierten Ergebnisse haben. In einigen Fällen wird eine solche `UNION ALL` an mehreren Stellen benötigt oder die Abfrage wird von einer Drittanbieteranwendung generiert. Wenn ja, können Sie eine `VIEW` erstellen, um die `UNION ALL`-Operation zu kapseln, damit dieses Detail nicht für den Rest der SQL-Anwendung verfügbar gemacht wird.

Sie können eine materialisierte Ansicht verwenden, um die Ergebnisse einer beliebigen SELECT-Anweisung rechtzeitig zu einem Snapshot zu materialisieren. Sie können sie auch verwenden, um die materialisierte Ansicht jederzeit in der Zukunft zu aktualisieren. Derzeit unterstützt PostgreSQL keine inkrementelle Aktualisierung, so dass jedes Mal, wenn die materialisierte Ansicht aktualisiert wird, die materialisierte Ansicht vollständig neu berechnet wird.

Sie können materialisierte Ansichten mit der CONCURRENTLY-Option aktualisieren, die den Inhalt der materialisierten Ansicht aktualisiert, ohne eine exklusive Sperre durchzuführen. Dadurch kann eine SQL-Anwendung aus der materialisierten Ansicht lesen, während sie aktualisiert wird.

Überwachung von Aurora Machine Learning

Sie können die aws_ml-Funktionen überwachen, indem Sie den track_functions-Parameter in Ihrer benutzerdefinierten DB-Cluster-Parametergruppe auf all festlegen. Standardmäßig ist dieser Parameter auf pl eingestellt, was bedeutet, dass nur „procedure-language“-Funktionen verfolgt werden. Wenn Sie diese Einstellung in all ändern, werden auch die aws_ml-Funktionen verfolgt. Weitere Informationen finden Sie unter Laufzeitstatistik in der PostgreSQL-Dokumentation.

Informationen zur Überwachung der Leistung der SageMaker KI-Operationen, die von Aurora-Funktionen für maschinelles Lernen aufgerufen werden, finden Sie unter Monitor Amazon SageMaker AI im Amazon SageMaker AI Developer Guide.

Wenn track_functions auf all festgelegt ist, können Sie die pg_stat_user_functions-Ansicht abfragen, um Statistiken über die Funktionen zu erhalten, die Sie definieren und zum Aufrufen von Aurora-Services für Machine Learning verwenden. Die Ansicht gibt die Anzahl der calls, total_time und self_time für jede Funktion an.

Wenn Sie die Statistiken für die aws_sagemaker.invoke_endpoint- und aws_comprehend.detect_sentiment-Funktionen anzeigen möchten, können Sie die Ergebnisse mit der folgenden Abfrage nach Schemanamen filtern.

```
SELECT * FROM pg_stat_user_functions
    WHERE schemaname
    LIKE 'aws_%';
```

Gehen Sie wie folgt vor, um die Statistiken zu löschen.

```
SELECT pg_stat_reset();
```

Sie können die Namen Ihrer SQL-Funktionen, die die `aws_sagemaker.invoke_endpoint`-Funktion aufrufen, abrufen, indem Sie den `pg_proc`-Systemkatalog von PostgreSQL abfragen. Dieser Katalog speichert Informationen zu Funktionen, Prozeduren und mehr. Weitere Informationen finden Sie unter pg_proc in der PostgreSQL-Dokumentation. Das folgende Beispiel veranschaulicht das Abfragen der Tabelle, um die Namen von Funktionen (`proname`) zu erhalten, deren Quelle (`prosrc`) den Text invoke_endpoint enthält.

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

Codebeispiele für Aurora mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie Aurora mit einem AWS Software Development Kit (SDK) verwendet wird.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erste Schritte

Hello Aurora

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung von Aurora beginnen.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Code für die CMake Datei CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
```

```
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code für die Quelldatei „hello_aurora.cpp“.

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
```

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(
  ctx, &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
```

```
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
```

```
    }

    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK for Java 2.x API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import boto3

# Create an RDS client
rds = boto3.client("rds")

# Create a paginator for the describe_db_clusters operation
paginator = rds.get_paginator("describe_db_clusters")

# Use the paginator to get a list of DB clusters
response_iterator = paginator.paginate(
    PaginationConfig={
        "PageSize": 50,  # Adjust PageSize as needed
        "StartingToken": None,
    }
```

```
)

# Iterate through the pages of the response
clusters_found = False
for page in response_iterator:
    if "DBClusters" in page and page["DBClusters"]:
        clusters_found = True
        print("Here are your RDS Aurora clusters:")
        for cluster in page["DBClusters"]:
            print(
                f"Cluster ID: {cluster['DBClusterIdentifier']}, Engine:
 {cluster['Engine']}"
            )

if not clusters_found:
    print("No clusters found!")
```

- Einzelheiten zur API finden Sie unter DBClustersDescribe in AWS SDK for Python (Boto3) API-Referenz.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
require 'aws-sdk-rds'

# Creates an Amazon RDS client for the AWS Region
rds = Aws::RDS::Client.new

puts 'Listing clusters in this AWS account...'

# Calls the describe_db_clusters method to get information about clusters
resp = rds.describe_db_clusters(max_records: 20)
```

```
# Checks if any clusters are found and prints the appropriate message
if resp.db_clusters.empty?
  puts 'No clusters found!'
else
  # Loops through the array of cluster objects and prints the cluster identifier
  resp.db_clusters.each do |cluster|
    puts "Cluster identifier: #{cluster.db_cluster_identifier}"
  end
end
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
```

```
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- Einzelheiten zur API finden Sie unter DBClustersDescribe in AWS SDK for Rust API-Referenz.

Codebeispiele

- Grundlegende Beispiele für die Verwendung von Aurora AWS SDKs
  - Hello Aurora
  - Lernen Sie die Grundlagen von Aurora mit einem AWS SDK kennen
  - Aktionen für Aurora mit AWS SDKs
    - Verwendung CreateDBCluster mit einem AWS SDK
    - Verwendung CreateDBClusterParameterGroup mit einem AWS SDK
    - Verwendung CreateDBClusterSnapshot mit einem AWS SDK
    - Verwendung CreateDBInstance mit einem AWS SDK
    - Verwendung DeleteDBCluster mit einem AWS SDK

    - Verwendung DeleteDBClusterParameterGroup mit einem AWS SDK
    - Verwendung DeleteDBInstance mit einem AWS SDK
    - Verwendung DescribeDBClusterParameterGroups mit einem AWS SDK
    - Verwendung DescribeDBClusterParameters mit einem AWS SDK
    - Verwendung DescribeDBClusterSnapshots mit einem AWS SDK
    - Verwendung DescribeDBClusters mit einem AWS SDK
    - Verwendung DescribeDBEngineVersions mit einem AWS SDK
    - Verwendung DescribeDBInstances mit einem AWS SDK
    - Verwendung DescribeOrderableDBInstanceOptions mit einem AWS SDK oder CLI
    - Verwendung ModifyDBClusterParameterGroup mit einem AWS SDK
- Szenarien für die Verwendung von Aurora AWS SDKs
    - Leihbibliothek-REST-API erstellen
    - Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

Grundlegende Beispiele für die Verwendung von Aurora AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie die Grundlagen von Amazon Aurora mit verwenden können AWS SDKs.

Beispiele

- Hello Aurora
- Lernen Sie die Grundlagen von Aurora mit einem AWS SDK kennen
- Aktionen für Aurora mit AWS SDKs
    - Verwendung CreateDBCluster mit einem AWS SDK
    - Verwendung CreateDBClusterParameterGroup mit einem AWS SDK
    - Verwendung CreateDBClusterSnapshot mit einem AWS SDK
    - Verwendung CreateDBInstance mit einem AWS SDK
    - Verwendung DeleteDBCluster mit einem AWS SDK
    - Verwendung DeleteDBClusterParameterGroup mit einem AWS SDK
    - Verwendung DeleteDBInstance mit einem AWS SDK

- Verwendung DescribeDBClusterParameterGroups mit einem AWS SDK
- Verwendung DescribeDBClusterParameters mit einem AWS SDK
- Verwendung DescribeDBClusterSnapshots mit einem AWS SDK
- Verwendung DescribeDBClusters mit einem AWS SDK
- Verwendung DescribeDBEngineVersions mit einem AWS SDK
- Verwendung DescribeDBInstances mit einem AWS SDK
- Verwendung DescribeOrderableDBInstanceOptions mit einem AWS SDK oder CLI
- Verwendung ModifyDBClusterParameterGroup mit einem AWS SDK

Hello Aurora

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung von Aurora beginnen.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
```

```
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Code für die CMake Datei CMake Lists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.
```

```
    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code für die Quelldatei „hello_aurora.cpp“.

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(
  ctx, &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
```

```
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
```

```
    }

    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK for Java 2.x API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import boto3

# Create an RDS client
rds = boto3.client("rds")

# Create a paginator for the describe_db_clusters operation
paginator = rds.get_paginator("describe_db_clusters")

# Use the paginator to get a list of DB clusters
response_iterator = paginator.paginate(
    PaginationConfig={
        "PageSize": 50,  # Adjust PageSize as needed
        "StartingToken": None,
    }
```

```
)

# Iterate through the pages of the response
clusters_found = False
for page in response_iterator:
    if "DBClusters" in page and page["DBClusters"]:
        clusters_found = True
        print("Here are your RDS Aurora clusters:")
        for cluster in page["DBClusters"]:
            print(
                f"Cluster ID: {cluster['DBClusterIdentifier']}, Engine:
 {cluster['Engine']}"
            )

if not clusters_found:
    print("No clusters found!")
```

- Einzelheiten zur API finden Sie unter DBClustersDescribe in AWS SDK for Python (Boto3) API-Referenz.

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
require 'aws-sdk-rds'

# Creates an Amazon RDS client for the AWS Region
rds = Aws::RDS::Client.new

puts 'Listing clusters in this AWS account...'

# Calls the describe_db_clusters method to get information about clusters
resp = rds.describe_db_clusters(max_records: 20)
```

```
# Checks if any clusters are found and prints the appropriate message
if resp.db_clusters.empty?
  puts 'No clusters found!'
else
  # Loops through the array of cluster objects and prints the cluster identifier
  resp.db_clusters.each do |cluster|
    puts "Cluster identifier: #{cluster.db_cluster_identifier}"
  end
end
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für Ruby API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
```

```
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- Einzelheiten zur API finden Sie unter DBClustersDescribe in AWS SDK for Rust API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Lernen Sie die Grundlagen von Aurora mit einem AWS SDK kennen

Die folgenden Code-Beispiele veranschaulichen Folgendes:

- Erstellen Sie eine benutzerdefinierte Aurora-DB-Cluster-Parametergruppe und legen Sie Parameterwerte fest.
- Erstellen Sie einen DB-Cluster, der die Parametergruppe verwendet.
- Erstellen Sie eine DB-Instance, die eine Datenbank enthält.

- Erstellen Sie einen Snapshot des DB-Clusters und bereinigen Sie dann die Ressourcen.

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using
 the DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group
 using the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
```

```
   4.  Get some parameters in the group using the
DescribeDBClusterParametersAsync method.
   5.  Parse and display some parameters in the group.
   6.  Modify the auto_increment_offset and auto_increment_increment parameters
       using the ModifyDBClusterParameterGroupAsync method.
   7.  Get and display the updated parameters using the
DescribeDBClusterParametersAsync method with a source of "user".
   8.  Get a list of allowed engine versions using the
DescribeDBEngineVersionsAsync method.
   9.  Create an Aurora DB cluster that contains a MySql database and uses the
parameter group.
       using the CreateDBClusterAsync method.
   10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
method.
   11. Display and select from a list of instance classes available for the
selected engine and version
       using the paginated DescribeOrderableDBInstanceOptions method.
   12. Create a database instance in the cluster using the CreateDBInstanceAsync
method.
   13. Wait for the DB instance to be ready using the DescribeDBInstances
method.
   14. Display the connection endpoint string for the new DB cluster.
   15. Create a snapshot of the DB cluster using the
CreateDBClusterSnapshotAsync method.
   16. Wait for DB snapshot to be ready using the
DescribeDBClusterSnapshotsAsync method.
   17. Delete the DB instance using the DeleteDBInstanceAsync method.
   18. Delete the DB cluster using the DeleteDBClusterAsync method.
   19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync
methods.
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
```

```
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
                    .AddTransient<AuroraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger<AuroraScenario>();

        auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

        Console.WriteLine(sepBar);
        Console.WriteLine(
            "Welcome to the Amazon Aurora: get started with DB clusters
example.");
        Console.WriteLine(sepBar);

        DBClusterParameterGroup parameterGroup = null!;
        DBCluster? newCluster = null;
        DBInstance? newInstance = null;

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await
ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName, parameters);
```

```
            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
```

```
   /// Choose the Aurora DB parameter group family from a list of available
options.
   /// </summary>
   /// <returns>The selected parameter group family.</returns>
   public static async Task<string> ChooseParameterGroupFamilyAsync()
   {
       Console.WriteLine(sepBar);
       // 1. Get a list of available engines.
       var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

       Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

       var parameterGroupFamilies =
           engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
       for (var i = 1; i <= parameterGroupFamilies.Count; i++)
       {
           var parameterGroupFamily = parameterGroupFamilies[i - 1];
           // List the available parameter group families.
           Console.WriteLine(
               $"\t{i}. Family: {parameterGroupFamily.Key}");
       }

       var choiceNumber = 0;
       while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
       {
           Console.WriteLine("2. Select an available DB parameter group family
by entering a number from the preceding list:");
           var choice = Console.ReadLine();
           Int32.TryParse(choice, out choiceNumber);
       }
       var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
1];
       Console.WriteLine(sepBar);
       return parameterGroupFamilyChoice.Key;
   }

   /// <summary>
   /// Create and get information on a DB parameter group.
   /// </summary>
   /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
new DB parameter group.</param>
   /// <returns>The new DBParameterGroup.</returns>
```

```
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParamet

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>>
 DescribeParametersInGroupAsync(string parameterGroupName, List<string>?
 parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);
```

```
        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
 parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
 List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await
 auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
 parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
 parameterGroupName)
```

```
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
group.");

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName,
"user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }
```

```
        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
 entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
```

```
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDBInstanceClass(string engine, string engineVersion)
    {
```

```
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await
 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by
 entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
```

```
   public static async Task<DBInstance?> CreateNewInstance(
       string clusterIdentifier,
       string engineName,
       string engineVersion,
       string instanceClass,
       string instanceIdentifier)
   {
       Console.WriteLine(sepBar);
       Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
       bool isInstanceReady = false;
       DBInstance newInstance;
       var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
       isInstanceReady = instances.FirstOrDefault(i =>
           i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

       if (isInstanceReady)
       {
           Console.WriteLine("Instance already created.");
           newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
       }
       else
       {
           newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
               clusterIdentifier,
               instanceIdentifier,
               engineName,
               engineVersion,
               instanceClass
           );

           Console.WriteLine("13. Waiting for DB instance to be ready...");
           while (!isInstanceReady)
           {
               Console.Write(".");
               Thread.Sleep(5000);
               instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
               isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
               newInstance = instances.First();
           }
```

```
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await
 auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
```

```
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
```

```
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
 {newCluster.DBClusterIdentifier}.");
            await
 auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
 newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
 group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
 {parameterGroup.DBClusterParameterGroupName}.");
            await
 auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGr
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
```

```
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Wrapper-Methoden, die vom Szenario aufgerufen werden, um Aurora-Aktionen zu verwalten.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
```

```
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
```

```
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
 _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
```

```
   /// <returns>The name of the group that was modified.</returns>
   public async Task<string> ModifyIntegerParametersInGroupAsync(string
groupName, List<Parameter> parameters, int newValue = 0)
   {
       foreach (var p in parameters)
       {
           if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
           {
               while (newValue == 0)
               {
                   Console.WriteLine(
                       $"Enter a new value for {p.ParameterName} from the
allowed values {p.AllowedValues} ");

                   var choice = Console.ReadLine();
                   int.TryParse(choice, out newValue);
               }

               p.ParameterValue = newValue.ToString();
           }
       }

       var request = new ModifyDBClusterParameterGroupRequest
       {
           Parameters = parameters,
           DBClusterParameterGroupName = groupName,
       };

       var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
       return result.DBClusterParameterGroupName;
   }


   /// <summary>
   /// Get a list of orderable DB instance options for a specific
   /// engine and engine version.
   /// </summary>
   /// <param name="engine">Name of the engine.</param>
   /// <param name="engineVersion">Version of the engine.</param>
   /// <returns>List of OrderableDBInstanceOptions.</returns>
   public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
```

```
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
```

```
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
```

```
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
```

```
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });
```

```
        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });
```

```
        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für .NET - API-Referenz.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen
  - CreateDBInstance
  - LöschenDBCluster
  - LöschenDBClusterParameterGroup
  - LöschenDBInstance
  - Beschreiben DBCluster ParameterGroups
  - Beschreiben Sie die DBCluster Parameter
  - Beschreiben Sie DBCluster Schnappschüsse
  - Beschreiben DBClusters

- DBEngineVersionen beschreiben
- Beschreiben DBInstances
- DescribeOrderableDBInstanceOptionen
- Modifizieren SieDBClusterParameterGroup

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Routine which creates an Amazon Aurora DB cluster and demonstrates several
 operations
//! on that cluster.
/*!
 \sa gettingStartedWithDBClusters()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::gettingStartedWithDBClusters(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon
 Aurora)"
              << std::endl;
    std::cout << "get started with DB clusters demo." << std::endl;
    printAsterisksLine();

    std::cout << "Checking for an existing DB cluster parameter group named '" <<
              CLUSTER_PARAMETER_GROUP_NAME << "'." << std::endl;
```

```
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
    {
        // 1. Check if the DB cluster parameter group already exists.
        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' does not exist." <<
 std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available parameter group families for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }

        std::cout << "Getting available parameter group families for " <<
 DB_ENGINE
```

```
                  << "."
                  << std::endl;
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
    if (!parameterGroupFound) {
        // 3.  Create a DB cluster parameter group.
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    printAsterisksLine();
    std::cout << "Let's set some parameter values in your cluster parameter
group."
```

```
              << std::endl;

   Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
   // 4.  Get the parameters in the DB cluster parameter group.
   if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME,
AUTO_INCREMENT_PREFIX,
                               NO_SOURCE,
                               autoIncrementParameters,
                               client)) {
       cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
       return false;
   }

   Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

   for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
       if (autoIncParameter.GetIsModifiable() &&
           (autoIncParameter.GetDataType() == "integer")) {
           std::cout << "The " << autoIncParameter.GetParameterName()
                     << " is described as: " <<
                     autoIncParameter.GetDescription() << "." << std::endl;
           if (autoIncParameter.ParameterValueHasBeenSet()) {
               std::cout << "The current value is "
                         << autoIncParameter.GetParameterValue()
                         << "." << std::endl;
           }
           std::vector<int> splitValues = splitToInts(
                   autoIncParameter.GetAllowedValues(), '-');
           if (splitValues.size() == 2) {
               int newValue = askQuestionForIntRange(
                       Aws::String("Enter a new value between ") +
                       autoIncParameter.GetAllowedValues() + ": ",
                       splitValues[0], splitValues[1]);
               autoIncParameter.SetParameterValue(std::to_string(newValue));
               updateParameters.push_back(autoIncParameter);

           }
           else {
               std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                         << std::endl;
           }
       }
   }
```

```
   {
       // 5.  Modify the auto increment parameters in the DB cluster parameter
group.
       Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
       request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
       request.SetParameters(updateParameters);

       Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
               client.ModifyDBClusterParameterGroup(request);

       if (outcome.IsSuccess()) {
           std::cout << "The DB cluster parameter group was successfully
modified."
                     << std::endl;
       }
       else {
           std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
       }
   }

   std::cout
           << "You can get a list of parameters you've set by specifying a
source of 'user'."
           << std::endl;

   Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
   // 6.  Display the modified parameters in the DB cluster parameter group.
   if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME, NO_NAME_PREFIX,
"user",
                               userParameters,
                               client)) {
       cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
       return false;
   }

   for (const auto &userParameter: userParameters) {
       std::cout << "  " << userParameter.GetParameterName() << ", " <<
                 userParameter.GetDescription() << ", parameter value - "
                 << userParameter.GetParameterValue() << std::endl;
   }
```

```
    printAsterisksLine();
    std::cout << "Checking for an existing DB Cluster." << std::endl;

    Aws::RDS::Model::DBCluster dbCluster;
    // 7.  Check if the DB cluster already exists.
    if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::String engineVersionName;
    Aws::String engineName;
    if (dbCluster.DBClusterIdentifierHasBeenSet()) {
        std::cout << "The DB cluster already exists." << std::endl;
        engineVersionName = dbCluster.GetEngineVersion();
        engineName = dbCluster.GetEngine();

    }
    else {
        std::cout << "Let's create a DB cluster." << std::endl;
        const Aws::String administratorName = askQuestion(
                "Enter an administrator username for the database: ");
        const Aws::String administratorPassword = askQuestion(
                "Enter a password for the administrator (at least 8 characters):
");
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 8.  Get a list of engine versions for the parameter group family.
        if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                 client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }

        std::cout << "The available engines for your parameter group family are:"
                  << std::endl;

        int index = 1;
        for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
            std::cout << "   " << index << ": " <<
engineVersion.GetEngineVersion()
                      << std::endl;
```

```
            ++index;
        }
        int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
        const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                              1];

        engineName = engineVersion.GetEngine();
        engineVersionName = engineVersion.GetEngineVersion();
        std::cout << "Creating a DB cluster named '" << DB_CLUSTER_IDENTIFIER
                  << "' and database '" << DB_NAME << "'.\n"
                  << "The DB cluster is configured to use your custom cluster
parameter group '"
                  << CLUSTER_PARAMETER_GROUP_NAME << "', and \n"
                  << "selected engine version " <<
engineVersion.GetEngineVersion()
                  << ".\nThis typically takes several minutes." << std::endl;

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
    }
```

```
    std::cout << "Waiting for the DB cluster to become available." << std::endl;

    int counter = 0;
    // 11. Wait for the DB cluster to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for cluster to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        dbCluster = Aws::RDS::Model::DBCluster();
        if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB cluster status is '"
                      << dbCluster.GetStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbCluster.GetStatus() != "available");

    if (dbCluster.GetStatus() == "available") {
        std::cout << "The DB cluster has been created." << std::endl;
    }

    printAsterisksLine();
    Aws::RDS::Model::DBInstance dbInstance;
    // 11.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER, "",
                         client);
        return false;
    }
```

```
    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
    }
    else {
        std::cout << "Let's create a DB instance." << std::endl;

        Aws::String dbInstanceClass;
        // 12.  Get a list of instance classes.
        if (!chooseDBInstanceClass(engineName,
                                   engineVersionName,
                                   dbInstanceClass,
                                   client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }

        std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                  << "' with selected DB instance class '" << dbInstanceClass
                  << "'.\nThis typically takes several minutes." << std::endl;

        // 13. Create a DB instance.
        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
```

```
        }
    }

    std::cout << "Waiting for the DB instance to become available." << std::endl;

    counter = 0;
    // 14. Wait for the DB instance to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for instance to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        dbInstance = Aws::RDS::Model::DBInstance();
        if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB instance status is '"
                      << dbInstance.GetDBInstanceStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbInstance.GetDBInstanceStatus() != "available");

    if (dbInstance.GetDBInstanceStatus() == "available") {
        std::cout << "The DB instance has been created." << std::endl;
    }

    // 15. Display the connection string that can be used to connect a 'mysql'
shell to the database.
    displayConnection(dbCluster);

    printAsterisksLine();
```

```
   if (askYesNoQuestion(
           "Do you want to create a snapshot of your DB cluster (y/n)? ")) {
       Aws::String snapshotID(DB_CLUSTER_IDENTIFIER + "-" +
                              Aws::String(Aws::Utils::UUID::RandomUUID()));
       {
           std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
           std::cout << "This typically takes a few minutes." << std::endl;

           // 16. Create a snapshot of the DB cluster. (CreateDBClusterSnapshot)
           Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
           request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
           request.SetDBClusterSnapshotIdentifier(snapshotID);

           Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                   client.CreateDBClusterSnapshot(request);

           if (outcome.IsSuccess()) {
               std::cout << "Snapshot creation has started."
                         << std::endl;
           }
           else {
               std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                         << outcome.GetError().GetMessage()
                         << std::endl;
               cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
               return false;
           }
       }

       std::cout << "Waiting for the snapshot to become available." <<
std::endl;

       Aws::RDS::Model::DBClusterSnapshot snapshot;
       counter = 0;
       do {
           std::this_thread::sleep_for(std::chrono::seconds(1));
           ++counter;
           if (counter > 600) {
               std::cerr << "Wait for snapshot to be available timed out ofter "
                         << counter
```

```
                        << " seconds." << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            // 17. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }

    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
```

```
            "Do you want to delete the DB cluster, DB instance, and parameter
 group (y/n)? ")) {
        result = cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                  DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
                                  client);
    }

    return result;
}

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }
```

```
    return result;

}


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                             const Aws::String &namePrefix,
                                             const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                             const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
```

```
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
```

```
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
```

```
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
```

```
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}

//! Routine which deletes resources created by the scenario.
```

```
/*!
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::Aurora::cleanUpResources(const Aws::String &parameterGroupName,
                                      const Aws::String &dbClusterIdentifier,
                                      const Aws::String &dbInstanceIdentifier,
                                      const Aws::RDS::RDSClient &client) {
    bool result = true;
    bool instanceDeleting = false;
    bool clusterDeleting = false;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 18. Delete the DB instance.
            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
 parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    if (!dbClusterIdentifier.empty()) {
```

```
        {
            // 19. Delete the DB cluster.
            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }
    int counter = 0;

    while (clusterDeleting || instanceDeleting) {
        // 20. Wait for the DB cluster and instance to be deleted.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 800) {
            std::cerr << "Wait for instance to delete timed out ofter " <<
counter
                      << " seconds." << std::endl;
            return false;
        }

        Aws::RDS::Model::DBInstance dbInstance = Aws::RDS::Model::DBInstance();
        if (instanceDeleting) {
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
```

```
            }
            instanceDeleting = dbInstance.DBInstanceIdentifierHasBeenSet();
        }

        Aws::RDS::Model::DBCluster dbCluster = Aws::RDS::Model::DBCluster();
        if (clusterDeleting) {
            if (!describeDBCluster(dbClusterIdentifier, dbCluster, client)) {
                return false;
            }

            clusterDeleting = dbCluster.DBClusterIdentifierHasBeenSet();
        }

        if ((counter % 20) == 0) {
            if (instanceDeleting) {
                std::cout << "Current DB instance status is '"
                          << dbInstance.GetDBInstanceStatus() << "." <<
std::endl;
            }

            if (clusterDeleting) {
                std::cout << "Current DB cluster status is '"
                          << dbCluster.GetStatus() << "." << std::endl;
            }
        }
    }

    if (!parameterGroupName.empty()) {
        // 21. Delete the DB cluster parameter group.
        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
```

```
        }
    }

    return result;
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für C++ -API-Referenz.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen
  - CreateDBInstance
  - LöschenDBCluster
  - LöschenDBClusterParameterGroup
  - LöschenDBInstance
  - Beschreiben DBCluster ParameterGroups
  - Beschreiben Sie die DBCluster Parameter
  - Beschreiben Sie DBCluster Schnappschüsse
  - Beschreiben DBClusters
  - DBEngineVersionen beschreiben
  - Beschreiben DBInstances
  - DescribeOrderableDBInstanceOptionen
  - Modifizieren SieDBClusterParameterGroup

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "aurora/actions"
 "context"
 "fmt"
 "log"
 "slices"
 "sort"
 "strconv"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/google/uuid"
)

// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service
 (Amazon RDS)
// client and create wrappers for the actions used in the scenario.
```

```
func NewGetStartedClusters(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine,
 parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 cluster := scenario.CreateCluster(ctx, clusterName, dbEngine, dbName,
 parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(ctx, cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(ctx, clusterName)
 scenario.Cleanup(ctx, dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
```

```
// database engine and create a DB cluster parameter group that is compatible
 with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
```

```
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedClusters) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(ctx, parameterGroupName,
 "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.dbClusters.UpdateParameters(ctx, parameterGroupName,
 updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("You can get a list of parameters you've set by specifying a source
 of 'user'.")
```

```
 userParameters, err := scenario.dbClusters.GetParameters(ctx,
 parameterGroupName, "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you've set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a
 database
// of a specified type. The database is also configured to use a custom DB
 cluster
// parameter group.
func (scenario GetStartedClusters) CreateCluster(ctx context.Context, clusterName
 string, dbEngine string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
 cluster, err := scenario.dbClusters.GetDbCluster(ctx, clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
   "Enter a password for the administrator (at least 8 characters): ",
 demotools.NotEmpty{})
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine,
 *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  log.Println("The available engines for your parameter group are:")
  engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?
\n", engineChoices)
```

```
  log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
  log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
\n",
   *parameterGroup.DBClusterParameterGroupName)
  log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
  log.Println("This typically takes several minutes.")
  cluster, err = scenario.dbClusters.CreateDbCluster(
   ctx, clusterName, *parameterGroup.DBClusterParameterGroupName, dbName,
 dbEngine,
   engineChoices[engineIndex], adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *cluster.Status != "available" {
   scenario.helper.Pause(30)
   cluster, err = scenario.dbClusters.GetDbCluster(ctx, clusterName)
   if err != nil {
    panic(err)
   }
   log.Println("Cluster created and available.")
  }
 }
 log.Println("Cluster data:")
 log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
 log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
 log.Printf("\tStatus: %v\n", *cluster.Status)
 log.Printf("\tEngine: %v\n", *cluster.Engine)
 log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
 log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
 log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
 log.Println(strings.Repeat("-", 88))
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(ctx context.Context, cluster
 *types.DBCluster) *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(ctx,
 *cluster.DBClusterIdentifier)
```

```
if err != nil {
 panic(err)
}
if dbInstance == nil {
 log.Println("Let's create a database instance in your DB cluster.")
 log.Println("First, choose a DB instance type:")
 instOpts, err := scenario.dbClusters.GetOrderableInstances(
  ctx, *cluster.Engine, *cluster.EngineVersion)
 if err != nil {
  panic(err)
 }
 var instChoices []string
 for _, opt := range instOpts {
  instChoices = append(instChoices, *opt.DBInstanceClass)
 }
 slices.Sort(instChoices)
 instChoices = slices.Compact(instChoices)
 instIndex := scenario.questioner.AskChoice(
  "Which DB instance class do you want to use?\n", instChoices)
 log.Println("Creating a database instance. This typically takes several
minutes.")
 dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
  ctx, *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier,
*cluster.Engine,
  instChoices[instIndex])
 if err != nil {
  panic(err)
 }
 for *dbInstance.DBInstanceStatus != "available" {
  scenario.helper.Pause(30)
  dbInstance, err = scenario.dbClusters.GetInstance(ctx,
*cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
 }
}
log.Println("Instance data:")
log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
log.Printf("\tEngine: %v\n", *dbInstance.Engine)
log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
log.Println(strings.Repeat("-", 88))
```

```
 return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster
 and tips
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
 log.Println(
  "You can now connect to your database using your favorite MySql client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
 +
   "that is running in the same VPC as your database cluster. Pass the endpoint,
\n" +
   "port, and administrator user name to 'mysql' and enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
 log.Println("For more information, see the User Guide for Aurora:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(ctx context.Context,
 clusterName string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(ctx, clusterName,
 snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(ctx, snapshotId)
   if err != nil {
    panic(err)
   }
  }
```

```
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster
 parameter group.
// Before the DB cluster parameter group can be deleted, all associated DB
 instances and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(ctx context.Context, dbInstance
 *types.DBInstance, cluster *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n",
 *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(ctx, *cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
```

```
    dbInstance, err = scenario.dbClusters.GetInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(ctx,
 *cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err = scenario.dbClusters.DeleteParameterGroup(ctx,
 *parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}

// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 UniqueId() string
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

Definieren Sie Funktionen, die vom Szenario aufgerufen werden, um Aurora-Aktionen zu verwalten.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

```
// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
```

```
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName
 string, parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
```

```
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context,
 clusterName string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

```
// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context,
 clusterName string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}



// GetInstance gets data about a DB instance.
```

```
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx,
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
```

```
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
```

```
 return instances, err
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen
  - CreateDBInstance
  - LöschenDBCluster
  - LöschenDBClusterParameterGroup
  - LöschenDBInstance
  - Beschreiben DBCluster ParameterGroups
  - Beschreiben Sie die DBCluster Parameter
  - Beschreiben Sie DBCluster Schnappschüsse
  - Beschreiben DBClusters
  - DBEngineVersionen beschreiben
  - Beschreiben DBInstances
  - DescribeOrderableDBInstanceOptionen
  - Modifizieren SieDBClusterParameterGroup

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/**
 * Before running this Java (v2) code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
 * This Java example performs the following tasks:
 *
 * 1. Gets available engine families for Amazon Aurora MySQL-Compatible Edition
 * by calling the DescribeDbEngineVersions(Engine='aurora-mysql') method.
 * 2. Selects an engine family and creates a custom DB cluster parameter group
 * by invoking the describeDBClusterParameters method.
 * 3. Gets the parameter groups by invoking the describeDBClusterParameterGroups
 * method.
 * 4. Gets parameters in the group by invoking the describeDBClusterParameters
 * method.
 * 5. Modifies the auto_increment_offset parameter by invoking the
 * modifyDbClusterParameterGroupRequest method.
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions by invoking the
 * describeDbEngineVersions method.
 * 8. Creates an Aurora DB cluster database cluster that contains a MySQL
 * database.
 * 9. Waits for DB instance to be ready.
 * 10. Gets a list of instance classes available for the selected engine.
 * 11. Creates a database instance in the cluster.
 * 12. Waits for DB instance to be ready.
 * 13. Creates a snapshot.
 * 14. Waits for DB snapshot to be ready.
 * 15. Deletes the DB cluster.
 * 16. Deletes the DB cluster group.
 */
public class AuroraScenario {
    public static long sleepTime = 20;
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");
```

```
   public static void main(String[] args) throws InterruptedException {
       final String usage = "\n" +
               "Usage:\n" +
               "    <dbClusterGroupName> <dbParameterGroupFamily>
<dbInstanceClusterIdentifier> <dbInstanceIdentifier> <dbName>
<dbSnapshotIdentifier><secretName>"
               +
               "Where:\n" +
               "    dbClusterGroupName - The name of the DB cluster parameter
group. \n" +
               "    dbParameterGroupFamily - The DB cluster parameter group
family name (for example, aurora-mysql5.7). \n"
               +
               "    dbInstanceClusterIdentifier - The instance cluster
identifier value.\n" +
               "    dbInstanceIdentifier - The database instance identifier.\n"
+
               "    dbName - The database name.\n" +
               "    dbSnapshotIdentifier - The snapshot identifier.\n" +
               "    secretName - The name of the AWS Secrets Manager secret that
contains the database credentials\"\n";
       ;

       if (args.length != 7) {
           System.out.println(usage);
           System.exit(1);
       }

       String dbClusterGroupName = args[0];
       String dbParameterGroupFamily = args[1];
       String dbInstanceClusterIdentifier = args[2];
       String dbInstanceIdentifier = args[3];
       String dbName = args[4];
       String dbSnapshotIdentifier = args[5];
       String secretName = args[6];

       // Retrieve the database credentials using AWS Secrets Manager.
       Gson gson = new Gson();
       User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
       String username = user.getUsername();
       String userPassword = user.getPassword();
```

```
        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Aurora example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBClusterParameterGroup(rdsClient, dbClusterGroupName,
dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbClusterParameterGroups(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the parameters in the group");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBClusterParas(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated parameter value");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Get a list of allowed engine versions");
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("8. Create an Aurora DB cluster database");
        String arnClusterVal = createDBCluster(rdsClient, dbClusterGroupName,
dbName, dbInstanceClusterIdentifier,
                username, userPassword);
        System.out.println("The ARN of the cluster is " + arnClusterVal);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get a list of instance classes available for the
selected engine");
        String instanceClass = getListInstanceClasses(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Create a database instance in the cluster.");
        String clusterDBARN = createDBInstanceCluster(rdsClient,
dbInstanceIdentifier, dbInstanceClusterIdentifier,
                instanceClass);
        System.out.println("The ARN of the database is " + clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB instance to be ready");
        waitDBInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Create a snapshot");
        createDBClusterSnapshot(rdsClient, dbInstanceClusterIdentifier,
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Wait for DB snapshot to be ready");
        waitForSnapshotReady(rdsClient, dbSnapshotIdentifier,
dbInstanceClusterIdentifier);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("14. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Delete the DB cluster");
        deleteCluster(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete the DB cluster group");
        deleteDBClusterGroup(rdsClient, dbClusterGroupName, clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)

.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
    }

    private static String getSecretValues(String secretName) {
        SecretsManagerClient secretClient = getSecretClient();
        GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                .secretId(secretName)
                .build();

        GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
        return valueResponse.secretString();
    }

    public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
```

```
            throws InterruptedException {
        try {
            boolean isDataDel = false;
            boolean didFind;
            String instanceARN;

            // Make sure that the database has been deleted.
            while (!isDataDel) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
                List<DBInstance> instanceList = response.dbInstances();
                int listSize = instanceList.size();
                didFind = false;
                int index = 1;
                for (DBInstance instance : instanceList) {
                    instanceARN = instance.dbInstanceArn();
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        System.out.println(clusterDBARN + " still exists");
                        didFind = true;
                    }
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
```

```
    }

    public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
            String dbInstanceClusterIdentifier) {
```

```
        try {
            boolean snapshotReady = false;
            String snapshotReadyStr;
            System.out.println("Waiting for the snapshot to become available.");

            DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .build();

            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
                List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
            String dbSnapshotIdentifier) {
        try {
            CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();
```

```
            CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitDBInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection
endpoint is " + endpoint);

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
    }

    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static String getListInstanceClasses(RdsClient rdsClient) {
        try {
            DescribeOrderableDbInstanceOptionsRequest optionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                    .builder()
                    .engine("aurora-mysql")
                    .maxRecords(20)
                    .build();

            DescribeOrderableDbInstanceOptionsResponse response = rdsClient
                    .describeOrderableDBInstanceOptions(optionsRequest);
            List<OrderableDBInstanceOption> instanceOptions =
response.orderableDBInstanceOptions();
            String instanceClass = "";
```

```
            for (OrderableDBInstanceOption instanceOption : instanceOptions) {
                instanceClass = instanceOption.dbInstanceClass();
                System.out.println("The instance class is " +
instanceOption.dbInstanceClass());
                System.out.println("The engine version is " +
instanceOption.engineVersion());
            }
            return instanceClass;

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");
```

```
        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
                    .build();

            CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
```

```
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify the auto_increment_offset parameter.
    public static void modifyDBClusterParas(RdsClient rdsClient, String
dClusterGroupName) {
        try {
            Parameter parameter1 = Parameter.builder()
                    .parameterName("auto_increment_offset")
                    .applyMethod("immediate")
                    .parameterValue("5")
                    .build();

            List<Parameter> paraList = new ArrayList<>();
            paraList.add(parameter1);
            ModifyDbClusterParameterGroupRequest groupRequest =
ModifyDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dClusterGroupName)
                    .parameters(paraList)
                    .build();

            ModifyDbClusterParameterGroupResponse response =
rdsClient.modifyDBClusterParameterGroup(groupRequest);
            System.out.println(
                    "The parameter group " +
response.dbClusterParameterGroupName() + " was successfully modified");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
               // auto_increment_increment.
               paraName = para.parameterName();
               if ((paraName.compareTo("auto_increment_offset") == 0)
                       || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                   System.out.println("*** The parameter name is  " + paraName);
                   System.out.println("*** The parameter value is  " +
para.parameterValue());
                   System.out.println("*** The parameter data type is " +
para.dataType());
                   System.out.println("*** The parameter description is " +
para.description());
                   System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
               }
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
```

```
        }
    }

    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
            CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
            System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
 DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
 rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - CreateDBCluster
  - CreateDBClusterParameterGroup

- DBClusterSchnappschuss erstellen
- CreateDBInstance
- LöschenDBCluster
- LöschenDBClusterParameterGroup
- LöschenDBInstance
- Beschreiben DBCluster ParameterGroups
- Beschreiben Sie die DBCluster Parameter
- Beschreiben Sie DBCluster Schnappschüsse
- Beschreiben DBClusters
- DBEngineVersionen beschreiben
- Beschreiben DBInstances
- DescribeOrderableDBInstanceOptionen
- Modifizieren SieDBClusterParameterGroup

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:
```

```
https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This Kotlin example performs the following tasks:

1. Returns a list of the available DB engines.
2. Creates a custom DB parameter group.
3. Gets the parameter groups.
4. Gets the parameters in the group.
5. Modifies the auto_increment_increment parameter.
6. Displays the updated parameter value.
7. Gets a list of allowed engine versions.
8. Creates an Aurora DB cluster database.
9. Waits for DB instance to be ready.
10. Gets a list of instance classes available for the selected engine.
11. Creates a database instance in the cluster.
12. Waits for the database instance in the cluster to be ready.
13. Creates a snapshot.
14. Waits for DB snapshot to be ready.
15. Deletes the DB instance.
16. Deletes the DB cluster.
17. Deletes the DB cluster group.
 */

var slTime: Long = 20

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbName> <dbSnapshotIdentifier> <secretName>
        Where:
            dbClusterGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceClusterIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """

    if (args.size != 7) {
        println(usage)
```

```
        exitProcess(1)
    }

    val dbClusterGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceClusterIdentifier = args[2]
    val dbInstanceIdentifier = args[3]
    val dbName = args[4]
    val dbSnapshotIdentifier = args[5]
    val secretName = args[6]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
User::class.java)
    val username = user.username
    val userPassword = user.password

    println("1. Return a list of the available DB engines")
    describeAuroraDBEngines()

    println("2. Create a custom parameter group")
    createDBClusterParameterGroup(dbClusterGroupName, dbParameterGroupFamily)

    println("3. Get the parameter group")
    describeDbClusterParameterGroups(dbClusterGroupName)

    println("4. Get the parameters in the group")
    describeDbClusterParameters(dbClusterGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBClusterParas(dbClusterGroupName)

    println("6. Display the updated parameter value")
    describeDbClusterParameters(dbClusterGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedClusterEngines(dbParameterGroupFamily)

    println("8. Create an Aurora DB cluster database")
    val arnClusterVal = createDBCluster(dbClusterGroupName, dbName,
dbInstanceClusterIdentifier, username, userPassword)
    println("The ARN of the cluster is $arnClusterVal")

    println("9. Wait for DB instance to be ready")
```

```
    waitForClusterInstanceReady(dbInstanceClusterIdentifier)

    println("10. Get a list of instance classes available for the selected
 engine")
    val instanceClass = getListInstanceClasses()

    println("11. Create a database instance in the cluster.")
    val clusterDBARN = createDBInstanceCluster(dbInstanceIdentifier,
 dbInstanceClusterIdentifier, instanceClass)
    println("The ARN of the database is $clusterDBARN")

    println("12. Wait for DB instance to be ready")
    waitDBAuroraInstanceReady(dbInstanceIdentifier)

    println("13. Create a snapshot")
    createDBClusterSnapshot(dbInstanceClusterIdentifier, dbSnapshotIdentifier)

    println("14. Wait for DB snapshot to be ready")
    waitSnapshotReady(dbSnapshotIdentifier, dbInstanceClusterIdentifier)

    println("15. Delete the DB instance")
    deleteDBInstance(dbInstanceIdentifier)

    println("16. Delete the DB cluster")
    deleteCluster(dbInstanceClusterIdentifier)

    println("17. Delete the DB cluster group")
    if (clusterDBARN != null) {
        deleteDBClusterGroup(dbClusterGroupName, clusterDBARN)
    }
    println("The Scenario has successfully completed.")
}

@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
```

```
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}

suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
```

```
    }
}

suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}

suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
```

```
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}

suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}

suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
```

```
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}

suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
            dbInstanceClass = instanceClassVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}

suspend fun getListInstanceClasses(): String {
    val optionsRequest =
        DescribeOrderableDbInstanceOptionsRequest {
            engine = "aurora-mysql"
            maxRecords = 20
        }
    var instanceClass = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(optionsRequest)
        response.orderableDbInstanceOptions?.forEach { instanceOption ->
            instanceClass = instanceOption.dbInstanceClass.toString()
            println("The instance class is ${instanceOption.dbInstanceClass}")
            println("The engine version is ${instanceOption.engineVersion}")
```

```
        }
    }
    return instanceClass
}

// Waits until the database instance is available.
suspend fun waitForClusterInstanceReady(dbClusterIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest =
        DescribeDbClustersRequest {
            dbClusterIdentifier = dbClusterIdentifierVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbClusters(instanceRequest)
            response.dbClusters?.forEach { cluster ->
                instanceReadyStr = cluster.status.toString()
                if (instanceReadyStr.contains("available")) {
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database cluster is available!")
}

suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
```

```
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}

// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
```

```
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}

suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
```

```
            }
        }
    }
}

suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}

suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}

suspend fun describeAuroraDBEngines() {
    val engineVersionsRequest =
        DescribeDbEngineVersionsRequest {
            engine = "aurora-mysql"
            defaultOnly = true
```

```
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        response.dbEngineVersions?.forEach { engineOb ->
            println("The name of the DB parameter group family for the database
 engine is ${engineOb.dbParameterGroupFamily}")
            println("The name of the database engine ${engineOb.engine}")
            println("The version number of the database engine
 ${engineOb.engineVersion}")
        }
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für Kotlin.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen
  - CreateDBInstance
  - LöschenDBCluster
  - LöschenDBClusterParameterGroup
  - LöschenDBInstance
  - Beschreiben DBCluster ParameterGroups
  - Beschreiben Sie die DBCluster Parameter
  - Beschreiben Sie DBCluster Schnappschüsse
  - Beschreiben DBClusters
  - DBEngineVersionen beschreiben
  - Beschreiben DBInstances
  - DescribeOrderableDBInstanceOptionen
  - Modifizieren SieDBClusterParameterGroup

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
class AuroraClusterScenario:
    """Runs a scenario that shows how to get started using Aurora DB clusters."""

    def __init__(self, aurora_wrapper):
        """
        :param aurora_wrapper: An object that wraps Aurora DB cluster actions.
        """
        self.aurora_wrapper = aurora_wrapper

    def create_parameter_group(self, db_engine, parameter_group_name):
        """
        Shows how to get available engine versions for a specified database
 engine and
        create a DB cluster parameter group that is compatible with a selected
 engine family.

        :param db_engine: The database engine to use as a basis.
        :param parameter_group_name: The name given to the newly created
 parameter group.
        :return: The newly created parameter group.
        """
        print(
            f"Checking for an existing DB cluster parameter group named
 {parameter_group_name}."
        )
        parameter_group =
 self.aurora_wrapper.get_parameter_group(parameter_group_name)
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions = self.aurora_wrapper.get_engine_versions(db_engine)
```

```
            families = list({ver["DBParameterGroupFamily"] for ver in
engine_versions})
            family_index = q.choose("Which family do you want to use? ",
families)
            print(f"Creating a DB cluster parameter group.")
            self.aurora_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
group."
            )
            parameter_group = self.aurora_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group
{parameter_group['DBClusterParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group

    def set_user_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.

        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
```

```
                )
                update_params.append(auto_inc)
        self.aurora_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, source="user"
        )
        pp(user_parameters)
        print("-" * 88)

    def create_cluster(self, cluster_name, db_engine, db_name, parameter_group):
        """
        Shows how to create an Aurora DB cluster that contains a database of a
specified
        type. The database is also configured to use a custom DB cluster
parameter group.

        :param cluster_name: The name given to the newly created DB cluster.
        :param db_engine: The engine of the created database.
        :param db_name: The name given to the created database.
        :param parameter_group: The parameter group that is associated with the
DB cluster.
        :return: The newly created DB cluster.
        """
        print("Checking for an existing DB cluster.")
        cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
        if cluster is None:
            admin_username = q.ask(
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.aurora_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [
```

```
                ver["EngineVersionDescription"] for ver in engine_versions
            ]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            print(
                f"Creating DB cluster {cluster_name} and database {db_name}.\n"
                f"The DB cluster is configured to use\n"
                f"your custom parameter group
{parameter_group['DBClusterParameterGroupName']}\n"
                f"and selected engine {engine_choices[engine_index]}.\n"
                f"This typically takes several minutes."
            )
            cluster = self.aurora_wrapper.create_db_cluster(
                cluster_name,
                parameter_group["DBClusterParameterGroupName"],
                db_name,
                db_engine,
                engine_versions[engine_index]["EngineVersion"],
                admin_username,
                admin_password,
            )
            while cluster.get("Status") != "available":
                wait(30)
                cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
            print("Cluster created and available.\n")
        print("Cluster data:")
        pp(cluster)
        print("-" * 88)
        return cluster

    def create_instance(self, cluster):
        """
        Shows how to create a DB instance in an existing Aurora DB cluster. A new
DB cluster
        contains no DB instances, so you must add one. The first DB instance that
is added
        to a DB cluster defaults to a read-write DB instance.

        :param cluster: The DB cluster where the DB instance is added.
        :return: The newly created DB instance.
        """
        print("Checking for an existing database instance.")
        cluster_name = cluster["DBClusterIdentifier"]
```

```
        db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        if db_inst is None:
            print("Let's create a database instance in your DB cluster.")
            print("First, choose a DB instance type:")
            inst_opts = self.aurora_wrapper.get_orderable_instances(
                cluster["Engine"], cluster["EngineVersion"]
            )
            inst_choices = list(
                {
                    opt["DBInstanceClass"] + ", storage type: " +
opt["StorageType"]
                    for opt in inst_opts
                }
            )
            inst_index = q.choose(
                "Which DB instance class do you want to use? ", inst_choices
            )
            print(
                f"Creating a database instance. This typically takes several
minutes."
            )
            db_inst = self.aurora_wrapper.create_instance_in_cluster(
                cluster_name,
                cluster_name,
                cluster["Engine"],
                inst_opts[inst_index]["DBInstanceClass"],
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(30)
                db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst

    @staticmethod
    def display_connection(cluster):
        """
        Displays connection information about an Aurora DB cluster and tips on
how to
        connect to it.

        :param cluster: The DB cluster to display.
        """
```

```
        print(
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your database cluster. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {cluster['Endpoint']} -P {cluster['Port']} -u
 {cluster['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Aurora:\n"
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
        )
        print("-" * 88)

    def create_snapshot(self, cluster_name):
        """
        Shows how to create a DB cluster snapshot and wait until it's available.

        :param cluster_name: The name of a DB cluster to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB cluster (y/n)? ",
 q.is_yesno
        ):
            snapshot_id = f"{cluster_name}-{uuid.uuid4()}"
            print(
                f"Creating a snapshot named {snapshot_id}. This typically takes a
 few minutes."
            )
            snapshot = self.aurora_wrapper.create_cluster_snapshot(
                snapshot_id, cluster_name
            )
            while snapshot.get("Status") != "available":
                wait(30)
                snapshot = self.aurora_wrapper.get_cluster_snapshot(snapshot_id)
            pp(snapshot)
```

```
            print("-" * 88)

    def cleanup(self, db_inst, cluster, parameter_group):
        """
        Shows how to clean up a DB instance, DB cluster, and DB cluster parameter
group.
        Before the DB cluster parameter group can be deleted, all associated DB
instances and
        DB clusters must first be deleted.

        :param db_inst: The DB instance to delete.
        :param cluster: The DB cluster to delete.
        :param parameter_group: The DB cluster parameter group to delete.
        """
        cluster_name = cluster["DBClusterIdentifier"]
        parameter_group_name = parameter_group["DBClusterParameterGroupName"]
        if q.ask(
            "\nDo you want to delete the database instance, DB cluster, and
parameter "
            "group (y/n)? ",
            q.is_yesno,
        ):
            print(f"Deleting database instance
{db_inst['DBInstanceIdentifier']}.")

self.aurora_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
            print(f"Deleting database cluster {cluster_name}.")
            self.aurora_wrapper.delete_db_cluster(cluster_name)
            print(
                "Waiting for the DB instance and DB cluster to delete.\n"
                "This typically takes several minutes."
            )
            while db_inst is not None or cluster is not None:
                wait(30)
                if db_inst is not None:
                    db_inst = self.aurora_wrapper.get_db_instance(
                        db_inst["DBInstanceIdentifier"]
                    )
                if cluster is not None:
                    cluster = self.aurora_wrapper.get_db_cluster(
                        cluster["DBClusterIdentifier"]
                    )
            print(f"Deleting parameter group {parameter_group_name}.")
            self.aurora_wrapper.delete_parameter_group(parameter_group_name)
```

```
    def run_scenario(self, db_engine, parameter_group_name, cluster_name,
 db_name):
        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) get
 started\n"
            "with Aurora DB clusters demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(db_engine,
 parameter_group_name)
        self.set_user_parameters(parameter_group_name)
        cluster = self.create_cluster(cluster_name, db_engine, db_name,
 parameter_group)
        wait(5)
        db_inst = self.create_instance(cluster)
        self.display_connection(cluster)
        self.create_snapshot(cluster_name)
        self.cleanup(db_inst, cluster, parameter_group)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    try:
        scenario = AuroraClusterScenario(AuroraWrapper.from_client())
        scenario.run_scenario(
            "aurora-mysql",
            "doc-example-cluster-parameter-group",
            "doc-example-aurora",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

Definieren Sie Funktionen, die vom Szenario aufgerufen werden, um Aurora-Aktionen zu verwalten.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
 parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        else:
            return parameter_group


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
```

```
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
```

```
                if p["ParameterName"].startswith(name_prefix)
            ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
```

```
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
```

```
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
```

```
            raise


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
```

```
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return db_inst


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
```

```
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
```

```
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für Python (Boto3).
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen

- CreateDBInstance
- LöschenDBCluster
- LöschenDBClusterParameterGroup
- LöschenDBInstance
- Beschreiben DBCluster ParameterGroups
- Beschreiben Sie die DBCluster Parameter
- Beschreiben Sie DBCluster Schnappschüsse
- Beschreiben DBClusters
- DBEngineVersionen beschreiben
- Beschreiben DBInstances
- DescribeOrderableDBInstanceOptionen
- Modifizieren SieDBClusterParameterGroup

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Eine Bibliothek, die die Szenarienpezifischen Funktionen für das Aurora-Szenario enthält.

```
use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

 operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
 Parameter},
};
```

```
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};

const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str =
 "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
struct MetadataError {
    message: Option<String>,
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
}

#[derive(Debug, PartialEq, Eq)]
```

```
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
            context: Some(MetadataError::from(err)),
        }
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display
 them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}

impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
```

```
            f,
            "{}: {} (allowed: {})",
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}

pub struct AuroraScenario {
    rds: crate::rds::Rds,
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
        }
    }
```

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");

       if let Err(err) = describe_db_engine_versions {
           return Err(ScenarioError::new(
               "Failed to retrieve DB Engine Versions",
               &err,
           ));
       };

       let version_count = describe_db_engine_versions
           .as_ref()
           .map(|o| o.db_engine_versions().len())
           .unwrap_or_default();
       info!(version_count, "got list of versions");

       // Create a map of engine families to their available versions.
       let mut versions = HashMap::<String, Vec<String>>::new();
       describe_db_engine_versions
           .unwrap()
           .db_engine_versions()
           .iter()
           .filter_map(
               |v| match (&v.db_parameter_group_family, &v.engine_version) {
                   (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                   _ => None,
               },
           )
           .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

       Ok(versions)
   }

   pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
       let describe_orderable_db_instance_options_items = self
```

```
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .filter(|o| o.storage_type() == Some("aurora"))
                    .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }

    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
```

```
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub fn set_instance_class(&mut self, instance_class: Option<String>) {
        self.instance_class = instance_class;
    }

    pub fn set_login(&mut self, username: Option<String>, password:
Option<SecretString>) {
        self.username = username;
        self.password = password;
    }

    pub async fn connection_string(&self) -> Result<String, ScenarioError> {
        let cluster = self.get_cluster().await?;
        let endpoint = cluster.endpoint().unwrap_or_default();
        let port = cluster.port().unwrap_or_default();
        let username = cluster.master_username().unwrap_or_default();
        Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
    }

    pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
        let describe_db_clusters_output = self
            .rds
            .describe_db_clusters(
```

```
                self.db_cluster_identifier
                    .as_ref()
                    .expect("cluster identifier")
                    .as_str(),
            )
            .await;
        if let Err(err) = describe_db_clusters_output {
            return Err(ScenarioError::new("Failed to get cluster", &err));
        }

        let db_cluster = describe_db_clusters_output
            .unwrap()
            .db_clusters
            .and_then(|output| output.first().cloned());

        db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
    }

    // Get the parameter group. rds.DescribeDbClusterParameterGroups
    // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
    // Parse the ParameterName, Description, and AllowedValues values and display
them.
    pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
        let parameters_output = self
            .rds
            .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
            .await;

        if let Err(err) = parameters_output {
            return Err(ScenarioError::new(
                format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
                &err,
            ));
        }

        let parameters = parameters_output
            .unwrap()
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
```

```
            .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }

    // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }
```

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
Status == 'available'.
    pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
        let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
        let snapshot = self
            .rds
            .snapshot_cluster(id, format!("{id}_{name}").as_str())
            .await;
        match snapshot {
            Ok(output) => match output.db_cluster_snapshot {
                Some(snapshot) => Ok(snapshot),
                None => Err(ScenarioError::with("Missing Snapshot")),
            },
            Err(err) => Err(ScenarioError::new("Failed to create snapshot",
&err)),
        }
    }

    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
```

```
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
```

```
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
```

```
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
```

```
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }
}

#[cfg(test)]
pub mod tests;
```

Tests für die Bibliothek, die Automocks für den RDS-Client-Wrapper nutzen.

```
use crate::rds::MockRdsImpl;

use super::*;

use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
```

```
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
 DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
use secrecy::ExposeSecret;

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });
```

```
    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

.db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
```

```
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}

#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
```

```
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Did not find the cluster");
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
```

```
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Failed to get cluster");
}

#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
```

```
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let connection_string = scenario.connection_string().await;

    assert_eq!(
        connection_string,
        Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
```

```
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
```

```
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);
```

```
    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
```

```
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
```

```
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
```

```
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
```

```
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

```
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
```

```
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
```

```
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
if message == "Failed to check instance state during deletion");
```

```
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")

 .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Failed to create snapshot");
}

#[tokio::test]
async fn test_scenario_snapshot_invalid() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _|
 Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Missing Snapshot");
}
```

Eine Binärdatei zum vollständigen Ausführen des Szenarien, wobei Inquirer verwendet wird, damit der Benutzer einige Entscheidungen treffen kann.

```
use std::fmt::Display;

use anyhow::anyhow;
use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
use secrecy::SecretString;
use tracing::warn;

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }

    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
```

```
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

// Prepare the Aurora Scenario. Prompt for several settings that are optional to
 the Scenario, but that the user should choose for the demo.
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario,
 anyhow::Error> {
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
        return Err(anyhow!("Failed to get available engines: {}", error));
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
 rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(),
 version.as_str()).await;
```

```
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
        Err(err) => return Err(anyhow!("Failed to get instance classes for
 engine: {err}")),
    }

    Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
 parameters based on user input.
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings)
 -> Result<(), ()> {
    show_parameters(scenario, warnings).await;

    let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
    let increment = prompt_number_or_default(warnings,
 "auto_increment_increment", 3);

    // Modify both the auto_increment_offset and auto_increment_increment
 parameters in one call in the custom parameter group. Set their ParameterValue
 fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    let update_auto_increment = scenario.update_auto_increment(offset,
 increment).await;

    if let Err(error) = update_auto_increment {
        warnings.push("Failed to update auto increment", error);
        return Err(());
    }

    // Get and display the updated parameters. Specify Source of 'user' to get
 just the modified parameters. rds.DescribeDbClusterParameters(Source='user')
```

```
    show_parameters(scenario, warnings).await;

    let username = inquire::Text::new("Username for the database (default
'testuser')")
        .with_default("testuser")
        .with_initial_value("testuser")
        .prompt();

    if let Err(error) = username {
        warnings.push(
            "Failed to get username, using default",
            ScenarioError::with(format!("Error from inquirer: {error}")),
        );
        return Err(());
    }
    let username = username.unwrap();

    let password = inquire::Text::new("Password for the database (minimum 8
characters)")
        .with_validator(|i: &str| {
            if i.len() >= 8 {
                Ok(inquire::validator::Validation::Valid)
            } else {
                Ok(inquire::validator::Validation::Invalid(
                    "Password must be at least 8 characters".into(),
                ))
            }
        })
        .prompt();

    let password: Option<SecretString> = match password {
        Ok(password) => Some(SecretString::from(password)),
        Err(error) => {
            warnings.push(
                "Failed to get password, using none (and not starting a DB)",
                ScenarioError::with(format!("Error from inquirer: {error}")),
            );
            return Err(());
        }
    };

    scenario.set_login(Some(username), password);

    Ok(())
```

```
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError>
 {
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;

    let connection_string = scenario.connection_string().await?;

    println!("Database ready: {connection_string}",);

    let _ = inquire::Text::new("Use the database with the connection string. When
 you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
 Status == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();
```

```
    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

    // Clean up the instance, cluster, and parameter group, waiting for the
 instance and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
    }
}

#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
 CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;
```

```
    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) ->
 u8 {
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
        .with_validator(U8Validator {})
        .prompt();

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

Ein Wrapper für den Service Amazon RDS, der Automocking für Tests ermöglicht.

```
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
            DeleteDBClusterParameterGroupError,
 DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{OrderableDbInstanceOption, Parameter},
    Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;
```

```
#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
        RdsImpl { inner }
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }

    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
 SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
```

```
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }
```

```
    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
```

```
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

    pub async fn describe_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner
            .describe_db_instances()
            .db_instance_identifier(instance_identifier)
            .send()
            .await
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
```

```
    ) -> Result<DescribeDbClusterEndpointsOutput,
SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
```

```
    }
}
```

Die Datei „Cargo.toml“ mit in diesem Szenario verwendeten Abhängigkeiten.

```
[package]
name = "aurora-code-examples"
authors = [
  "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
inquire = "0.6.2"
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für Rust.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen
  - CreateDBInstance
  - LöschenDBCluster

- LöschenDBClusterParameterGroup
- LöschenDBInstance
- Beschreiben DBCluster ParameterGroups
- Beschreiben Sie die DBCluster Parameter
- Beschreiben Sie DBCluster Schnappschüsse
- Beschreiben DBClusters
- DBEngineVersionen beschreiben
- Beschreiben DBInstances
- DescribeOrderableDBInstanceOptionen
- Modifizieren SieDBClusterParameterGroup

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktionen für Aurora mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie einzelne Aurora-Aktionen mit ausgeführt AWS SDKs werden. Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes finden.

Diese Auszüge rufen die Aurora-API auf und sind Codeauszüge aus größeren Programmen, die im Kontext ausgeführt werden müssen. Sie können Aktionen im Kontext unter Szenarien für die Verwendung von Aurora AWS SDKs anzeigen.

Die folgenden Beispiele enthalten nur die am häufigsten verwendeten Aktionen. Eine vollständige Liste finden Sie in der Amazon-Aurora-API-Referenz.

Beispiele

- Verwendung CreateDBCluster mit einem AWS SDK
- Verwendung CreateDBClusterParameterGroup mit einem AWS SDK
- Verwendung CreateDBClusterSnapshot mit einem AWS SDK
- Verwendung CreateDBInstance mit einem AWS SDK
- Verwendung DeleteDBCluster mit einem AWS SDK
- Verwendung DeleteDBClusterParameterGroup mit einem AWS SDK

- Verwendung DeleteDBInstance mit einem AWS SDK
- Verwendung DescribeDBClusterParameterGroups mit einem AWS SDK
- Verwendung DescribeDBClusterParameters mit einem AWS SDK
- Verwendung DescribeDBClusterSnapshots mit einem AWS SDK
- Verwendung DescribeDBClusters mit einem AWS SDK
- Verwendung DescribeDBEngineVersions mit einem AWS SDK
- Verwendung DescribeDBInstances mit einem AWS SDK
- Verwendung DescribeOrderableDBInstanceOptions mit einem AWS SDK oder CLI
- Verwendung ModifyDBClusterParameterGroup mit einem AWS SDK

Verwendung **`CreateDBCluster`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `CreateDBCluster` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
```

```
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- Einzelheiten zur API finden Sie unter DBCluster In der AWS SDK für .NET API-Referenz erstellen.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
```

- Einzelheiten zur API finden Sie unter DBCluster In der AWS SDK für C++ API-Referenz erstellen.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName
 string, parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
```

```
 }
}
```

- Einzelheiten zur API finden Sie unter DBCluster In der AWS SDK für Go API-Referenz erstellen.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
           String dbClusterIdentifier, String userName, String password) {
       try {
           CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                   .databaseName(dbName)
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .dbClusterParameterGroupName(dbParameterGroupFamily)
                   .engine("aurora-mysql")
                   .masterUsername(userName)
                   .masterUserPassword(password)
                   .build();

           CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
           return response.dbCluster().dbClusterArn();

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
       return "";
```

```
    }
```

- Einzelheiten zur API finden Sie unter DBCluster In der AWS SDK for Java 2.x API-Referenz erstellen.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createDBCluster(
    dbParameterGroupFamilyVal: String?,
    dbName: String?,
    dbClusterIdentifierVal: String?,
    userName: String?,
    password: String?,
): String? {
    val clusterRequest =
        CreateDbClusterRequest {
            databaseName = dbName
            dbClusterIdentifier = dbClusterIdentifierVal
            dbClusterParameterGroupName = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
            masterUsername = userName
            masterUserPassword = password
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}
```

- API-Details finden Sie unter API-Referenz DBCluster im AWS SDK für Kotlin erstellen.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
 group.
```

```
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster
```

- API-Einzelheiten finden Sie unter Create DBCluster in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
```

```
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
```

```
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));
```

```
            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
```

```
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
```

```
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
```

```
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
```

```
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie in der Referenz „Create DBCluster in AWS SDK for Rust API“.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`CreateDBClusterParameterGroup`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `CreateDBClusterParameterGroup` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
```

```
    public async Task<DBClusterParameterGroup>
CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
_amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- Einzelheiten zur API finden Sie unter DBCluster ParameterGroup In der AWS SDK für .NET API-Referenz erstellen.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
```

```
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
 created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- Einzelheiten zur API finden Sie unter DBCluster ParameterGroup In der AWS SDK für C++ API-Referenz erstellen.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- Einzelheiten zur API finden Sie unter DBCluster ParameterGroup In der AWS SDK für Go API-Referenz erstellen.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
           String dbParameterGroupFamily) {
       try {
           CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .dbParameterGroupFamily(dbParameterGroupFamily)
                   .description("Created by using the AWS SDK for Java")
                   .build();

           CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
           System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Einzelheiten zur API finden Sie unter DBCluster ParameterGroup In der AWS SDK for Java 2.x API-Referenz erstellen.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createDBClusterParameterGroup(
    dbClusterGroupNameVal: String?,
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupNameVal
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}
```

- API-Details finden Sie unter API-Referenz DBCluster ParameterGroup im AWS SDK für Kotlin erstellen.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
 parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
 group.
        :param parameter_group_family: The family that is used as the basis of
 the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API-Einzelheiten finden Sie unter Create DBCluster ParameterGroup in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
   pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
       self.engine_family = Some(engine.to_string());
       self.engine_version = Some(version.to_string());
       let create_db_cluster_parameter_group = self
           .rds
           .create_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
               engine,
           )
           .await;

       match create_db_cluster_parameter_group {
           Ok(CreateDbClusterParameterGroupOutput {
```

```
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
 do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
 SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

#[tokio::test]
```

```
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
```

- Einzelheiten zur API finden Sie in der Referenz „Create DBCluster ParameterGroup in AWS SDK for Rust API“.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`CreateDBClusterSnapshot`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `CreateDBClusterSnapshot` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- Einzelheiten zur API finden Sie unter DBClusterSnapshot erstellen in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- Einzelheiten zur API finden Sie unter DBClusterSnapshot erstellen in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context,
 clusterName string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
```

```
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBClusterSnapshot erstellen in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .build();

           CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
           System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Einzelheiten zur API finden Sie unter DBClusterSnapshot erstellen in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createDBClusterSnapshot(
    dbInstanceClusterIdentifier: String?,
    dbSnapshotIdentifier: String?,
) {
    val snapshotRequest =
        CreateDbClusterSnapshotRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}
```

- API-Details finden Sie unter DBClusterSnapshot im AWS SDK erstellen für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
 DBClusterIdentifier=cluster_id
            )
```

```
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- API-Einzelheiten finden Sie unter Create DBCluster Snapshot in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
```

```
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
           .db_cluster
           .and_then(|c| c.db_cluster_identifier);

       if self.db_cluster_identifier.is_none() {
           return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
       }

       info!(
           "Started a db cluster: {}",
           self.db_cluster_identifier
               .as_deref()
               .unwrap_or("Missing ARN")
       );
```

```
        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
```

```
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }
```

```
        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
 SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
```

```
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
```

```
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
```

```
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
```

```
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie in der API-Referenz „DBClusterSnapshot im AWS SDK für Rust erstellen“.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`CreateDBInstance`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `CreateDBInstance` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- Einzelheiten zur API finden Sie unter DBInstance In der AWS SDK für .NET API-Referenz erstellen.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
```

- Einzelheiten zur API finden Sie unter DBInstance In der AWS SDK für C++ API-Referenz erstellen.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context,
 clusterName string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
```

```
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBInstance In der AWS SDK für Go API-Referenz erstellen.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
```

- Einzelheiten zur API finden Sie unter DBInstance In der AWS SDK for Java 2.x API-Referenz erstellen.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createDBInstanceCluster(
    dbInstanceIdentifierVal: String?,
    dbInstanceClusterIdentifierVal: String?,
    instanceClassVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbClusterIdentifier = dbInstanceClusterIdentifierVal
            engine = "aurora-mysql"
            dbInstanceClass = instanceClassVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}
```

- API-Details finden Sie unter API-Referenz DBInstance im AWS SDK für Kotlin erstellen.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
 that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
 created.
```

```
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- API-Einzelheiten finden Sie unter Create DBInstance in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
```

```
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
```

```
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
```

```
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
```

```
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
```

```
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie in der Referenz „Create DBInstance in AWS SDK for Rust API“.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteDBCluster`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DeleteDBCluster` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
```

```
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
       try {
           DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .skipFinalSnapshot(true)
                   .build();

           rdsClient.deleteDBCluster(deleteDbClusterRequest);
           System.out.println(dbInstanceClusterIdentifier + " was deleted!");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest =
        DeleteDbClusterRequest {
            dbClusterIdentifier = dbInstanceClusterIdentifier
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

- API-Details finden Sie unter Löschen DBCluster im AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
```

```
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise
```

- API-Einzelheiten finden Sie unter Delete DBCluster in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
```

```
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
```

```
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
```

```
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
```

```
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
```

```
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
```

```
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster im AWS SDK für die Rust-API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteDBClusterParameterGroup`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DeleteDBClusterParameterGroup` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
```

```
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster ParameterGroup in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);
```

```
        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster ParameterGroup in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster ParameterGroup in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
```

```
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
                List<DBInstance> instanceList = response.dbInstances();
                int listSize = instanceList.size();
                didFind = false;
                int index = 1;
                for (DBInstance instance : instanceList) {
                    instanceARN = instance.dbInstanceArn();
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        System.out.println(clusterDBARN + " still exists");
                        didFind = true;
                    }
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster ParameterGroup in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(
    dbClusterGroupName: String,
    clusterDBARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
```

```
                }
            }
        }
        val clusterParameterGroupRequest =
            DeleteDbClusterParameterGroupRequest {
                dbClusterParameterGroupName = dbClusterGroupName
            }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}
```

- API-Details finden Sie unter Löschen DBCluster ParameterGroup im AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API-Einzelheiten finden Sie unter Delete DBCluster ParameterGroup in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
```

```
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
```

```
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
```

```
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
```

```
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
```

```
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
```

```
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster ParameterGroup im AWS SDK für die Rust-API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteDBInstance`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DeleteDBInstance` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <summary>
   /// Delete a particular DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <returns>DB instance object.</returns>
   public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
   {
       var response = await _amazonRDS.DeleteDBInstanceAsync(
           new DeleteDBInstanceRequest()
           {
```

```
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
```

```
                        << "Waiting for DB instance to delete before deleting the
 parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx,
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
       try {
           DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .deleteAutomatedBackups(true)
                   .skipFinalSnapshot(true)
```

```
                    .build();

            DeleteDbInstanceResponse response =
 rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
 response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}
```

- API-Details finden Sie unter Löschen DBInstance im AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
```

```
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- API-Einzelheiten finden Sie unter Delete DBInstance in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
```

```
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
```

```
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
```

```
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
```

```
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance im AWS SDK für die Rust-API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeDBClusterParameterGroups`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DescribeDBClusterParameterGroups` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- Einzelheiten zur API finden Sie unter Describe DBCluster ParameterGroups in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }

        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- Einzelheiten zur API finden Sie unter Describe DBCluster ParameterGroups in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
```

```
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBCluster ParameterGroups in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
               System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
           }
```

```
        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter Describe DBCluster ParameterGroups in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest =
        DescribeDbClusterParameterGroupsRequest {
            dbClusterParameterGroupName = dbClusterGroupName
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

- Einzelheiten zur API finden Sie unter DBClusterParameterGroupsDescribe in AWS SDK for Kotlin API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group
```

- Einzelheiten zur API finden Sie unter DBClusterParameterGroupsDescribe in AWS SDK for Python (Boto3) API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. Verwenden Sie diesen Dienst mit einem AWS SDK Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeDBClusterParameters`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DescribeDBClusterParameters` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- Einzelheiten zur API finden Sie unter Describe DBCluster Parameters in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBClusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBClusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
```

```
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- Einzelheiten zur API finden Sie unter Describe DBCluster Parameters in der AWS SDK für C ++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
```

```
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- Einzelheiten zur API finden Sie unter Describe DBCluster Parameters in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
```

```
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter Describe DBCluster Parameters in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
```

```
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- API-Details finden Sie unter Describe DBCluster Parameters in AWS SDK for Kotlin API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters
```

- Einzelheiten zur API finden Sie unter DBClusterDescribe Parameters in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Get the parameter group. rds.DescribeDbClusterParameterGroups
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
           ));
       }

       let parameters = parameters_output
           .unwrap()
           .into_iter()
           .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
           .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
           .map(AuroraScenarioParameter::from)
```

```
            .collect::<Vec<_>>();

        Ok(parameters)
    }

    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
 SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
```

- Einzelheiten zur API finden Sie unter DBClusterDescribe Parameters in AWS SDK for Rust API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeDBClusterSnapshots`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DescribeDBClusterSnapshots` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <summary>
   /// Return a list of DB snapshots for a particular DB cluster.
   /// </summary>
   /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
   /// <returns>List of DB snapshots.</returns>
   public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
   {
       var results = new List<DBClusterSnapshot>();

       DescribeDBClusterSnapshotsResponse response;
       DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
       {
           DBClusterIdentifier = dbClusterIdentifier
       };
       // Get the full list if there are multiple pages.
       do
```

```
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Einzelheiten zur API finden Sie unter Beschreiben von DBCluster Snapshots in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
```

```
                        << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }
```

- Einzelheiten zur API finden Sie unter Beschreiben von DBCluster Snapshots in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
```

```
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- Einzelheiten zur API finden Sie unter Beschreiben von DBCluster Snapshots in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .build();
```

```
            while (!snapshotReady) {
                DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
                List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
                for (DBClusterSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter Beschreiben von DBCluster Snapshots in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun waitSnapshotReady(
    dbSnapshotIdentifier: String?,
    dbInstanceClusterIdentifier: String?,
```

```
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbClusterSnapshotsRequest {
            dbClusterSnapshotIdentifier = dbSnapshotIdentifier
            dbClusterIdentifier = dbInstanceClusterIdentifier
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

- Einzelheiten zur API finden Sie unter Beschreiben von DBCluster Snapshots im AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
```

```
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- Einzelheiten zur API finden Sie unter Describe DBCluster Snapshots in AWS SDK for Python (Boto3) API Reference.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. Verwenden Sie diesen Dienst mit einem AWS SDK Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeDBClusters`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DescribeDBClusters` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
```

```
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
```

```
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
```

```
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun describeDbClusterParameters(
    dbCLusterGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
            }
        } else {
            DescribeDbClusterParametersRequest {
                dbClusterParameterGroupName = dbCLusterGroupName
                source = "user"
            }
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
```

```
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- Einzelheiten zur API finden Sie unter DBClustersDescribe in AWS SDK for Kotlin API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster
```

- Einzelheiten zur API finden Sie unter DBClustersDescribe in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
```

```
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
```

```
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
```

```
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie unter DBClustersDescribe in AWS SDK for Rust API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeDBEngineVersions`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DescribeDBEngineVersions` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
```

```
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void describeDBEngines(RdsClient rdsClient) {
       try {
           DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                   .engine("aurora-mysql")
                   .defaultOnly(true)
                   .maxRecords(20)
                   .build();

           DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
           List<DBEngineVersion> engines = response.dbEngineVersions();

           // Get all DBEngineVersion objects.
           for (DBEngineVersion engineOb : engines) {
               System.out.println("The name of the DB parameter group family for
the database engine is "
                       + engineOb.dbParameterGroupFamily());
               System.out.println("The name of the database engine " +
engineOb.engine());
               System.out.println("The version number of the database engine " +
engineOb.engineVersion());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "aurora-mysql"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}
```

- API-Details finden Sie unter DBEngineVersionen beschreiben im AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
 list of
                                       engine versions to those that are
 compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
```

```
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- Einzelheiten zur API finden Sie unter DBEngineDescribe Versions in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");
```

```
        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }
```

```
#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}
```

```
#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der Referenz zum AWS SDK für die Rust-API.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeDBInstances`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `DescribeDBInstances` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
```

```
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    // Waits until the database instance is available.
```

```
   public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
       boolean instanceReady = false;
       String instanceReadyStr;
       System.out.println("Waiting for instance to become available.");
       try {
           DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .build();

           while (!instanceReady) {
               DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
               List<DBCluster> clusterList = response.dbClusters();
               for (DBCluster cluster : clusterList) {
                   instanceReadyStr = cluster.status();
                   if (instanceReadyStr.contains("available")) {
                       instanceReady = true;
                   } else {
                       System.out.print(".");
                       Thread.sleep(sleepTime * 1000);
                   }
               }
           }
           System.out.println("Database cluster is available!");

       } catch (RdsException | InterruptedException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    var endpoint = ""
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

- Einzelheiten zur API finden Sie unter DBInstancesDescribe in AWS SDK for Kotlin API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst
```

- Einzelheiten zur API finden Sie unter DBInstancesDescribe in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
```

```
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
```

```
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
```

```
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
```

```
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
```

```
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Cluster
    tokio::time::resume();
```

```
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));
```

```
    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
```

```
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.first(), Some(ScenarioError {message, context: _})
 if message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Einzelheiten zur API finden Sie unter DBInstancesDescribe in AWS SDK for Rust API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeOrderableDBInstanceOptions`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DescribeOrderableDBInstanceOptions` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
```

```
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
```

```
 })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();
```

```
            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS SDK for Java 2.x API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Dieses Beispiel listet die DB-Engine-Versionen auf, die eine bestimmte DB-Instance-Klasse in einer AWS-Region unterstützen.

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

Beispiel 2: Dieses Beispiel listet die DB-Instance-Klassen auf, die von einer bestimmten DB-Engine-Version in einer AWS-Region unterstützt werden.

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
```

```
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS - Tools für PowerShell Cmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Dieses Beispiel listet die DB-Engine-Versionen auf, die eine bestimmte DB-Instance-Klasse in einer AWS-Region unterstützen.

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

Beispiel 2: Dieses Beispiel listet die DB-Instance-Klassen auf, die von einer bestimmten DB-Engine-Version in einer AWS-Region unterstützt werden.

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS - Tools für PowerShell Cmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
 instance.
        :param db_engine_version: The engine version that must be supported by
 the DB instance.
        :return: The list of DB instance options that can be used to create a
 compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS API-Referenz zum SDK for Python (Boto3).

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .filter(|o| o.storage_type() == Some("aurora"))
```

```
                    .map(|o|
 o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
 classes", &err))
    }

    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
 SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
```

```
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .storage_type("aurora-iopt1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .storage_type("aurora")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .storage_type("aurora")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
```

```
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
```

- Einzelheiten zur API finden Sie in der Referenz zu den DescribeOrderableDBInstanceOptionen im AWS SDK für die Rust-API.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ModifyDBClusterParameterGroup`** mit einem AWS SDK

Die folgenden Code-Beispiele zeigen, wie `ModifyDBClusterParameterGroup` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
```

```
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- Einzelheiten zur API finden Sie unter Ändern DBCluster ParameterGroup in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
```

```
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Einzelheiten zur API finden Sie unter Ändern DBCluster ParameterGroup in der AWS SDK für C++ API-Referenz.

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
```

```
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Ändern DBCluster ParameterGroup in der AWS SDK für Go API-Referenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
```

```
                System.out.println("The group ARN is " +
 group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Einzelheiten zur API finden Sie unter Ändern DBCluster ParameterGroup in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.fromValue("immediate")
            parameterValue = "5"
        }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dClusterGroupName
            parameters = paraList
        }
```

```
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}
```

- API-Details finden Sie unter Modify DBCluster ParameterGroup in AWS SDK for Kotlin API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
```

```
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API-Einzelheiten finden Sie unter Modify DBCluster ParameterGroup in AWS SDK for Python (Boto3) API-Referenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
```

```
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
```

```
            .set_parameters(Some(parameters))
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
```

- Einzelheiten zur API finden Sie unter API-Referenz zum Ändern DBCluster ParameterGroup im AWS SDK für Rust.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Szenarien für die Verwendung von Aurora AWS SDKs

Die folgenden Codebeispiele zeigen Ihnen, wie Sie gängige Szenarien in Aurora mit implementieren AWS SDKs. Diese Szenarien zeigen Ihnen, wie Sie bestimmte Aufgaben durch den Aufruf mehrerer Funktionen innerhalb von Aurora oder in Kombination mit anderen AWS-Services ausführen können. Jedes Szenario enthält einen Link zum vollständigen Quell-Code, wo Sie Anleitungen zum Einrichten und Ausführen des Codes finden.

Szenarien zielen auf eine mittlere Erfahrungsebene ab, um Ihnen zu helfen, Service-Aktionen im Kontext zu verstehen.

Beispiele

- Leihbibliothek-REST-API erstellen
- Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

Leihbibliothek-REST-API erstellen

Im folgenden Codebeispiel wird veranschaulicht, wie man eine Leihbibliothek erstellt, in der Kunden Bücher mithilfe einer REST-API ausleihen und zurückgeben können, die von einer Amazon-Aurora-Datenbank unterstützt wird.

Python

SDK für Python (Boto3)

Zeigt, wie die AWS SDK for Python (Boto3) mit der Amazon Relational Database Service (Amazon RDS) API und AWS Chalice verwendet wird, um eine REST-API zu erstellen, die von einer Amazon Aurora Aurora-Datenbank unterstützt wird. Der Webservice ist vollständig Serverless und stellt eine einfache Leihbibliothek dar, in der die Kunden Bücher ausleihen und zurückgeben können. Lernen Sie Folgendes:

- Erstellen und verwalten Sie einen Serverless-Aurora-Datenbank-Cluster.
- Wird AWS Secrets Manager zur Verwaltung von Datenbankanmeldedaten verwendet.
- Implementieren Sie einen Datenspeicher-Layer, der Amazon RDS verwendet, um Daten in die und aus der Datenbank zu verschieben.
- Verwenden Sie AWS Chalice, um eine serverlose REST-API für Amazon API Gateway bereitzustellen und. AWS Lambda
- Verwenden Sie das Anforderungspaket, um Anfragen an den Webservice zu senden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. GitHub

In diesem Beispiel verwendete Dienste

- API Gateway
- Aurora
- Lambda
- Secrets Manager

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

Die folgenden Code-Beispiele zeigen, wie eine Webanwendung erstellt wird, die Arbeitselemente in einer Amazon Aurora Serverless-Datenbank verfolgt und mithilfe von Amazon Simple Email Service (Amazon SES) Berichte sendet.

.NET

SDK für .NET

Zeigt, wie Sie mithilfe von Amazon Simple Email Service (Amazon SES) eine Webanwendung erstellen, die Arbeitsaufgaben in einer Amazon Aurora Aurora-Datenbank nachverfolgt und Berichte per E-Mail versendet. AWS SDK für .NET In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit RESTful einem.NET-Backend zu interagieren.

- Integrieren Sie eine React-Webanwendung in AWS Dienste.
- Auflisten, Hinzufügen, Aktualisieren und Löschen von Elementen in einer Aurora-Tabelle.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.
- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

C++

SDK für C++

Zeigt, wie eine Webanwendung erstellt wird, die in einer Datenbank von Amazon Aurora Serverless gespeicherte Arbeitselemente verfolgt und darüber berichtet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung einer C++-REST-API, die Amazon Aurora Aurora-Serverless-Daten abfragt und von einer React-Anwendung verwendet werden kann, finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Java

SDK für Java 2.x

Zeigt, wie eine Webanwendung erstellt wird, die Arbeitselemente, die in einer Amazon RDS-Datenbank gespeichert sind, verfolgt und darüber berichtet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung einer Spring REST-API, die Amazon Aurora Aurora-Serverless-Daten abfragt und von einer React-Anwendung verwendet werden kann, finden Sie im vollständigen Beispiel unter GitHub.

Den vollständigen Quellcode und Anweisungen zum Einrichten und Ausführen eines Beispiels, das die JDBC-API verwendet, finden Sie im vollständigen Beispiel unter. GitHub

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

JavaScript

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript (v3) eine Webanwendung erstellen, die Arbeitsaufgaben in einer Amazon Aurora Aurora-Datenbank verfolgt und Berichte mithilfe von Amazon Simple Email Service (Amazon SES) per E-Mail versendet. In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem Express-Node.js-Backend zu interagieren.

- Integrieren Sie eine React.js Webanwendung mit AWS-Services.
- Auflisten, hinzufügen und aktualisieren von Elementen in einer Aurora-Tabelle.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.
- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Kotlin

SDK für Kotlin

Zeigt, wie eine Webanwendung erstellt wird, die Arbeitselemente, die in einer Amazon RDS-Datenbank gespeichert sind, verfolgt und darüber berichtet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung einer Spring REST-API, die Amazon Aurora Aurora-Serverless-Daten abfragt und von einer React-Anwendung verwendet werden kann, finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- Aurora

- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

PHP

SDK für PHP

Zeigt, wie Sie mithilfe von Amazon Simple Email Service (Amazon SES) eine Webanwendung erstellen, die Arbeitselemente in einer Amazon RDS-Datenbank verfolgt und Berichte per E-Mail versendet. AWS SDK für PHP In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem RESTful PHP-Backend zu interagieren.

- Integrieren Sie eine React.js -Webanwendung in AWS Dienste.
- In einer Amazon-RDS-Tabelle können Sie Elemente auflisten, aktualisieren und löschen.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.
- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Python

SDK für Python (Boto3)

Zeigt, wie Sie mithilfe von Amazon Simple Email Service (Amazon SES) einen REST-Service erstellen, der Arbeitselemente in einer Amazon Aurora Aurora-Serverless-Datenbank nachverfolgt und Berichte per E-Mail versendet. AWS SDK for Python (Boto3) In diesem Beispiel wird das Flask-Web-Framework für das HTTP-Routing verwendet und in eine React-Webseite integriert, um eine voll funktionsfähige Webanwendung zu präsentieren.

- Erstellen Sie einen Flask-REST-Service, der sich in integrieren lässt. AWS-Services
- Lesen, schreiben und aktualisieren Sie Arbeitsaufgaben, die in einer Aurora-Serverless-Datenbank gespeichert sind.
- Erstellen Sie ein AWS Secrets Manager Geheimnis, das Datenbankanmeldedaten enthält, und verwenden Sie es, um Aufrufe an die Datenbank zu authentifizieren.
- Verwenden Sie Amazon SES, um E-Mail-Berichte über Arbeitsaufgaben zu senden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter GitHub.

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Bewährte Methoden mit Amazon Aurora

Nachstehend finden Sie Informationen zu allgemeinen bewährten Methoden und zu Optionen für die Verwendung oder Migration von Daten zu einem Amazon Aurora-DB-Cluster.

Einige der bewährten Methoden für Amazon Aurora sind spezifisch für eine bestimmte Datenbank-Engine. Weitere Informationen zu den bewährten Methoden in Aurora für eine spezifische Datenbank-Engine finden Sie im Folgenden.

Datenbank-Engine	Bewährte Methoden
Amazon Aurora MySQL	Siehe Bewährte Methoden mit Amazon Aurora MySQL
Amazon Aurora PostgreSQL	Siehe Bewährte Methoden mit Amazon Aurora PostgreSQL

> **Note**
>
> Allgemeine Empfehlungen für Aurora finden Sie unter Empfehlungen von Amazon Aurora.

Themen

- Grundlegende Anleitungen für Amazon Aurora
- RAM-Empfehlungen für DB-Instances
- AWS-Datenbanktreiber
- Überwachen von Amazon Aurora
- Arbeiten mit DB-Parametergruppen und DB-Cluster-Parametergruppen
- Video zu bewährten Amazon-Aurora-Methoden

Grundlegende Anleitungen für Amazon Aurora

Im Folgenden finden Sie einige grundlegende Anleitungen für die Ausführung, die bei der Arbeit mit Amazon Aurora befolgt werden sollten. Das Amazon RDS Service Level Agreement setzt voraussetzt, dass Sie diese Anleitungen befolgen:

- Sie müssen die Nutzung von Arbeitsspeicher, CPU und Speicher überwachen. Sie können Amazon CloudWatch so einrichten, dass Sie benachrichtigt werden, wenn sich Nutzungsmuster ändern oder wenn die Kapazität Ihrer Bereitstellung fast erreicht ist. Auf diese Weise können Sie leichter die Leistung und Verfügbarkeit des Systems wahren.
- Wenn Ihre Client-Anwendung die Domain Name Service (DNS)-Daten Ihrer DB-Instances zwischenspeichert, sollten Sie einen Time-to-Live (TTL)-Wert von weniger als 30 Sekunden festlegen. Die zugrunde liegende IP-Adresse einer DB-Instance kann sich nach einem Failover ändern. Daher kann das Zwischenspeichern der DNS-Daten über einen längeren Zeitraum zu Verbindungsfehlern führen, wenn Ihre Anwendung versucht, eine Verbindung mit einer IP-Adresse herzustellen, die nicht mehr verwendet wird. Bei Aurora-DB-Clustern mit mehreren Lesereplikaten kann es auch zu Verbindungsfehlern kommen, wenn Verbindungen den Leser-Endpunkt verwenden und eine der Lesereplikat-Instances gewartet wird oder gelöscht wurde.
- Testen Sie den Failover für Ihren DB-Cluster, um zu verstehen, wie lange der Vorgang für Ihren Anwendungsfall dauert. Durch Testen des Failovers können Sie sicherstellen, dass die Anwendung, die auf Ihren DB-Cluster zugreift, nach dem Failover automatisch eine Verbindung zum neuen DB-Cluster herstellen kann.

RAM-Empfehlungen für DB-Instances

Um die Leistung zu optimieren, sollten Sie ausreichend RAM zuteilen, damit sich Ihr Arbeitssatz beinahe vollständig im Arbeitsspeicher befindet. Um zu ermitteln, ob sich Ihr Arbeitssatz beinahe vollständig im Arbeitsspeicher befindet, überprüfen Sie die folgenden Metriken in Amazon CloudWatch:

- `VolumeReadIOPS`: Diese Metrik misst die durchschnittliche Anzahl von I/O-Lesevorgängen aus einem Cluster-Volume, die in 5-minütigen Intervallen gemeldet werden. Der Wert für `VolumeReadIOPS` sollte klein und stabil sein. In einigen Fällen stellen Sie möglicherweise fest, dass Ihre Lese-I/O steil ansteigt oder höher ist als üblich. Untersuchen Sie in diesem Fall die DB-Instances in Ihrem DB-Cluster, um festzustellen, welche DB-Instances die erhöhte I/O verursachen.

> **Tip**
>
> Wenn Ihr Aurora MySQL Cluster eine parallele Abfrage verwendet, wird möglicherweise eine Zunahme der `VolumeReadIOPS`-Werte. Parallel Query verwendet den Bufferpool nicht. Obwohl die Abfragen schnell sind, kann diese optimierte Verarbeitung zu einem Anstieg der Lesevorgänge und damit verbundenen Gebühren führen.

- `BufferCacheHitRatio`: Diese Metrik misst den Prozentsatz der Anfragen, die vom Puffer-Cache einer DB-Instance in Ihrem DB-Cluster verarbeitet werden. Diese Metrik stellt Ihnen Einsichten in die Menge der Daten bereit, die vom Arbeitsspeicher bereitgestellt werden.

  Eine hohe Trefferrate bedeutet, dass Ihre DB-Instance über genügend Speicher verfügt. Wenn die Trefferrate niedrig ist, zeigt dies an, dass Ihre Abfragen für diese DB-Instance häufig zum Datenträger geleitet werden. Untersuchen Sie den Workload darauf, welche Abfragen dieses Verhalten verursachen.

Wenn Sie aufgrund der Untersuchung des Workloads feststellen, dass Sie mehr Arbeitsspeicher benötigen, sollten Sie erwägen, die DB-Instance-Klasse auf eine Klasse mit mehr RAM zu skalieren. Anschließend können Sie die voranstehend beschriebene Metrik untersuchen und weiter skalieren wie erforderlich. Wenn Ihr Aurora-Cluster größer als 40 TB ist, verwenden Sie nicht die Instance-Klassen db.t2, db.t3 oder db.t4g.

Weitere Informationen finden Sie unter CloudWatch Amazon-Metriken für Amazon Aurora.

AWS-Datenbanktreiber

Wir empfehlen die AWS-Treibersuite für die Anwendungskonnektivität. Die Treiber wurden so konzipiert, dass sie schnellere Umstellungs- und Failover-Zeiten sowie die Authentifizierung mit AWS Secrets Manager, AWS Identity and Access Management (IAM) und Verbundidentität unterstützen. Die AWS-Treiber müssen den Status des DB-Clusters überwachen und die Clustertopologie kennen, um den neuen Writer zu ermitteln. Dieser Ansatz reduziert die Umstellungs- und Failover-Zeiten auf Werte im einstelligen Sekundenbereich, im Vergleich zu Werten im zweistelligen Bereich bei Open-Source-Treibern.

Im Zuge der Einführung neuer Servicefunktionen besteht das Ziel der AWS-Treibersuite darin, eine integrierte Unterstützung für diese Servicefunktionen zu bieten.

Weitere Informationen finden Sie unter Herstellen einer Verbindung mit Aurora-DB-Clustern mithilfe der AWS-Treiber.

Überwachen von Amazon Aurora

Amazon Aurora stellt verschiedene Metriken und Erkenntnisse bereit, die Sie überwachen können, um den Zustand und die Leistung Ihres Aurora-DB-Clusters zu ermitteln. Sie können verschiedene Tools, wie die AWS-Managementkonsole, AWS CLI und CloudWatch-API verwenden, um Ihre

Aurora-Metriken anzuzeigen. Sie können die kombinierten Performance-Insights- und CloudWatch-Metriken im Performance-Insights-Dashboard einsehen und Ihre DB-Instance überwachen. Performance Insights muss für Ihre DB-Instance aktiviert sein, damit diese Ansicht verwendet werden kann. Weitere Informationen zu dieser Überwachungsansicht finden Sie unter Anzeigen kombinierter Metriken mit dem Performance-Insights-Dashboard.

Sie können einen Leistungsanalysebericht für einen bestimmten Zeitraum erstellen und sich die ermittelten Erkenntnisse und Empfehlungen zur Lösung der Probleme ansehen. Weitere Informationen finden Sie unter Erstellen eines Leistungsanalyseberichts in Performance Insights.

Arbeiten mit DB-Parametergruppen und DB-Cluster-Parametergruppen

Es wird empfohlen, Änderungen für DB-Parametergruppen und DB-Cluster-Parametergruppen in einem DB-Test-Cluster zu testen, bevor Sie Änderungen für Parametergruppen auf Ihr DB-Produktions-Cluster anwenden. Wenn die Parameter für eine DB-Engine falsch festgelegt werden, kann dies unbeabsichtigte nachteilige Auswirkungen haben, einschließlich einer reduzierten Leistung und Systeminstabilität.

Gehen Sie stets vorsichtig vor, wenn Sie DB-Engine-Parameter ändern, und sichern Sie Ihren DB-Cluster, bevor Sie eine DB-Parametergruppe ändern. Informationen zum Sichern eines DB-Clusters finden Sie unter Sichern und Wiederherstellen eines Amazon-Aurora-DB-Clusters.

Video zu bewährten Amazon-Aurora-Methoden

Der AWS Online Tech Talks Channel auf YouTube umfasst eine Videopräsentation zu bewährten Methoden für die Erstellung und Konfiguration eines noch sicheren und hochverfügbaren DB-Clusters von Amazon Aurora. Siehe Bewährte Methoden für die Hochverfügbarkeit von Amazon Aurora.

Durchführen eines Machbarkeitsnachweises mit Amazon Aurora

Im Folgenden wird beschrieben, wie ein Machbarkeitsnachweises für Aurora eingerichtet und ausgeführt wird. Ein Machbarkeitsnachweis ist eine Untersuchung, die Sie selbst durchführen, um herauszufinden, ob Aurora für Ihre Anwendung geeignet ist. Mithilfe des Machbarkeitsnachweises können Sie die Aurora-Funktionen leichter im Rahmen Ihrer eigenen Datenbankanwendungen verstehen und Aurora mit Ihrer aktuellen Datenbankumgebung vergleichen. Er kann auch aufzeigen, wie groß der erforderliche Aufwand für das Verschieben von Daten, Portieren von SQL-Code, Leistungsoptimierung und Anpassen Ihrer aktuellen Verwaltungsverfahren ist.

In diesem Thema finden Sie eine Übersicht und schrittweise Skizzierung der wichtigsten Verfahren und Entscheidungen im Zusammenhang mit der Ausführung eines Machbarkeitsnachweises, wie im Folgenden aufgelistet. Eine ausführliche Anleitung finden Sie unter den Links zur vollständigen Dokumentation für bestimmte Themen.

Übersicht über einen Aurora-Machbarkeitsnachweis

Wenn Sie einen Machbarkeitsnachweis für Amazon Aurora durchführen, lernen Sie, was mit dem Portieren Ihrer vorhanden Daten und SQL-Anwendungen zu Aurora verbunden ist. Sie üben die wichtigsten Aspekte von Aurora im großen Umfang unter Verwendung eines Datenvolumens und von Aktivitäten, die für Ihre Produktionsumgebung repräsentativ sind. Ziel ist es, sich davon zu überzeugen, dass die Vorteile von Aurora den Herausforderungen gewachsen sind, denen Ihre vorherige Datenbankinfrastruktur nicht mehr gerecht wurde. Am Ende des Machbarkeitsnachweises haben Sie einen gründlichen Plan für die Durchführung von Benchmark-Tests und Anwendungstests im größeren Maßstab. An dieser Stelle haben Sie eine Vorstellung von den wichtigsten Arbeitsschritten auf Ihrem Weg zu einer Produktionsbereitstellung.

Mit den folgenden Ratschlägen zu bewährten Methoden lassen sich geläufige Fehler vermeiden, die beim Durchführen von Benchmark-Tests Probleme bereiten können. Dieses Thema behandelt jedoch keine schrittweise Anleitung für Benchmark-Tests und Leistungsoptimierung. Diese Verfahren hängen von Ihrer Workload ab und welche Aurora-Funktionen Sie benutzen. Detaillierte Informationen können Sie der leistungsbezogenen Dokumentation entnehmen, z. B. Verwalten von Performance und Skalierung für einen Aurora-DB-Cluster, Amazon Aurora MySQL-Leistungserweiterungen, Leistung und Skalierung für Amazon Aurora PostgreSQL und Überwachung mit Performance Insights auf .

Die Informationen in diesem Thema gelten hauptsächlich für Anwendungen, bei denen Ihre Organisation den Code verfasst und das Schema entwirft und von denen die MySQL- und PostgreSQL-Open-Source-Datenbank-Engines unterstützt werden. Wenn Sie eine kommerzielle Anwendung testen oder eine Anwendung, die von einem Anwendungs-Framework generiert wurde, haben Sie möglicherweise nicht die Flexibilität, alle Richtlinien anzuwenden. Erkundigen Sie sich in solchen Fällen bei Ihrem AWS-Mitarbeiter, ob für Ihren Typ von Anwendung bewährte Methode für Aurora oder Fallbeispiele vorhanden sind.

1. Identifizieren Ihrer Ziele

Wenn Sie Aurora als Teil des Machbarkeitsnachweises auswerten, wählen Sie die vorzunehmenden Messungen aus und wie der Erfolg der Übung zu bewerten ist.

Sie müssen sicherstellen, dass die gesamte Funktionalität Ihre Anwendung mit Aurora kompatibel ist. Da die Aurora-Hauptversionen wire-kompatibel mit entsprechenden Hauptversionen von MySQL und PostgreSQL sind, sind die meisten für diese Engines entwickelten Anwendungen ebenfalls mit Aurora kompatibel. Sie müssen die Kompatibilität jedoch weiterhin je nach Anwendung validieren.

So wirken sich beispielsweise einige der Konfigurationsoptionen, die Sie beim Einrichten eines Aurora-Clusters ausgewählt haben, darauf aus, ob Sie bestimmte Datenbankfunktionen verwenden können oder sollten. Sie können mit einem Aurora-Allzweck-Cluster beginnen der als bereitgestellt bezeichnet wird. Danach können Sie entscheiden, ob eine spezialisierte Konfiguration wie z. B. Serverless oder Parallelabfrage Vorteile für Ihre Workload bietet.

Anhand der folgenden Fragen können Sie Ihre Ziele leichter identifizieren und quantifizieren:

- Unterstützt Aurora alle funktionalen Anwendungsfälle Ihrer Workload?
- An welcher Datensatzgröße und Laststufe sind Sie interessiert? Können Sie auf diese Stufe skalieren?
- Was sind Ihre spezifischen Anforderungen für Abfragedurchsatz und Latenz? Können Sie sie erreichen?
- Was ist die akzeptable Mindestmenge an geplanten oder nicht geplanten Ausfallzeiten für Ihre Workload? Können Sie sie erzielen?
- Was sind die erforderlichen Metriken für Betriebseffizienz? Können Sie sie genau überwachen?
- Unterstützt Aurora Ihre spezifischen geschäftlichen Ziele, z. B. Kostenreduzierung, Zunahme der Bereitstellung oder Bereitstellungsgeschwindigkeit? Haben Sie eine Methode zur Quantifizierung dieser Ziele?

- Können Sie alle Sicherheits- und Compliance-Anforderung für Ihre Workload erfüllen?

Nehmen Sie sich etwas Zeit, sich mit Aurora-Datenbank-Engines und -Plattformfunktionen weiter vertraut zu machen, und sehen Sie Servicedokumentation ein. Notieren Sie sich alle Funktionen, die Ihnen helfen können, die von Ihnen angestrebten Ergebnisse zu erzielen. Eine dieser Funktionen könnte die Workload-Konsolidierung sein, die im AWS Database Blog-Beitrag How to plan and optimize Amazon Aurora with MySQL compatibility for consolidated workloads beschrieben wird. Eine andere könnte bedarfsorientierte Skalierung sein, die unter Amazon-Aurora-Auto-Scaling mit Aurora-Replikaten im Amazon Aurora Benutzerhandbuch beschrieben wird. Weitere mögliche Funktionen sind Steigerung der Performance und vereinfachte Datenbankoperationen.

2. Verständnis der Workload-Eigenschaften

Bewerten Sie Aurora im Kontext Ihres beabsichtigten Anwendungsfalls. Aurora ist eine gute Wahl für Online-Transaktionsverarbeitungs-Workloads (OLTP). Sie können auch Berichte für den Cluster mit den Echtzeit-OLTP-Daten ausführen, ohne einen separaten Data Warehouse-Cluster bereitzustellen. Sie können erkennen, ob Ihr Anwendungsfall in diese Kategorien fällt, indem Sie die folgenden Merkmale überprüfen:

- Hohe Parallelität mit Dutzenden, Hunderten oder Tausenden gleichzeitigen Clients.
- Großes Volumen von schneller Abfragen (Millisekunden bis Sekunden).
- Kurze Echtzeit-Transaktionen.
- Hoch selektive Abfragemuster mit indexbasierten Suchvorgängen.
- Für HTAP können analytische Abfragen die Vorteile der Parallelabfrage von Aurora nutzen.

Einer der Schlüsselfaktoren, der bei der Auswahl Ihrer Datenbankoptionen eine Rolle spielt, ist die Geschwindigkeit der Daten. Hohe Geschwindigkeit bedeutet, dass Daten sehr häufig eingefügt und aktualisiert werden. Ein solches System kann Tausende von Verbindungen und Hunderttausende gleichzeitiger Abfragen haben, die gleichzeitig aus einer Datenbank gelesen und in sie geschrieben werden. Abfragen bei Systemen hoher Geschwindigkeit betreffen gewöhnlich eine relativ kleine Anzahl von Zeilen und greifen in der Regel auf mehrere Spalten in derselben Zeile zu.

Aurora ist auf die Verarbeitung von Daten hoher Geschwindigkeit ausgelegt. Abhängig von der Workload kann ein Aurora-Cluster mit einer einzigen r4.16xlarge-DB-Instance mehr als 600 000 `SELECT`-Anweisungen pro Sekunden verarbeiten. Auch hier kann ein solcher Cluster je nach Workload 200 000 `INSERT`, `UPDATE` und `DELETE`-Anweisungen pro Sekunde verarbeiten. Aurora

ist eine Zeilenspeicher-Datenbank und eignet sich ideal für hochvolumige, hochdurchsatzstarke und hochgradig parallelisierte OLTP-Workloads.

Aurora kann außerdem Berichtsabfragen in demselben Cluster durchführen, der die OLTP-Workload verarbeitet. Aurora unterstützt bis zu 15 Replikate, die jeweils innerhalb von 10-20 Millisekunden der primären Instance liegen. Analysten können OLTP-Daten in Echtzeit abfragen, ohne die Daten in einen separaten Data Warehouse-Cluster kopieren zu müssen. Mit Aurora-Clustern, die die Parallelabfragefunktion nutzen, können Sie viele der Verarbeitungs-, Filterungs- und Aggregationsvorgänge in das stark verteilte Aurora-Speichersubsystem verlagern.

Verwenden Sie diese Planungsphase, um sich mit den Funktionen von Aurora, anderen AWS-Managementkonsole-Services, der AWS und der AWS CLI vertraut zu machen. Finden Sie auch heraus, wie diese mit anderen Tools funktionieren, die Sie bei dem Machbarkeitsnachweis verwenden möchten.

3. Durchführen einer Übung mit der AWS-Managementkonsole oder der AWS CLI

Führen Sie als nächster Schritt eine Übung mit der AWS-Managementkonsole oder der AWS CLI durch, um sich mit diesen Tools und mit Aurora vertraut zu machen.

Durchführen einer Übung mit der AWS-Managementkonsole

Die folgenden anfänglichen Aktivitäten mit Aurora-Datenbank-Clustern sollen Ihnen in erster Linie Gelegenheit geben, sich mit der AWS-Managementkonsole-Umgebung vertraut zu machen und sich im Einrichten und Ändern von Aurora-Clustern zu üben. Wenn Sie die MySQL-kompatiblen und PostgreSQL-kompatiblen Datenbank-Engines mit Amazon RDS verwenden, kommen Ihnen diese Kenntnisse bei der Verwendung von Aurora zugute.

Durch Nutzung des freigegebenen Speichermodells von Aurora und von Funktionen wie Replikation und Snapshots können Sie gesamte Datenbank-Cluster wie andere Arten von Objekten verwenden, die Sie ganz nach Wunsch bearbeiten. Sie können Aurora-Cluster während des Machbarkeitsnachweises einrichten, außer Betrieb nehmen und ihre Kapazität häufig ändern. Sie sind nicht an frühzeitige Entscheidungen bezüglich Kapazität, Datenbankeinstellungen und physisches Daten-Layout gebunden.

Richten Sie zum Einsteig einen leeren Aurora-Cluster ein. Wählen Sie als Kapazitätstyp provisioned (bereitgestellt) und unter regional den Standort für Ihre anfänglichen Experimente aus.

Stellen Sie mit einem Client-Programm, wie z. B. einer SQL-Befehlszeilen-Anwendung, eine Verbindung mit diesem Cluster her. Anfänglich stellen Sie eine Verbindung über den Cluster-Endpunkt her. Sie stellen eine Verbindung mit diesem Endpunkt her, um Schreibvorgänge wie Data Definition Language (DDL)-Anweisungen und Extract, Transform, Load (ETL)-Prozesse durchzuführen. An späterer Stelle im Machbarkeitsnachweis verbinden Sie abfrageintensive Sitzungen über den Reader-Endpunkt, der die Abfrage-Workload über mehrere DB-Instances im Cluster verteilt.

Skalieren Sie den Cluster horizontal, indem Sie weitere Aurora Replicas hinzufügen. Informationen zu diesen Verfahren finden Sie unter Replikation mit Amazon Aurora. Skalieren Sie die DB-Instances horizontal nach oben oder unten, indem Sie die AWS-Instance-Klasse ändern. Verstehen Sie, wie Aurora diese Art von Vorgängen vereinfacht, sodass Sie später nicht wieder von vorne beginnen müssen, sollte Ihre anfängliche Schätzung der Systemkapazität falsch sein.

Erstellen Sie einen Snapshot und stellen Sie ihn in einem anderen Cluster wieder her.

Stellen Sie durch Untersuchen der Cluster-Metriken fest, welche Aktivitäten im Zeitverlauf auftreten und wie sich die Metriken auf die DB-Instances im Cluster beziehen.

Es ist hilfreich, sich damit vertraut zu machen, wie diese Aufgaben anfänglich über die AWS-Managementkonsole ausgeführt werden. Wenn Sie die Verwendungsmöglichkeiten von Aurora verstehen, können Sie dazu übergehen, diese Operationen mithilfe der AWS CLI zu automatisieren. In den folgenden Abschnitten finden Sie weitere Informationen zu den Verfahren und bewährten Methoden für diese Aktivitäten während des Machbarkeitsnachweises.

Durchführen einer Übung mit der AWS CLI

Wir empfehlen, Bereitstellungs- und Verwaltungsverfahren selbst im Rahmen eines Machbarkeitsnachweises zu automatisieren. Machen Sie sich hierzu mit der AWS CLI vertraut, sofern noch nicht geschehen. Wenn Sie die MySQL-kompatiblen und PostgreSQL-kompatiblen Datenbank-Engines mit Amazon RDS verwenden, kommen Ihnen diese Kenntnisse bei der Verwendung von Aurora zugute.

Aurora wirkt sich in der Regel auf Gruppen von DB-Instances aus, die als Cluster angeordnet sind. Viele Operationen bestehen daher daraus, zu bestimmen, welche DB-Instances zu einem Cluster gehören, und dann administrative Operationen als Schleife für alle Instances auszuführen.

Sie können beispielsweise Schritte automatisieren, um Aurora-Cluster zu erstellen und sie dann mit größeren Instance-Klassen nach oben oder mit zusätzlichen DB-Instances horizontal zu skalieren.

Auf diese Weise können Sie beliebige Phasen des Machbarkeitsnachweises wiederholen und Was-wäre-wenn-Szenarien mit unterschiedlichen Konfigurationen von Aurora-Clustern zu erkunden.

Machen Sie sich mit den Fähigkeiten und Einschränkungen von Infrastruktur-Bereitstellungstools wie vertrau AWS CloudFormation. Sie stellen möglicherweise fest, dass von Ihnen im Kontext des Machbarkeitsnachweises durchgeführte Aktivitäten nicht für den Produktionseinsatz geeignet sind. Das CloudFormation-Verhalten bei Änderungen ist beispielsweise, eine neue Instance zu erstellen und die aktuelle Instance einschließlich ihrer Daten zu löschen. Weitere Einzelheiten zu diesem Verhalten finden Sie unter Aktualisierungsverhalten von Stack-Ressourcen im AWS CloudFormation-Benutzerhandbuch.

4. Erstellen Ihres Aurora-Clusters

Mit Aurora können Sie Was-wäre-wenn-Szenarien erkunden, indem Sie DB-Instances zum Cluster hinzufügen und die DB-Instances auf leistungsfähigere Instance-Klassen hochskalieren. Sie können auch Cluster mit anderen Konfigurationseinstellungen erstellen, um die gleiche Workload parallel auszuführen. Mit Aurora haben Sie sehr viel Flexibilität zum Einrichten, Außerbetriebnehmen und Neukonfigurieren von DB-Clustern. Angesichts dieser Optionen ist es hilfreich, diese Methoden in den frühen Phasen des Machbarkeitsnachweises auszuprobieren. Allgemeine Verfahren zum Erstellen von Aurora-Clustern finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Starten Sie nach Möglichkeit mit einem Cluster mit den folgenden Einstellungen. Überspringen Sie diesen Schritt nur, wenn Ihnen bestimmte Anwendungsfälle vorschweben. So können Sie diesen Schritt beispielsweise überspringen, wenn für Ihren Anwendungsfall eine besondere Art von Aurora-Cluster erforderlich ist. Oder Sie können ihn überspringen, wenn Sie an einer bestimmten Kombination von Datenbank-Engine und -Version interessiert sind.

- Deaktivieren Sie Easy create (Einfache Erstellung). Wir empfehlen, dass Sie sich für den Machbarkeitsnachweis über alle von Ihnen gewählten Einstellungen bewusst sind, sodass Sie anschließend identische oder geringfügig andere Cluster erstellen können.
- Verwenden Sie eine aktuelle DB-Engine-Version. Diese Kombinationen aus Datenbank-Engine und -Version sind weitgehend mit anderen Aurora-Funktionen kompatibel und werden von Kunden in großem Umfang für Produktionsanwendungen genutzt.
  - Aurora-MySQL-Version 3.x (mit MySQL 8.0 kompatibel)
  - Aurora-PostgreSQL-Version 15.x oder 16.x
- Wählen Sie die Vorlage Dev/Test. Die Wahl dieser Option ist für Ihre Machbarkeitsnachweis-Aktivitäten ohne Bedeutung.

- Wählen Sie für DB instance class (DB-Instance-Klasse) die Option Memory optimized classes (Speicheroptimierte Klassen) und eine der xlarge-Instance-Klassen aus. Sie können die Instance-Klasse zu einem späteren Zeitpunkt nach oben oder unten skalieren.
- Wählen Sie unter Multi-AZ Deployment (Multi-AZ-Bereitstellung) die Option Create an Aurora Replica or Reader node in a different AZ (Aurora-Replikat- oder Reader-Knoten in anderer AZ erstellen) aus. Viele der nützlichsten Aspekte von Aurora betreffen Cluster aus mehreren DB-Instances. Es ist sinnvoll, immer mit mindestens zwei DB-Instances in einem neuen Cluster zu beginnen. Beim Testen unterschiedlicher Szenarien mit hoher Verfügbarkeit ist es hilfreich, für die zweite DB-Instance eine andere Availability Zone zu verwenden.
- Halten Sie sich bei der Wahl von Namen für die DB-Instances an eine generische Namenskonvention. Bezeichnen Sie eine Cluster-DB-Instance nicht als „Writer“, da verschiedene DB-Instances diese Rollen nach Bedarf annehmen. Wir empfehlen, so etwas wie `clustername-az-serialnumber` zu verwenden, z. B. `myprodappdb-a-01`. Durch diese Angaben werden die DB-Instance und ihre Platzierung eindeutig identifiziert.
- Legen Sie eine lange Aufbewahrungsfrist für Backups des Aurora-Clusters fest. Eine lange Aufbewahrungsfrist ermöglicht Ihnen, für einen Zeitraum von bis zu 35 Tagen eine zeitpunktbezogene Wiederherstellung (Point-in-Time Recovery, PITR) durchzuführen. Sie können Ihre Datenbank nach dem Durchführen von Tests mit DDL- und Data Manipulation Language (DML)-Anweisungen auf einen bekannten Zustand zurücksetzen. Eine Wiederherstellung ist auch möglich, wenn Sie versehentlich Daten löschen oder ändern.
- Aktivieren Sie beim Erstellen des Clusters zusätzliche Wiederherstellungs-, Protokollierungs- und Überwachungsfunktionen. Aktivieren Sie alle Optionen, die unter Rückverfolgung, Performance Insights, Überwachung und Protokollexporte verfügbar sind. Wenn diese Funktionen aktiviert sind, können Sie die Eignung von Funktionen wie Rückverfolgung, erweiterte Überwachung oder Performance Insights for Ihre Workload testen. Auf diese Weise lässt sich während des Machbarkeitsnachweises leicht die Leistung überprüfen und eine Fehlerbehebung durchführen.

5. Einrichten Ihres Schemas

Richten Sie im Aurora-Cluster Datenbanken, Tabellen, Indizes, Fremdschlüssel und andere Schemaobjekte für Ihre Anwendung ein. Wenn Sie bereits mit einem anderen MySQL-kompatiblen oder PostgreSQL-kompatiblen Datenbanksystem gearbeitet haben, ist zu erwarten, dass sich diese Phase einfach und schnell gestaltet. Sie verwenden für Ihre Datenbank-Engine die gleiche SQL-Syntax und Befehlszeile oder andere Client-Anwendungen, mit denen Sie vertraut sind.

Um SQL-Anweisungen in Ihrem Cluster auszugeben, identifizieren Sie den Cluster-Endpunkt und stellen Sie diesen Wert als Verbindungsparameter für Ihre Client-Anwendung bereit. Sie finden den Cluster-Endpunkt auf der Registerkarte Connectivity (Konnektivität) der Detailseite Ihres Clusters. Der Cluster-Endpunkt ist mit Writer beschriftet. Der andere mit Reader beschriftete Endpunkt stellt eine schreibgeschützte Verbindung dar, die Sie für Endbenutzer bereitstellen können, die Berichte oder andere schreibgeschützte Abfragen ausführen. Unterstützung bei Problemen mit dem Verbinden Ihres Clusters finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

Wenn Sie Ihr Schema und Ihre Daten von einem anderen Datenbanksystem portieren, müssen Sie an dieser Stelle wahrscheinlich einige Änderungen am Schema vornehmen. Diese Schemaänderungen sind entsprechend der SQL-Syntax und den Funktionen vorzunehmen, die in Aurora verfügbar sind. An dieser Stelle können Sie bestimmte Spalten, Einschränkungen, Auslöser oder andere Schemaobjekte weglassen. Dies kann insbesondere dann hilfreich sein, wenn für die Aurora-Kompatibilität Änderungen an Objekten erforderlich sind, die für Ihren beabsichtigten Machbarkeitsnachweis jedoch nicht signifikant sind.

Wenn Sie von einem Datenbanksystem mit einer anderen zugrunde liegenden Engine als der von Aurora migrieren, sollten Sie AWS Schema Conversion Tool (AWS SCT) verwenden, um den Prozess zu vereinfachen. Einzelheiten finden Sie im AWS-Schema-Conversion-Tool-Benutzerhandbuch. Allgemeine Informationen zu Migrations- und Portierungsaktivitäten finden Sie im AWS-Whitepaper Migration Ihrer Datenbanken auf Amazon Aurora.

Während dieser Phase können Sie Ihr Schema-Setup auf Ineffizienzen untersuchen, z. B. bei Ihrer Indizierungsstrategie oder bei anderen Tabellenstrukturen, wie z. B. partitionierten Tabellen. Solche Ineffizienzen können verstärkt werden, wenn Sie Ihre Anwendung in einem Cluster mit mehreren DB-Instances und einer hohen Workload bereitstellen. Erwägen Sie, ob Sie solche Aspekte der Leistung zu diesem Zeitpunkt oder bei späteren Aktivitäten wie z. B. einem vollständigen Benchmark-Test, optimieren können.

6. Importieren Ihrer Daten

Während des Machbarkeitsnachweises verlagern Sie die Daten, oder eine repräsentative Stichprobe, von Ihrem ehemaligem Datenbanksystem. Richten Sie nach Möglichkeit mindestens einige Daten in Ihren Tabellen ein. Dadurch wird das Testen der Kompatibilität aller Datentypen und Schema-Funktionen ermöglicht. Wenn Sie eine Übung der grundlegenden Aurora-Funktionen durchgeführt haben, skalieren Sie die Datenmenge nach oben. Zu dem Zeitpunkt, wenn Sie Ihren Machbarkeitsnachweis fertigstellen, sollten Sie Ihre ETL-Tools, Abfragen und Gesamt-Workload mit einem Dataset testen, das groß genug für genaue Schlussfolgerungen ist.

Sie können mehrere Methoden zum Importieren von physischen oder logischen Sicherungsdaten in Aurora verwenden. Einzelheiten finden Sie unter Migrieren von Daten zu einem Amazon Aurora MySQL-DB-Cluster oder Migrieren von Daten nach Amazon Aurora mit PostgreSQL-Kompatibilität, abhängig von der Datenbank-Engine, die Sie bei dem Machbarkeitsnachweis verwenden.

Experimentieren Sie mit den ETL-Tools und Technologien, die Sie in Betracht ziehen. Sehen Sie, welche am besten Ihren Anforderungen genügen. Berücksichtigen Sie sowohl Durchsatz als auch Flexibilität. Einige ETL-Tools führen z. B. einmalige Übertragungen durch, während andere fortlaufende Replikationen von dem alten System zu Aurora umfassen.

Wenn Sie von einem MySQL-kompatiblen System zu Aurora MySQL migrieren, könne Sie die nativen Datenübertragungstools verwenden. Das Gleiche gilt, wenn Sie von einem PostgreSQL-kompatiblen System zu Aurora PostgreSQL migrieren. Wenn Sie von einem Datenbanksystem mit einer anderen zugrunde liegenden Engine als der von Aurora migrieren, können Sie mit AWS Database Migration Service (AWS DMS) experimentieren. Weitere Informationen zu AWS DMS finden Sie im AWS Database Migration Service-Benutzerhandbuch.

Einzelheiten zu Migrations- und Portierungsaktivitäten finden Sie im AWS-Whitepaper Aurora-Migrationshandbuch.

7. Portieren Ihres SQL-Codes

Der zum Ausprobieren von SQL und zugehörigen Anwendungen erforderliche Aufwand variiert je nach den unterschiedlichen Fällen. Der Aufwand ist insbesondere davon abhängig, ob Sie von einem MySQL-kompatiblen oder einem PostgreSQL-kompatiblen System oder von einer anderen Art von System portieren.

- Wenn Sie von RDS für MySQL oder RDS für PostgreSQL portieren, sind die SQL-Änderungen klein genug, sodass Sie den SQL-Originalcode mit Aurora ausprobieren und erforderliche Änderungen manuell einfügen können.
- Wenn Sie von einer lokalen Datenbank portieren, die mit MySQL oder PostgreSQL kompatibel ist, können Sie den SQL-Originalcode ebenfalls ausprobieren und erforderliche Änderungen manuell einfügen.
- Wenn Sie von einer anderen kommerziellen Datenbank portieren, sind die erforderlichen SQL-Änderungen etwas umfangreicher. Ziehen Sie in diesem Fall die Verwendung des in Betrach AWS SCT.

Während dieser Phase können Sie Ihr Schema-Setup auf Ineffizienzen untersuchen, z. B. bei Ihrer Indizierungsstrategie oder bei anderen Tabellenstrukturen, wie z. B. partitionierten Tabellen. Erwägen Sie, ob Sie solche Aspekte der Leistung zu diesem Zeitpunkt oder bei späteren Aktivitäten wie z. B. einem vollständigen Benchmark-Test, optimieren können.

Sie können die Logik der Datenbankverbindung in Ihrer Anwendung überprüfen. Um die Vorteile der verteilten Verarbeitung von Aurora nutzen zu können, müssen Sie möglicherweise separate Verbindungen für Lese- und Schreibvorgänge und relativ kurze Sitzungen für Abfrageoperationen verwenden. Weitere Informationen zu Verbindungen finden Sie unter 9. Verbinden mit Aurora.

Überlegen Sie, ob Sie Kompromisse eingehen und Abstriche machen müssen, um in Ihrer Produktionsdatenbank Probleme zu vermeiden. Berücksichtigen Sie in der Zeitplanung des Machbarkeitsnachweises die Vornahme von Verbesserungen am Schemadesign und an Abfragen. Um zu beurteilen, ob sich Leistung, Betriebskosten und Skalierbarkeit leicht verbessern lassen, probieren Sie die ursprünglichen und die abgewandelten Anwendungen nebeneinander in verschiedenen Aurora-Clustern aus.

Einzelheiten zu Migrations- und Portierungsaktivitäten finden Sie im AWS-Whitepaper Aurora-Migrationshandbuch.

8. Angeben der Konfigurationseinstellungen

Als Teil des Aurora-Machbarkeitsnachweises können Sie auch die Datenbank-Konfigurationsparameter überprüfen. Sie haben Ihre MySQL- oder PostgreSQL-Konfigurationseinstellungen möglicherweise bereits für Leistung und Skalierbarkeit in Ihrer aktuellen Umgebung optimiert. Das Aurora-Speichersubsystem wird für eine verteilte Cloud-basierte Umgebung mit einem Speichersubsystem hoher Geschwindigkeit angepasst und optimiert. Dies hat zur Folge, dass viele ehemaligen Einstellungen der Datenbank-Engine nicht anwendbar sind. Wir empfehlen, Ihre anfänglichen Experimente mit Aurora-Standard-Konfigurationseinstellungen durchzuführen. Wenden Sie Einstellungen von Ihrer aktuellen Umgebung nur dann erneut an, falls Engpässe bezüglich der Leistung und Skalierbarkeit auftreten. Wenn Sie interessiert sind, können Sie sich unter Introducing the Aurora Storage Engine im AWS Database Blog näher mit diesem Thema beschäftigen.

Aurora erleichtert die Wiederverwendung der optimalen Konfigurationseinstellungen für eine bestimmte Anwendung oder einen bestimmten Anwendungsfall. Anstatt für jede DB-Instance eine separate Konfigurationsdatei zu bearbeiten, verwalten Sie Parametersätze, die Sie ganzen Clustern oder bestimmten DB-Instances zuweisen. So gilt die Zeitzoneneinstellung beispielsweise für alle DB-

Instances im Cluster und Sie könne die Größeneinstellung des Seiten-Caches für jede DB-Instance anpassen.

Sie beginnen mit einem der Standard-Parametersätze und wenden nur auf die Parameter Änderungen an, die optimiert werden müssen. Details zur Arbeit mit Parametergruppen finden Sie unter Amazon Aurora-DB-Cluster und DB-Instance-Parameter. Konfigurationseinstellungen, die für Aurora-Cluster anwendbar oder nicht anwendbar sind, finden Sie unter Aurora MySQL Konfigurationsparameter oder Amazon-Aurora-PostgreSQL-Parameter, abhängig von Ihrer Datenbank-Engine.

9. Verbinden mit Aurora

Wie Sie beim Erstellen des anfänglichen Schemas und Daten-Setups und beim Ausführen von Beispielabfragen festgestellt haben, können Sie eine Verbindung mit verschiedenen Endpunkten in einem Aurora-Cluster herstellen. Welcher Endpunkt zu verwenden ist, hängt davon ab, ob die Operation ein Lesevorgang, wie z. B. eine `SELECT`-Anweisung, oder ein Schreibvorgang, wie z. B. eine `CREATE`- oder `INSERT`-Anweisung ist. Während Sie die Workload auf einem Aurora-Cluster erhöhen und mit Aurora-Funktionen experimentieren, ist es wichtig, dass Ihre Anwendung je Operation dem entsprechenden Endpunkt zuweist.

Indem Sie den Cluster-Endpunkt für Schreiboperationen verwenden, stellen Sie stets eine Verbindung mit einer DB-Instance in dem Cluster her, das eine Lese-Schreib-Funktionalität besitzt. Standardmäßig verfügt nur eine einzige DB-Instance in einem Aurora-Cluster über eine Lese-Schreib-Funktionalität. Diese DB-Instance wird als die primäre Instance bezeichnet. Wenn die ursprüngliche primäre Instance ausfällt, aktiviert Aurora einen Failover-Mechanismus und eine andere DB-Instance wird zur primären Instance.

Durch Weiterleiten von `SELECT`-Anweisungen an den Reader-Endpunkt verteilen Sie gleichermaßen die Arbeit von Verarbeitungsabfragen auf die DB-Instances im Cluster. Mithilfe der Round-Robin-DNS-Auflösung wird jede Reader-Verbindung einer anderen DB-Instance zugewiesen. Wenn die meisten Abfragevorgänge auf den schreibgeschützten DB-Aurora Replicas durchgeführt werden, verringert sich die Last auf der primären Instance, sodass sie mehr DDL- und DML-Anweisungen verarbeiten kann.

Durch Nutzung dieser Endpunkte wird die Abhängigkeit von hartcodierten Hostnamen reduziert und Ihre Anwendung kann bei DB-Instance-Ausfällen schneller wiederhergestellt werden.

> **Note**
>
> Aurora verfügt auch über benutzerdefinierte Endpunkte, die Sie erstellen. Diese Endpunkte müssen bei einem Machbarkeitsnachweis gewöhnlich nicht berücksichtigt werden.

Die Aurora Replicas unterliegen einer Replica-Verzögerung. Diese Verzögerung dauert gewöhnlich aber nur 10 bis 20 Milliseconds an. Sie können die Replikationsverzögerung überwachen und entscheiden, ob sie innerhalb des Bereichs Ihrer Datenkonsistenzanforderungen liegt. In einigen Fällen erfordern Ihre Leseabfragen unter Umständen starke Lesekonsistenz (Lesen-nach-Schreiben-Konsistenz). In diesen Fällen können Sie für sie weiterhin den Cluster-Endpunkt anstatt des Reader-Endpunkts verwenden.

Um die Aurora-Funktionen bei der verteilten Parallelausführung voll nutzen zu können, müssen Sie möglicherweise Änderungen an der Verbindungslogik vornehmen. Sie möchten vermeiden, dass alle Leseanforderungen an die primäre Instance gesendet werden. Die schreibgeschützten Aurora Replicas stehen mit genau denselben Daten für die Verarbeitung von `SELECT`-Anweisungen bereit. Kodieren Sie Ihre Anwendungslogik so, dass für jede Art von Vorgang der entsprechende Endpunkt verwendet wird. Befolgen Sie diese allgemeinen Richtlinien:

- Vermeiden Sie es, eine einzelne hartcodierte Verbindungszeichenfolge für alle Datenbanksitzungen zu verwenden.
- Sofern praktikabel, schließen Sie Schreiboperationen wie DDL- und DML-Anweisungen in Funktionen in Ihrem Client-Anwendungscode ein. Auf diese Weise können Sie bewirken, dass verschiedene Arten von Vorgängen bestimmte Verbindungen nutzen.
- Machen Sie separate Funktionen für Abfrageoperationen. Aurora weist jede neue Verbindung mit dem Reader-Endpunkt einem anderen Aurora-Replikat zu, um die Last leseintensiver Anwendungen auszugleichen.
- Schließen Sie bei Operationen mit Abfragesätzen die Verbindung mit dem Reader-Endpunkt und öffnen Sie sie erneut, wenn jeder Satz zugehöriger Abfragen abgeschlossen wurde. Nutzen Sie Verbindungspooling, sofern diese Funktion in Ihrem Software-Stack verfügbar ist. Durch die Weiterleitung von Abfragen zu verschiedenen Verbindungen kann Aurora die Lese-Workload leichter unter den DB-Instances im Cluster verteilen.

Allgemeine Informationen zur Verbindungsverwaltung und zu Endpunkten für Aurora finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster. Detaillierte Einblicke zu diesem Thema finden Sie in Aurora MySQL Database Administrator's Handbook – Connection Management.

10. Ausführen Ihrer Workload

Wenn Schema, Daten und Konfigurationseinstellungen vorhanden sind, können Sie mit der Arbeit mit dem Cluster beginnen, indem Sie Ihre Workload ausführen. Verwenden Sie im Machbarkeitsnachweis eine Workload, die die Hauptaspekte Ihrer Produktions-Workload widerspiegelt. Wir empfehlen, Entscheidungen über die Leistung immer anhand von realen Tests und Workloads und nicht anhand von synthetischen Benchmark-Tests wie sysbench oder TPC-C zu treffen. Sammeln Sie, wo immer möglich, Messungen basierend auf Ihrem eigenen Schema, Abfragemustern und Nutzungsvolumen.

Sofern praktikabel, replizieren Sie die tatsächlichen Bedingungen, unter denen die Anwendung ausgeführt wird. Sie führen Ihren Anwendungscode beispielsweise gewöhnlich auf Amazon EC2-Instances in derselben AWS-Region und derselben Virtual Private Cloud (VPC) wie der Aurora-Cluster durch. Wenn Ihre Produktionsanwendung auf mehreren EC2-Instances ausgeführt wird, die sich über mehrere Availability Zones erstrecken, dann richten Sie Ihre Machbarkeitsnachweis-Umgebung genauso ein. Weitere Informationen zu AWS-Regionen finden Sie unter Regionen und Availability Zones im Amazon-RDS-Benutzerhandbuch. Weitere Informationen zum Amazon VPC-Service finden Sie unter Was ist Amazon VPC? im Amazon VPC Benutzerhandbuch.

Nachdem Sie überprüft haben, dass die Grundfunktionen Ihrer Anwendung funktionieren und dass Sie über Aurora auf die Daten zugreifen können, können Sie mit Aspekten des Aurora-Clusters versuchsweise arbeiten. Einige Funktionen, die Sie ausführen sollten, sind gleichzeitige Verbindung mit Load Balancing, gleichzeitige Transaktionen und automatische Replikation.

An dieser Stelle sollten Sie mit den Mechanismen zur Datenübertragung vertraut sein und können mit einem größeren Teil von Beispieldaten Tests ausführen.

In dieser Phase sehen Sie die Auswirkungen von Änderungen an den Konfigurationseinstellungen, wie z. B. Speicherlimits und Verbindungslimits. Betrachten Sie sich erneut die Verfahren, die Sie unter erkundet habe 8. Angeben der Konfigurationseinstellungen.

Sie können auch mit Mechanismen wie Erstellen und Wiederherstellen von Snapshots experimentieren. So können Sie beispielsweise Cluster mit verschiedenen AWS-Instance-Klassen, einer unterschiedlichen Anzahl von AWS Replicas usw. erstellen. Sie können dann auf jedem Cluster

den gleichen Snapshot mit Ihrem Schema und allen Ihren Daten wiederherstellen. Einzelheiten zu diesem Zyklus finden Sie unter Erstellen eines DB-Cluster-Snapshots and Wiederherstellen aus einem DB-Cluster-Snapshot.

11. Messen der Leistung

Die bewährten Methoden in diesem Bereich sollen sicherstellen, dass die richtigen Tools und Prozesse eingerichtet werden, um anormales Verhalten während Workload-Vorgängen schnell aufzudecken. Sie sollen außerdem gut sichtbar sein, sodass Sie alle zutreffenden Ursachen zuverlässig identifizieren können.

Durch Ansicht der Registerkarte Monitoring (Überwachung) können Sie immer den aktuellen Zustand des Clusters sehen oder Trends im Zeitverlauf analysieren. Diese Registerkarte ist über die Detailseite der Konsole für jeden Aurora-Cluster oder jede DB-Instance verfügbar. Sie zeigt Metriken des Amazon CloudWatch-Überwachungsservice in Form von Diagrammen an. Sie können die Metriken nach Name, DB-Instance und Zeitraum filtern.

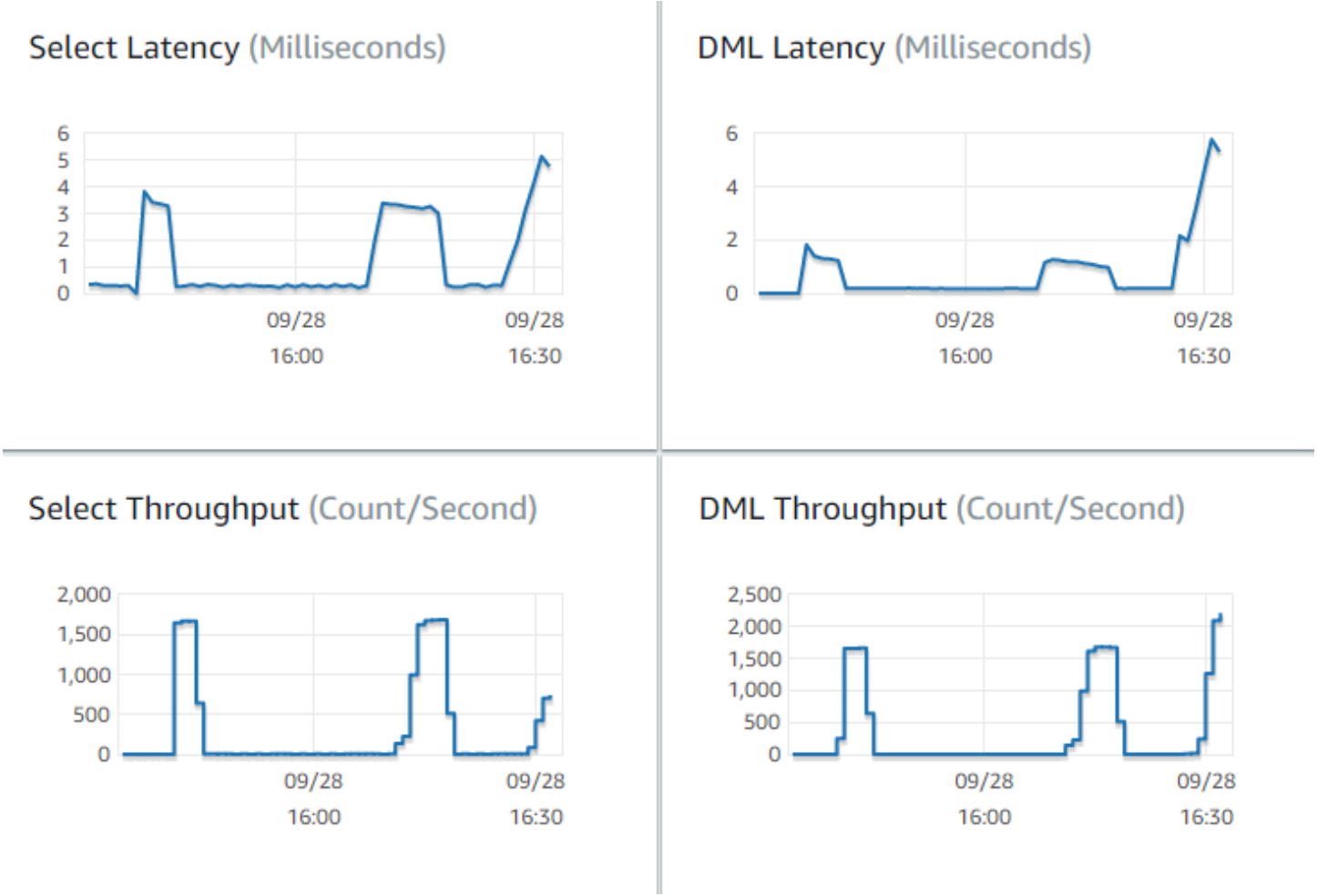
Wenn sich weitere Optionen auf der Registerkarte Monitoring (Überwachung) befinden, aktivieren Sie "Enhanced Monitoring (Erweiterte Überwachung)" und "Performance Insights" in den Cluster-Einstellungen. Sie können diese Optionen auch zu einem späteren Zeitpunkt aktvieren, sofern Sie sie beim Einrichten des Clusters nicht ausgewählt haben.

Zum Messen der Leistung stützen Sie sich größtenteils auf die Diagramme der Aktivitäten für den gesamten Aurora-Cluster. Sie können überprüfen, ob die Aurora Replicas ähnliche Lade- und Antwortzeiten aufweisen. Sie können auch erkennen, wie die Arbeit zwischen der primären Lese-Schreib-Instance und den schreibgeschützten Aurora-Replicas aufgeteilt wird. Bei einem Ungleichgewicht zwischen den DB-Instances oder bei einem Problem mit nur einer DB-Instance können Sie die Registerkarte Monitoring (Überwachung) bezüglich dieser spezifischen Instance untersuchen.

Nachdem die Umgebung und die tatsächliche Workload zum Emulieren Ihrer Produktionsumgebung eingerichtet wurden, können Sie die Leistung von Aurora messen. Die wichtigsten zu beantwortenden Fragen sind:

- Wie viele Abfragen pro Sekunden werden von Aurora verarbeitet? Sie können den Throughput (Durchsatz)-Metriken die Werte für verschiedene Arten von Vorgängen entnehmen.
- Wie lange braucht Aurora durchschnittlich zum Verarbeiten einer gegebenen Abfrage? Sie können den Latency (Latenz)-Metriken die Werte für verschiedene Arten von Vorgängen entnehmen.

Um die Durchsatz- und Latenzmetriken einzusehen, gehen Sie zur Registerkarte Überwachung für einen bestimmten Aurora-Cluster in der Amazon-RDS-Konsole. Der folgende Screenshot zeigt ein Beispiel für die Metriken Latenz auswählen, DML-Latenz, Durchsatz auswählen und DML-Durchsatz auf der Registerkarte Überwachung.



Wenn möglich, stellen Sie Baseline-Werte für diese Metriken in Ihrer aktuellen Umgebung auf. Wenn dies nicht praktikabel ist, erstellen Sie eine Baseline für den Aurora-Cluster, indem Sie eine Workload entsprechend Ihrer Produktionsanwendung ausführen. Führen Sie beispielsweise Ihre Aurora-Workload mit einer ähnlichen Anzahl gleichzeitiger Benutzer und Abfragen aus. Beobachten Sie dann, wie sich die Werte ändern, während Sie mit verschiedenen Instance-Klassen, Cluster-Größen, Konfigurationseinstellungen usw. experimentieren.

Wenn die Durchsatzwerte niedriger als erwartet sind, untersuchen Sie näher, welche Faktoren sich auf die Datenbank-Leistung Ihrer Workload auswirken. Untersuchen Sie höher als erwartete Latenzwerte gleichermaßen näher. Überwachen Sie hierzu die sekundären Metriken für den DB-Server (CPU, Arbeitsspeicher usw.). Sie können erkennen, ob sich die DB-Instances an ihre

Grenzwerte annähern. Auch ist ersichtlich, über wie viel extra Kapazität Ihre DB-Instances für die Verarbeitung weiterer gleichzeitiger Abfragen, Abfragen großer Tabellen usw. verfügt.

> **Tip**
>
> Richten Sie CloudWatch-Alarme zum Erkennen von Metrikwerten ein, die außerhalb der erwarteten Bereiche fallen.

Wenn Sie die ideale Aurora-Cluster-Größe und -Kapazität bewerten, können Sie herausfinden, mit welcher Konfiguration die Spitzenleistung der Anwendung erzielt wird, ohne unnötige Ressourcen bereitzustellen. Ein wichtiger Faktor ist, die angemessene Größe für die DB-Instances im Aurora-Cluster zu finden. Beginnen Sie mit der Wahl einer Instance-Größe mit einer ähnlichen CPU- und Arbeitsspeicher-Kapazität wie Ihre aktuelle Produktionsumgebung. Erfassen Sie Durchsatz- und Latenzwerte für die Workload bei dieser Instance-Größe. Skalieren Sie die Instance dann nach oben auf die nächste Größe. Überprüfen Sie, ob sich die Durchsatz- und Latenzwerte bessern. Skalieren Sie die Instance auch nach unten. Überprüfen Sie, ob die Latenz- und Durchsatzwerte gleich bleiben. Unser Ziel ist, den höchsten Durchsatz bei der geringsten Latenz für die kleinst mögliche Instance zu erreichen.

> **Tip**
>
> Bemessen Sie Ihre Aurora-Cluster und zugehörigen DB-Instances mit genug vorhandener Kapazität, um plötzlichen, nicht vorhersehbaren Datenverkehrsspitzen gewachsen zu sein. Belassen Sie für geschäftskritische Datenbanken mindestens 20 Prozent an Extrakapazität für CPU und Arbeitsspeicher.

Führen Sie Leistungstests lange genug aus, sodass die Datenbankleistung in einem warmen, stabilen Zustand gemessen wird. Sie müssen die Workload viele Minuten oder sogar einige Stunden lang ausführen, bis dieser stabile Zustand erreicht wird. Gewisse Abweichungen am Anfang der Ausführung sind normal. Diese Abweichungen sind dadurch bedingt, dass jedes Aurora Replica seinen Zwischenspeicher basierend auf den verarbeiteten `SELECT`-Abfragen aufwärmt.

Aurora weist die beste Leistung bei transaktionalen Workloads mit mehreren gleichzeitigen Benutzern und Abfragen auf. Um sicherzustellen, dass die verarbeitete Last für eine optimale Leistung ausreicht, führen Sie Benchmark-Tests mit Multithreading oder mehrere Instances von Leistungstests gleichzeitig aus. Messen Sie die Leistung mit Hunderten oder sogar Tausenden gleichzeitiger Client-

Threads. Simulieren Sie die Anzahl gleichzeitiger Threads, die Sie in Ihrer Produktionsumgebung erwarten. Sie können auch zusätzliche Stresstests mit weiteren Threads durchführen, um die Aurora-Skalierbarkeit zu messen.

12. Ausführen der Aurora-Hochverfügbarkeit

Viele der Hauptfunktionen von Aurora umfassen hohe Verfügbarkeit. Zu diesen Funktionen gehören automatische Replikation, automatisches Failover, automatische Backups mit zeitbezogener Wiederherstellung und die Fähigkeit zum Hinzufügen von DB-Instances zum Cluster. Die Sicherheit und Zuverlässigkeit von Funktionen wie diesen ist für geschäftskritische Anwendungen wichtig.

Die Auswertung dieser Funktionen erfordert eine bestimmte Denkweise. Bei früheren Aktivitäten, wie Leistungsmessungen, beobachten Sie die Systemleistung, wenn alles richtig funktioniert. Beim Testen der hohen Verfügbarkeit müssen Sie Verhalten im Schlimmstfall durchdenken. Sie müssen sich verschiedene Arten von Fehlern vorstellen, selbst wenn solche Bedingungen nur selten auftreten. Sie führen möglicherweise absichtlich Probleme herbei, um sicherzustellen, dass der Systembetrieb richtig und schnell wiederhergestellt wird.

> **Tip**
>
> Richten Sie für einen Machbarkeitsnachweis alle DB-Instances in einem Aurora-Cluster mit derselben AWS-Instance-Klasse ein. Dadurch ist es möglich, Aurora-Verfügbarkeitsfunktionen ohne große Änderungen an der Leistung und Skalierbarkeit auszuprobieren, während Sie DB-Instances offline nehmen, um Fehler zu simulieren.

Wir empfehlen, in jedem Aurora-Cluster mindestens zwei Instances zu verwenden. Die DB-Instances in einem Aurora-Cluster können sich über bis zu drei Availability Zones (AZs) erstrecken. Suchen Sie jede der ersten zwei oder drei DB-Instances in einer anderen AZ. Wenn Sie mit der Verwendung größerer Cluster beginnen, verteilen Sie Ihre DB-Instances über alle AZs in Ihrer AWS-Region. Dadurch wird die Fähigkeit zur Fehlertoleranz erhöht. Wenn sich ein Problem auf die gesamte AZ auswirkt, kann Aurora ein Failover auf eine DB-Instance in einer anderen AZ durchführen. Wenn Sie einen Cluster mit mehr als drei Instances ausführen, verteilen Sie die DB-Instances so gleichmäßig wie möglich über alle drei AZs.

> **Tip**
>
> Der Speicher eines Aurora-Clusters ist unabhängig von den DB-Instances. Der Speicher eines jeden Aurora-Clusters erstreckt sich immer über drei AZs.
> Wenn Sie die Funktionen für hohe Verfügbarkeit testen, verwenden Sie immer DB-Instances mit identischer Kapazität in Ihrem Test-Cluster. Dadurch lassen sich unvorhersehbare Änderungen in Leistung, Latenz usw. vermeiden, wenn eine DB-Instance die Rolle einer anderen übernimmt.

Informationen zum Simulieren von Fehlerbedingungen zum Testen von Funktionen für hohe Verfügbarkeit finden Sie unter Testen von Amazon Aurora MySQL unter Verwendung von Fehlersimulationsabfragen.

Als Teil Ihrer Machbarkeitsnachweis-Übung besteht ein Ziel darin, die ideale Anzahl von DB-Instances und die optimale Instance-Klasse für diese DB-Instances zu finden. Dazu müssen die Anforderungen für hohe Verfügbarkeit und für Leistung gegeneinander abgewogen werden.

Für Aurora gilt, dass je mehr DB-Instances sich in einem Cluster befinden, desto höher die Vorteile für hohe Verfügbarkeit sind. Mehr DB-Instances verbessern auch die Skalierbarkeit von leseintensiven Anwendungen. Aurora kann mehrere Verbindungen für `SELECT`-Abfragen unter den schreibgeschützten Aurora-Replikaten verteilen.

Andererseits wird durch Einschränken der Anzahl von DB-Instances der Replikationsdatenverkehr vom primären Knoten verringert. Der Replikationsdatenverkehr belegt Netzwerkbandbreite, wobei es sich um einen weiteren Aspekt der Gesamtleistung und Skalierbarkeit handelt. Daher ist bei schreibintensiven OLTP-Anwendungen eine kleiner Anzahl großer DB-Instances gegenüber vielen kleinen DB-Instances vorzuziehen.

In einem typischen Aurora-Cluster verarbeitet eine DB-Instance (die primäre Instance) alle DDL- und DML-Anweisungen. Die anderen DB-Instances (die Aurora Replicas) verarbeiten nur `SELECT`-Anweisungen. Obwohl die DB-Instances nicht genau die gleichen Aufgaben ausführen, raten wir zur Verwendung der gleichen Instance-Klasse für alle DB-Instances im Cluster. Wenn Aurora im Falle eines Ausfalls eine der schreibgeschützten DB-Instances auf die neue primäre Instance hochstuft, hat die primäre Instance auf diese Weise die gleiche Kapazität wie zuvor.

Wenn Sie in demselben Cluster DB-Instances unterschiedlicher Kapazitäten verwenden möchten, richten Sie Failover-Stufen für die DB-Instances ein. Diese Stufen bestimmen die Reihenfolge, in der

Aurora Replicas durch den Failover-Mechanismus hochgestuft werden. Platzieren Sie DB-Instances, die sehr viel größer oder kleiner als die anderen sind, in eine niedrigere Failover-Stufe. Dadurch wird sichergestellt, dass sie zuletzt zum Hochstufen ausgewählt werden.

Führen Sie Übungen mit den Wiederherstellungsfunktionen von Aurora, wie automatische zeitbezogene Wiederherstellung, manuelle Snapshots und Wiederherstellung und Cluster-Rückverfolgung, durch. Sofern angemessen, kopieren Sie Snapshots aus anderen AWS-Regions und stellen Sie sie in anderen AWS-Regionen wieder her, um Szenarien zur Notfallwiederherstellung nachzuahmen.

Untersuchen Sie die Anforderungen Ihrer Organisation bezüglich Wiederherstellungsdauer (Restore Time Objective, RTO), Wiederherstellungspunkt (Restore Point Objective, RPO) und geografischer Redundanz. Die meisten Organisationen gruppieren diese Elemente unter der großen Kategorie der Notfallwiederherstellung. Bewerten Sie die Aurora-Funktionen für hohe Verfügbarkeit, die in diesem Abschnitt beschrieben werden, im Kontext Ihres Prozesses zur Notfallwiederherstellung, um sicherzustellen,dass Ihre RTO- und RPO-Anforderungen erfüllt werden.

13. Weitere Vorgehensweisen

Am Ende eines erfolgreichen Machbarkeitsnachweises bestätigen Sie, dass Aurora basierend auf der erwarteten Workload eine geeignete Lösung für Sie ist. Durchweg durch den voranstehenden Prozess haben Sie überprüft, wie Aurora in einer realistischen Betriebsumgebung funktioniert und anhand Ihrer Erfolgskriterien gemessen.

Sobald Ihre Datenbankumgebung mit Aurora betriebsbereit ist, können Sie mit detaillierten Evaluierungsschritten fortfahren, die zu Ihrer endgültigen Migration und Produktionsbereitstellung führen. Abhängig von Ihrer Situation sind diese anderen Schritte nicht unbedingt Teil des Machbarkeitsnachweises. Einzelheiten zu Migrations- und Portierungsaktivitäten finden Sie im AWS-Whitepaper Aurora-Migrationshandbuch.

Erwägen Sie in einem anderen Schritt die Sicherheitskonfigurationen, die für Ihre Workload relevant sind und darauf ausgelegt sind, Ihre Sicherheitsanforderungen in einer Produktionsumgebung zu erfüllen. Planen Sie, welche Kontrollen zum Schutz des Zugriffs auf die Masterbenutzeranmeldedaten des Aurora-Clusters implementiert werden sollen. Definieren Sie die Rollen und Zuständigkeiten von Datenbankbenutzern, um den Zugriff auf die im Aurora-Cluster gespeicherten Daten zu kontrollieren. Berücksichtigen Sie Datenbank-Zugriffsanforderungen für Anwendungen, Skripts und Drittanbieter-Tools und -Services. Erkunden Sie AWS-Services und -Funktionen wie z. B. AWS Secrets Manager und AWS Identity and Access Management-(IAM)-Authentifizierung.

An dieser Stelle sollten Sie die Verfahren und bewährten Methoden für die Ausführung von Benchmark-Tests mit Aurora verstehen. Sie stellen möglicherweise fest, dass eine zusätzliche Leistungsoptimierung erforderlich ist. Einzelheiten finden Sie unter Verwalten von Performance und Skalierung für einen Aurora-DB-Cluster, Amazon Aurora MySQL-Leistungserweiterungen, Leistung und Skalierung für Amazon Aurora PostgreSQL und Überwachung mit Performance Insights auf . Wenn Sie eine zusätzliche Optimierung vornehmen, stellen Sie sicher, dass Sie mit den Metriken vertraut sind, die Sie während des Machbarkeitsnachweises erfasst haben. Als nächster Schritt können Sie neue Cluster mit verschiedenen Optionen für Konfigurationseinstellungen, Datenbank-Engine und Datenbankversion erstellen. Oder Sie können spezialisierte Arten von Aurora-Clustern erstellen, um die Anforderungen bestimmter Anwendungsfälle zu erfüllen.

Beispielsweise können Sie Aurora-Parallelabfrage-Cluster für hybride Transaktions-/analytische Verarbeitungsanwendungen (HTAP) erkunden. Wenn eine breite geografische Verteilung für die Notfallwiederherstellung oder zum Minimieren der Latenz von zentraler Bedeutung ist, können Sie globale Aurora-Datenbanken erkunden. Wenn Ihre Workload unregelmäßig ist oder wenn Sie Aurora in einem Entwicklungs-/Test-Szenario einsetzen, können Sie Aurora Serverless-Cluster erwägen.

Ihre Produktions-Cluster müssen möglicherweise auch eingehende Verbindungen hohen Umfangs verarbeiten. Informationen zu diesen Techniken finden Sie im AWS-Whitepaper Aurora-MySQL-Datenbankadministratorhandbuch – Verbindungsverwaltung.

Wenn Sie nach dem Machbarkeitsnachweis beschließen, dass Ihr Anwendungsfall nicht für Aurora geeignet ist, erwägen Sie die folgenden anderen AWS-Services:

- Bei rein analytischen Anwendungsfällen profitieren Workloads von einem spaltenbasierten Speicherformat und anderen Funktionen, die besser geeignet für OLAP-Workloads sind. AWS-Services, die sich mit solchen Anwendungsfällen befassen, gehören:
  - Amazon Redshift
  - Amazon EMR
  - Amazon Athena
- Viele Workloads profitieren von einer Kombination aus Aurora mit einem oder mehreren dieser Services. Sie können Daten mit den folgenden Methoden zwischen diesen Services verschieben:
  - AWS Glue
  - AWS DMS
  - Importieren aus Amazon S3, wie im Amazon Aurora Benutzerhandbuch beschrieben
  - Exportieren zu Amazon S3, wie im Amazon Aurora Benutzerhandbuch beschrieben

- Viele andere gängige ETL-Tools

Sicherheit in Amazon Aurora

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die auf die Anforderungen der sicherheitssensibelsten Unternehmen zugeschnitten sind.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Auditoren von Drittanbietern testen und überprüfen die Effektivität unserer Sicherheitsmaßnahmen im Rahmen der AWS -Compliance-Programme regelmäßig. Informationen zu den Compliance-Programmen, die für Amazon Aurora (Aurora) gelten, finden Sie unter AWS services in scope by compliance program.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. In Ihre Verantwortung fallen außerdem weitere Faktoren, wie z. B. die Vertraulichkeit der Daten, die Anforderungen Ihrer Organisation sowie geltende Gesetze und Vorschriften.

Diese Dokumentation erläutert, wie das Modell der geteilten Verantwortung bei der Verwendung von Amazon Aurora zum Tragen kommt. Die folgenden Themen veranschaulichen, wie Sie Amazon Aurora zur Erfüllung Ihrer Sicherheits- und Compliance-Ziele konfigurieren können. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, mit denen Sie Ihre Aurora Aurora-Ressourcen überwachen und sichern können.

Sie können den Zugriff auf Ihre Amazon-Aurora-Ressourcen und Ihre Datenbanken in einem DB-Cluster verwalten. Die Methode, die Sie verwenden, um den Zugriff zu verwalten, hängt vom Aufgabentyp ab, den der Benutzer mit Amazon Aurora ausführen muss:

- Führen Sie Ihren DB-Cluster in einer Virtual Private Cloud (VPC) basierend auf dem Amazon-VPC-Service für die größtmögliche Netzwerkzugriffskontrolle aus. Weitere Informationen zum Erstellen von DB- Clustern in einer VPC finden Sie unter Amazon VPC und Amazon Aurora.
- Verwenden Sie AWS Identity and Access Management (IAM) -Richtlinien, um Berechtigungen zuzuweisen, die festlegen, wer Aurora Aurora-Ressourcen verwalten darf. Beispielsweise können Sie IAM verwenden, um zu bestimmen, wer DB- Cluster erstellen, beschreiben, ändern und löschen, Ressourcen taggen oder Sicherheitsgruppen ändern darf.

IAM-Beispielrichtlinien finden Sie unter Beispiele für identitätsbasierte Amazon-Aurora-Richtlinien.

- Verwenden Sie Sicherheitsgruppen, um zu steuern, welche IP-Adressen oder EC2 Amazon-Instances eine Verbindung zu Ihren Datenbanken in einem herstellen können. Wenn Sie zum ersten Mal einen DB-Cluster erstellen, verhindert die zugehörige Firewall jeglichen Datenbankzugriff, außer den Zugriff über Regeln, die in einer zugehörigen Sicherheitsgruppe festgelegt sind.
- Verwenden Sie Secure Sockets Layer (SSL)- oder Transport Layer Security (TLS)-Verbindungen mit DB- Clustern, auf denen Aurora MySQL oder Aurora PostgreSQL ausgeführt wird. Weitere Informationen zur Verwendung SSL/TLS mit einem DB-Cluster finden Sie unterWird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.
- Verwenden Sie die Amazon-Aurora-Verschlüsselung, um Ihre DB-Cluster und Snapshots im Ruhezustand zu sichern. Die Amazon-Aurora-Verschlüsselung verwendet den branchenüblichen AES-256-Verschlüsselungsalgorithmus, um Ihre Daten auf dem Server zu verschlüsseln, der Ihren DB-Cluster hostet. Weitere Informationen finden Sie unter Verschlüsseln von Amazon Aurora-Ressourcen.
- Verwenden Sie die Sicherheitsfeatures Ihrer DB-Engine, um zu steuern, wer sich bei Ihren Datenbanken auf DB- Clustern anmelden kann. Diese Funktionen arbeiten genauso, als würden sich die Datenbanken in Ihrem lokalen Netzwerk befinden.

  Hinweise zur Sicherheit mit Aurora MySQL finden Sie unterSicherheit in Amazon Aurora MySQL. Hinweise zur Sicherheit mit Aurora PostgreSQL finden Sie unter. Sicherheit in Amazon Aurora PostgreSQL

Aurora ist Teil des verwalteten Datenbankservices Amazon Relational Database Service (Amazon RDS). Amazon RDS ist ein Webservice, der das Einrichten, Betreiben und Skalieren einer relationalen Datenbank in der Cloud vereinfacht. Wenn Sie noch keine Erfahrung mit Amazon RDS haben, beachten Sie die Informationen im Amazon-RDS-Benutzerhandbuch.

Aurora umfasst ein hochleistungsfähiges Speichersubsystem. Die mit MySQL und PostgreSQL kompatiblen Datenbank-Engines werden angepasst, um diesen schnellen verteilten Speicher zu nutzen. Aurora automatisiert und standardisiert auch das Clustering und die Replikation von Datenbanken, die normalerweise zu den schwierigsten Aspekten der Datenbankkonfiguration und -verwaltung gehören.

Sowohl für Amazon RDS als auch für Aurora können Sie programmgesteuert auf die RDS-API zugreifen, und Sie können die verwenden, AWS CLI um interaktiv auf die RDS-API zuzugreifen.

Einige RDS-API-Operationen und AWS CLI -Befehle gelten für Amazon RDS und Aurora, während andere entweder für Amazon RDS oder für Aurora gelten. Weitere Informationen zu RDS-API-Operationen finden Sie in der Amazon-RDS-API-Referenz. Weitere Informationen zu finden Sie in der AWS CLIAWS Command Line Interface Referenz für Amazon RDS.

> **Note**
>
> Sie müssen die Sicherheit nur für Ihre Anwendungsfälle konfigurieren. Sie müssen den Sicherheitszugriff für Prozesse, die Amazon Aurora verwaltet, nicht konfigurieren. Dazu gehört das Erstellen von Backups, automatischem Failover und anderen Prozessen.

Weitere Informationen zum Verwalten des Zugriffs auf Amazon-Aurora-Ressourcen und Ihre Datenbanken auf DB-Clustern finden Sie in den folgenden Themen.

Themen

- Datenbankauthentifizierung mit Amazon Aurora
- Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager
- Datenschutz in Amazon RDS
- Identity and Access Management für Amazon Aurora
- Protokollieren und Überwachen in Amazon Aurora
- Compliance-Validierung für Amazon Aurora
- Ausfallsicherheit in Amazon Aurora
- Sicherheit der Infrastruktur in Amazon Aurora
- Amazon-RDS-API und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)
- Bewährte Methoden für die Sicherheit in Amazon Aurora
- Zugriffskontrolle mit Sicherheitsgruppen
- Berechtigungen von Hauptbenutzerkonten
- Verwenden von serviceverknüpften Rollen für Amazon Aurora
- Amazon VPC und Amazon Aurora

Datenbankauthentifizierung mit Amazon Aurora

Amazon Aurora unterstützt verschiedene Möglichkeiten, Datenbankbenutzer zu authentifizieren.

Die Passwort-Authentifizierung ist standardmäßig für alle DB-Cluster verfügbar. Für Aurora MySQL und Aurora PostgreSQL können Sie auch entweder IAM-Datenbank-Authentifizierung oder Kerberos-Authentifizierung oder beides für denselben DB-Cluster hinzufügen.

Passwort-, Kerberos- und IAM-Datenbank-Authentifizierung verwenden verschiedene Methoden zur Authentifizierung bei der Datenbank. Daher kann sich ein bestimmter Benutzer mit nur einer einzigen Authentifizierungsmethode bei einer Datenbank anmelden.

Verwenden Sie für PostgreSQL nur eine der folgenden Rolleneinstellungen für einen Benutzer einer bestimmten Datenbank:

- Um die IAM-Datenbank-Authentifizierung zu verwenden, weisen Sie die `rds_iam`-Rolle dem Benutzer zu.
- Um Kerberos-Authentifizierung zu verwenden, weisen Sie die `rds_ad`-Rolle dem Benutzer zu.
- Um die Passwort-Authentifizierung zu verwenden, weisen Sie keine der beiden `rds_iam`- oder `rds_ad`- Rollen dem Benutzer zu.

Weisen Sie nicht beide Rollen `rds_iam` und `rds_ad` einem Benutzer einer PostgreSQL-Datenbank zu, weder direkt noch indirekt durch verschachtelten gewährtem Zugriff. Wenn die `rds_iam`-Rolle dem Hauptbenutzer hinzugefügt wird, hat die IAM-Authentifizierung Vorrang vor der Passwort-Authentifizierung, so dass sich der Hauptbenutzer als IAM-Benutzer anmelden muss.

> **Important**
>
> Wir empfehlen Ihnen, den Hauptbenutzer nicht direkt in Ihren Anwendungen zu verwenden. Bleiben Sie stattdessen bei der bewährten Methode, einen Datenbankbenutzer zu verwenden, der mit den Mindestberechtigungen erstellt wurde, die für Ihre Anwendung erforderlich sind.

Themen

- Passwortauthentifizierung
- IAM-Datenbankauthentifizierung
- Kerberos-Authentifizierung

Passwortauthentifizierung

Mit der Passwortauthentifizierung führt Ihre Datenbank die gesamte Verwaltung von Benutzerkonten durch. Sie erstellen Benutzer mit SQL-Anweisungen wie `CREATE USER`, mit der entsprechenden Klausel, die von der DB-Engine zum Angeben von Kennwörtern benötigt wird. In MySQL lautet die Anweisung beispielsweise `CREATE USER-`*`Name`* `IDENTIFIED BY-`*`Passwort`*, während die Anweisung in PostgreSQL `CREATE USER-`*`Name`* `WITH PASSWORD-`*`Passwort`* ist.

Mit der Passwortauthentifizierung steuert und authentifiziert Ihre Datenbank Benutzerkonten. Wenn eine DB-Engine über starke Passwortverwaltungsfunktionen verfügt, können diese die Sicherheit erhöhen. Die Datenbankauthentifizierung ist möglicherweise einfacher mit der Passwortauthentifizierung zu verwalten, wenn Sie kleine Benutzergemeinschaften haben. Da in diesem Fall Klartext-Passwörter generiert werden, kann die Integration mit AWS Secrets Manager die Sicherheit erhöhen.

Weitere Informationen zur Verwendung von Secrets Manager mit Amazon Aurora finden Sie unter Erstellen eines Basis-Secrets und Rotierende Secrets für unterstützte Amazon-RDS-Datenbanken im AWS Secrets Manager-Benutzerhandbuch. Informationen zum programmgesteuerten Abrufen Ihrer Secrets in Ihren benutzerdefinierten Anwendungen finden Sie unter Abrufen des Secret-Werts im AWS Secrets Manager-Benutzerhandbuch.

IAM-Datenbankauthentifizierung

Sie können sich mit der AWS Identity and Access Management (IAM)-Datenbankauthentifizierung bei DB-Clustern authentifizieren. Mit dieser Authentifizierungsmethode benötigen Sie kein Passwort, um eine Verbindung mit DB-Clustern herzustellen. Stattdessen verwenden Sie ein Authentifizierungstoken.

Weitere Informationen zur IAM-Datenbankauthentifizierung, einschließlich Informationen zur Verfügbarkeit bestimmter DB-Engines, finden Sie unter IAM-Datenbankauthentifizierung .

Kerberos-Authentifizierung

Amazon Aurora unterstützt die externe Authentifizierung von Datenbankbenutzern über Kerberos und Microsoft Active Directory. Kerberos ist ein Netzwerk-Authentifizierungsprotokoll, das Tickets und symmetrische Schlüsselkryptographie verwendet, um die Notwendigkeit der Übertragung von Passwörtern über das Netzwerk zu vermeiden. Kerberos wurde in Active Directory integriert und wurde entwickelt, um Benutzer gegenüber Netzwerkressourcen wie Datenbanken zu authentifizieren.

Die Amazon-Aurora-Unterstützung für Kerberos und Active Directory bietet die Vorteile des Single Sign-Ons und der zentralisierten Authentifizierung von Datenbankbenutzern. Sie können Ihre Benutzeranmeldeinformationen in Active Directory speichern. Active Directory bietet einen zentralen Ort für die Speicherung und Verwaltung von Anmeldeinformationen für mehrere DB-Cluster.

Um Anmeldeinformationen aus Ihrem selbstverwalteten Active Directory zu verwenden, müssen Sie für Microsoft Active Directory eine Vertrauensbeziehung zu dem Directory Service einrichten, mit dem der DB-Cluster verbunden ist.

Aurora PostgreSQL und Aurora MySQL unterstützen unidirektionale und bidirektionale Gesamtstruktur-Vertrauensstellungen mit strukturweiter Authentifizierung oder selektiver Authentifizierung.

In einigen Szenarien können Sie die Kerberos-Authentifizierung über eine externe Vertrauensstellung konfigurieren. Dazu muss Ihr selbstverwaltetes Active Directory über zusätzliche Einstellungen verfügen. Dies beinhaltet, ist aber nicht beschränkt auf Kerberos Forest Search Order.

Derzeit unterstützt Aurora die Kerberos-Authentifizierung für DB-Cluster von Aurora MySQL und Aurora PostgreSQL. Weitere Informationen erhalten Sie unter Verwenden der Kerberos-Authentifizierung für Aurora MySQL und Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL.

Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager

Amazon Aurora lässt sich in Secrets Manager integrieren, um Hauptbenutzerpasswörter für Ihre DB--Cluster zu verwalten.

Themen

- Verfügbarkeit von Regionen und Versionen
- Einschränkungen für die Integration von Secrets Manager in Amazon Aurora
- Überblick über die Verwaltung von Masterbenutzerkennwörtern mit AWS Secrets Manager
- Vorteile der Verwaltung von Hauptbenutzerpasswörtern mit Secrets Manager
- Erforderliche Berechtigungen für die Integration von Secrets Manager
- Durchsetzung der Verwaltung des Masterbenutzerkennworts durch Aurora in AWS Secrets Manager
- Verwaltung des Hauptbenutzerpassworts für einen DB-Cluster mit Secrets Manager
- Rotieren des Hauptbenutzerpasswort-Secrets für einen DB-Cluster
- Anzeigen der Details zu einem Secret für einen DB-Cluster

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur Verfügbarkeit von Versionen und Regionen für die Integration von Secrets Manager in Amazon Aurora finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für die Integration von Secrets Manager.

Einschränkungen für die Integration von Secrets Manager in Amazon Aurora

Die Verwaltung von Hauptpasswörtern mit Secrets Manager wird für die folgenden Funktionen nicht unterstützt:

- Amazon Blue/Green RDS-Bereitstellungen
- DB-Cluster, die Teil einer globalen Aurora-Datenbank sind
- Aurora Serverless v1-DB-Cluster

- Regionsübergreifende Lesereplikate
- Externe Replikation von Binärprotokollen

Überblick über die Verwaltung von Masterbenutzerkennwörtern mit AWS Secrets Manager

Mit AWS Secrets Manager können Sie hartcodierte Anmeldeinformationen in Ihrem Code, einschließlich Datenbankkennwörtern, durch einen API-Aufruf an Secrets Manager ersetzen, um das Geheimnis programmgesteuert abzurufen. Weitere Informationen zu Secrets Manager finden Sie im Benutzerhandbuch für AWS Secrets Manager.

Wenn Sie Datenbankgeheimnisse in Secrets Manager speichern, AWS-Konto fallen Gebühren an. Informationen zu den Preisen erhalten Sie unter AWS Secrets Manager Preise.

Sie können angeben, dass Aurora das Hauptbenutzerpasswort in Secrets Manager für einen DB-Cluster von Amazon Aurora verwaltet, wenn Sie eine der folgenden Operationen ausführen:

- Erstellen des -DB-Clusters
- Ändern eines -DB-Clusters
- Wiederherstellen eines DB-Clusters aus Amazon S3 (nur Aurora MySQL)

Wenn Sie angeben, dass Aurora das Hauptbenutzerpasswort in Secrets Manager verwaltet, generiert Aurora das Passwort und speichert es in Secrets Manager. Sie können direkt mit dem Secret interagieren, um die Anmeldeinformationen für den Hauptbenutzer abzurufen. Sie können auch einen vom Kunden verwalteten Schlüssel angeben, um das Secret zu verschlüsseln, oder den KMS-Schlüssel verwenden, der von Secrets Manager bereitgestellt wird.

Aurora verwaltet die Einstellungen für das Secret und rotiert das Secret standardmäßig alle sieben Tage. Sie können einige Einstellungen ändern, wie zum Beispiel den Rotationsplan. Wenn Sie einen DB-Cluster löschen, der ein Secret in Secrets Manager verwaltet, werden das Secret und die zugehörigen Metadaten ebenfalls gelöscht.

Um eine Verbindung mit mit den Anmeldeinformationen in einem Secret herzustellen, können Sie das Secret von Secrets Manager abrufen. Weitere Informationen finden Sie im Benutzerhandbuch unter Abrufen von Geheimnissen aus AWS Secrets Manager und Herstellen einer Verbindung zu einer SQL-Datenbank mit Anmeldeinformationen in einem AWS Secrets Manager Geheimnis herstellen.AWS Secrets Manager

Vorteile der Verwaltung von Hauptbenutzerpasswörtern mit Secrets Manager

Die Verwaltung von Aurora-Hauptbenutzerpasswörtern mit Secrets Manager bietet die folgenden Vorteile:

- Aurora generiert automatisch Datenbankanmeldeinformationen.
- Aurora speichert und verwaltet automatisch Datenbankanmeldedaten in AWS Secrets Manager.
- Aurora rotiert die Datenbankanmeldeinformationen regelmäßig, ohne dass Anwendungsänderungen erforderlich sind.
- Secrets Manager schützt Datenbankanmeldeinformationen vor menschlichem Zugriff und der Klartextansicht.
- Secrets Manager ermöglicht das Abrufen von Datenbankanmeldeinformationen in Secrets für Datenbankverbindungen.
- Secrets Manager ermöglicht eine detaillierte Steuerung des Zugriffs auf Datenbankanmeldeinformationen in Secrets mithilfe von IAM.
- Optional können Sie die Datenbankverschlüsselung von der Anmeldeinformationsverschlüsselung mit unterschiedlichen KMS-Schlüsseln trennen.
- Sie können die manuelle Verwaltung und Rotation der Datenbankanmeldeinformationen vermeiden.
- Sie können Datenbankanmeldedaten einfach mit AWS CloudTrail Amazon überwachen CloudWatch.

Weitere Informationen zu den Vorteilen von Secrets Manager finden Sie im Benutzerhandbuch für AWS Secrets Manager.

Erforderliche Berechtigungen für die Integration von Secrets Manager

Benutzer müssen über die erforderlichen Berechtigungen verfügen, um Operationen im Zusammenhang mit der Integration von Secrets Manager auszuführen. Sie können IAM-Richtlinien erstellen, die die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die benötigt werden. Sie können diese Richtlinien dann den IAM-Berechtigungssätzen oder -Rollen zuordnen, die diese Berechtigungen benötigen. Weitere Informationen finden Sie unter Identity and Access Management für Amazon Aurora.

Für Erstellungs-, Änderungs- oder Wiederherstellungsoperationen muss der Benutzer, der angibt, dass Aurora das Hauptbenutzerpasswort in Secrets Manager verwaltet, über entsprechende Berechtigungen für folgende Operationen verfügen:

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

Die Berechtigung `kms:DescribeKey` ist erforderlich, um auf Ihren vom Kunden verwalteten Schlüssel für die `MasterUserSecretKmsKeyId` zuzugreifen und `aws/secretsmanager` zu beschreiben.

Für Erstellungs-, Änderungs- oder Wiederherstellungsoperationen muss der Benutzer, der den benutzerdefinierten Schlüssel zum Entschlüsseln des Secrets in Secrets Manager angibt, über entsprechende Berechtigungen für folgende Operationen verfügen:

- `kms:Decrypt`
- `kms:GenerateDataKey`
- `kms:CreateGrant`

Für Änderungsoperationen muss der Benutzer, der das Hauptbenutzerpasswort in Secrets Manager rotiert, über entsprechende Berechtigungen für die folgende Operation verfügen:

- `secretsmanager:RotateSecret`

Durchsetzung der Verwaltung des Masterbenutzerkennworts durch Aurora in AWS Secrets Manager

Sie können IAM-Bedingungsschlüssel verwenden, um die Aurora-Verwaltung des Hauptbenutzerpassworts in AWS Secrets Manager zu erzwingen. Die folgende Richtlinie erlaubt Benutzern nicht, DB-Instances oder DB-Cluster zu erstellen es sei denn, das Hauptbenutzerpasswort wird von Aurora in Secrets Manager verwaltet.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

> **Note**
>
> Diese Richtlinie erzwingt die Passwortverwaltung bereits AWS Secrets Manager bei der Erstellung. Sie können die Secrets-Manager-Integration jedoch nach wie vor deaktivieren und ein Hauptpasswort manuell festlegen, indem Sie den Cluster ändern.
> Um dies zu verhindern, nehmen Sie `rds:ModifyDBInstance`, `rds:ModifyDBCluster` in den Aktionsblock der Richtlinie auf. Beachten Sie, dass der Benutzer dadurch keine weiteren Änderungen an vorhandenen Clustern vornehmen kann, für die die Secrets-Manager-Integration nicht aktiviert ist.

Weitere Informationen zum Verwenden der Bedingungsschlüssels in IAM-Richtlinien finden Sie unter Richtlinien-Bedingungsschlüssel für Aurora und Beispielrichtlinien: Verwenden von Bedingungsschlüsseln.

Verwaltung des Hauptbenutzerpassworts für einen DB-Cluster mit Secrets Manager

Sie können die Aurora-Verwaltung des Hauptbenutzerpassworts in Secrets Manager konfigurieren, wenn Sie die folgenden Aktionen ausführen:

- Erstellen eines Amazon Aurora-DB Clusters
- Ändern eines Amazon Aurora-DB-Clusters

- Migrieren von Daten aus einer externen MySQL-Datenbank zu einem Amazon-Aurora-MySQL-DB-Cluster

Sie können die RDS-Konsole AWS CLI, die oder die RDS-API verwenden, um diese Aktionen auszuführen.

Konsole

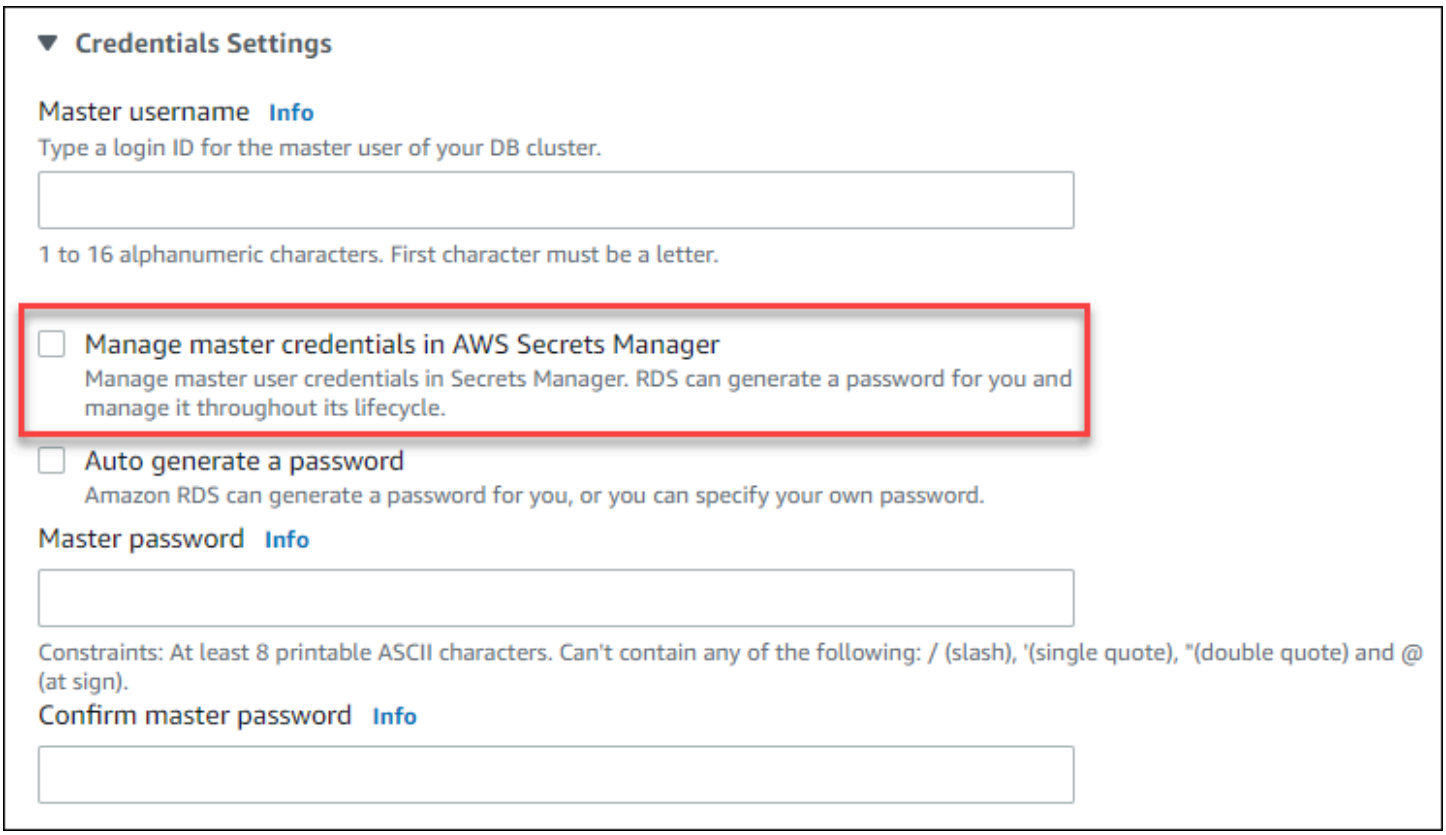
Folgen Sie den Anweisungen zum Erstellen oder Ändern eines DB-Clusters mit der RDS-Konsole:

- Erstellen eines DB-Clusters
- Ändern einer DB-Instance in einem DB-Cluster

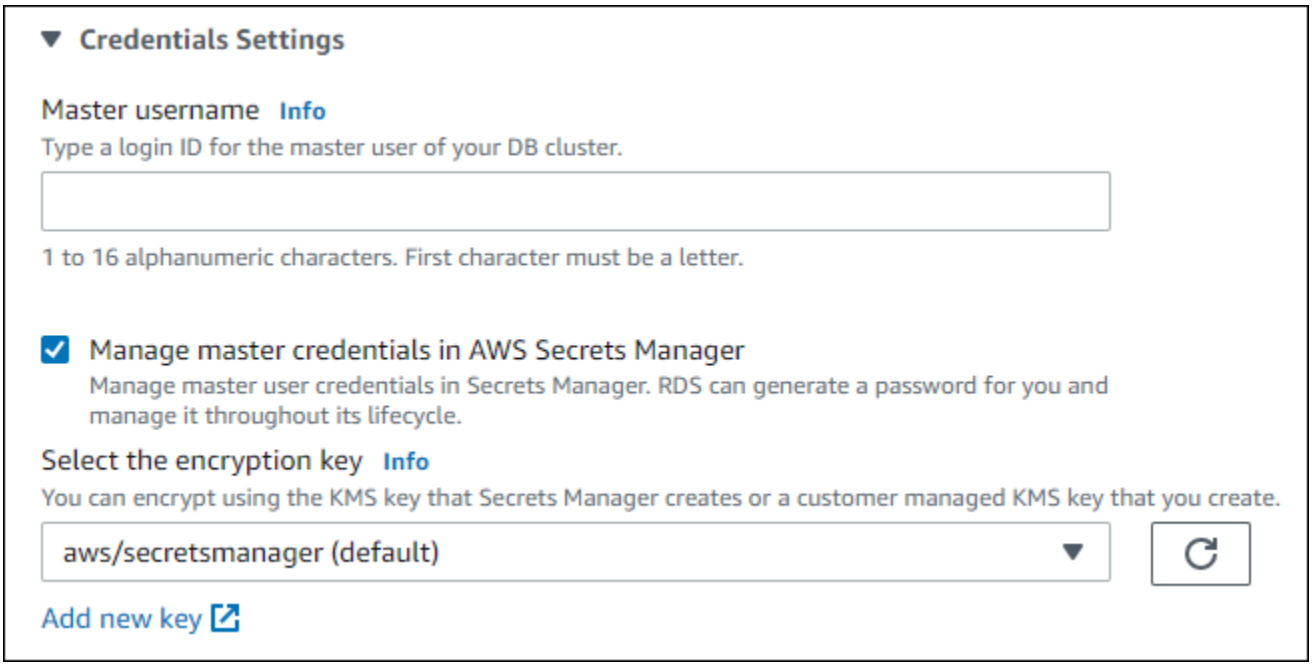
  In der RDS-Konsole können Sie jede DB-Instance ändern, um die Verwaltungseinstellungen für das Hauptbenutzerpasswort für den gesamten DB-Cluster anzugeben.
- Wiederherstellen eines Amazon Aurora-MySQL-DB-Clusters aus einem Amazon S3-Bucket

Wenn Sie die RDS-Konsole verwenden, um eine dieser Operationen auszuführen, können Sie angeben, dass das Hauptbenutzerpasswort von Aurora in Secrets Manager verwaltet wird. Wählen Sie dazu beim Erstellen oder Wiederherstellen eines DB-Clusters Hauptanmeldeinformationen in AWS Secrets Manager verwalten unter Anmeldeinformationseinstellungen aus. Wenn Sie einen DB-Cluster ändern, wählen Sie Hauptanmeldeinformationen in AWS Secrets Manager verwalten unter Einstellungen aus.

Die folgende Abbildung zeigt ein Beispiel für die Einstellung Hauptanmeldeinformationen in AWS Secrets Manager verwalten beim Erstellen oder Wiederherstellen eines DB-Clusters.



Wenn Sie diese Option auswählen, generiert Aurora das Hauptbenutzerpasswort und verwaltet es während seines gesamten Lebenszyklus in Secrets Manager.



Sie können wählen, ob Sie das Secret mit einem von Secrets Manager bereitgestellten KMS-Schlüssel oder mit einem von Ihnen erstellten kundenverwalteten Schlüssel verschlüsseln möchten.

Nachdem Aurora die Datenbankanmeldeinformationen für einen DB-Cluster verwaltet hat, können Sie den KMS-Schlüssel, der zum Verschlüsseln des Secrets verwendet wird, nicht mehr ändern.

Sie können andere Einstellungen auswählen, die Ihren Anforderungen entsprechen.

Weitere Informationen zu den verfügbaren Einstellungen beim Erstellen eines DB-Clusters finden Sie unter Einstellungen für Aurora-DB-Cluster. Weitere Informationen zu den verfügbaren Einstellungen beim Ändern eines DB-Clusters finden Sie unter Einstellungen für Amazon Aurora.

AWS CLI

Wenn Sie angeben möchten, dass Aurora das Hauptbenutzerpasswort in Secrets Manager verwalten soll, geben Sie die `--manage-master-user-password`-Option in einem der folgenden Befehle an:

- create-db-cluster
- modify-db-cluster
- restore-db-cluster-from-s3

Wenn Sie die `--manage-master-user-password`-Option angeben, generiert Aurora das Hauptbenutzerpasswort und verwaltet es während seines gesamten Lebenszyklus in Secrets Manager.

Sie können auch einen kundenverwalteten Schlüssel angeben, um das Secret zu verschlüsseln, oder den KMS-Standardschlüssel verwenden, der von Secrets Manager bereitgestellt wird. Verwenden Sie die `--master-user-secret-kms-key-id`-Option, um einen kundenverwalteten Schlüssel anzugeben. Die AWS KMS-Schlüssel-ID ist der Schlüssel-ARN, die Schlüssel-ID, der Alias-ARN oder der Aliasname für den KMS-Schlüssel. Um einen KMS-Schlüssel in einem anderen zu verwenden AWS-Konto, geben Sie den Schlüssel-ARN oder den Alias-ARN an. Nachdem Aurora die Datenbankanmeldeinformationen für einen DB-Cluster verwaltet hat, können Sie den KMS-Schlüssel, der zum Verschlüsseln des Secrets verwendet wird, nicht mehr ändern.

Sie können andere Einstellungen auswählen, die Ihren Anforderungen entsprechen.

Weitere Informationen zu den verfügbaren Einstellungen beim Erstellen eines DB-Clusters finden Sie unter Einstellungen für Aurora-DB-Cluster. Weitere Informationen zu den verfügbaren Einstellungen beim Ändern eines DB-Clusters finden Sie unter Einstellungen für Amazon Aurora.

In diesem Beispiel wird ein -DB-Cluster erstellt und angegeben, dass Aurora das Passwort in Secrets Manager verwaltet. Das Secret wird mit dem KMS-Schlüssel verschlüsselt, der von Secrets Manager bereitgestellt wird.

Example

Für Linux, macOS oder Unix:

```
aws rds create-db-cluster \
     --db-cluster-identifier sample-cluster \
     --engine aurora-mysql \
     --engine-version 8.0 \
     --master-username admin \
     --manage-master-user-password
```

Für Windows:

```
aws rds create-db-cluster ^
     --db-cluster-identifier sample-cluster ^
     --engine aurora-mysql ^
     --engine-version 8.0 ^
     --master-username admin ^
     --manage-master-user-password
```

RDS-API

Wenn Sie angeben möchten, dass Aurora das Hauptbenutzerpasswort in Secrets Manager verwaltet, legen Sie den `ManageMasterUserPassword`-Parameter in einer der folgenden Operationen auf `true` fest:

- CreateDBCluster
- Modifizieren SieDBCluster
- DBClusterAus S3 wiederherstellen

Wenn Sie den `ManageMasterUserPassword`-Parameter in einer dieser Operationen auf `true` festlegen, generiert Aurora das Hauptbenutzerpasswort und verwaltet es während seines gesamten Lebenszyklus in Secrets Manager.

Sie können auch einen kundenverwalteten Schlüssel angeben, um das Secret zu verschlüsseln, oder den KMS-Standardschlüssel verwenden, der von Secrets Manager bereitgestellt wird. Verwenden Sie den `MasterUserSecretKmsKeyId`-Parameter, um einen kundenverwalteten Schlüssel anzugeben. Die AWS KMS-Schlüssel-ID ist der Schlüssel-ARN, die Schlüssel-ID, der Alias-ARN oder der Aliasname für den KMS-Schlüssel. Geben Sie den Schlüssel-ARN oder Alias-ARN an,

um einen KMS-Schlüssel in einem anderen AWS-Konto zu verwenden. Nachdem Aurora die Datenbankanmeldeinformationen für einen DB-Cluster verwaltet hat, können Sie den KMS-Schlüssel, der zum Verschlüsseln des Secrets verwendet wird, nicht mehr ändern.

Rotieren des Hauptbenutzerpasswort-Secrets für einen DB-Cluster

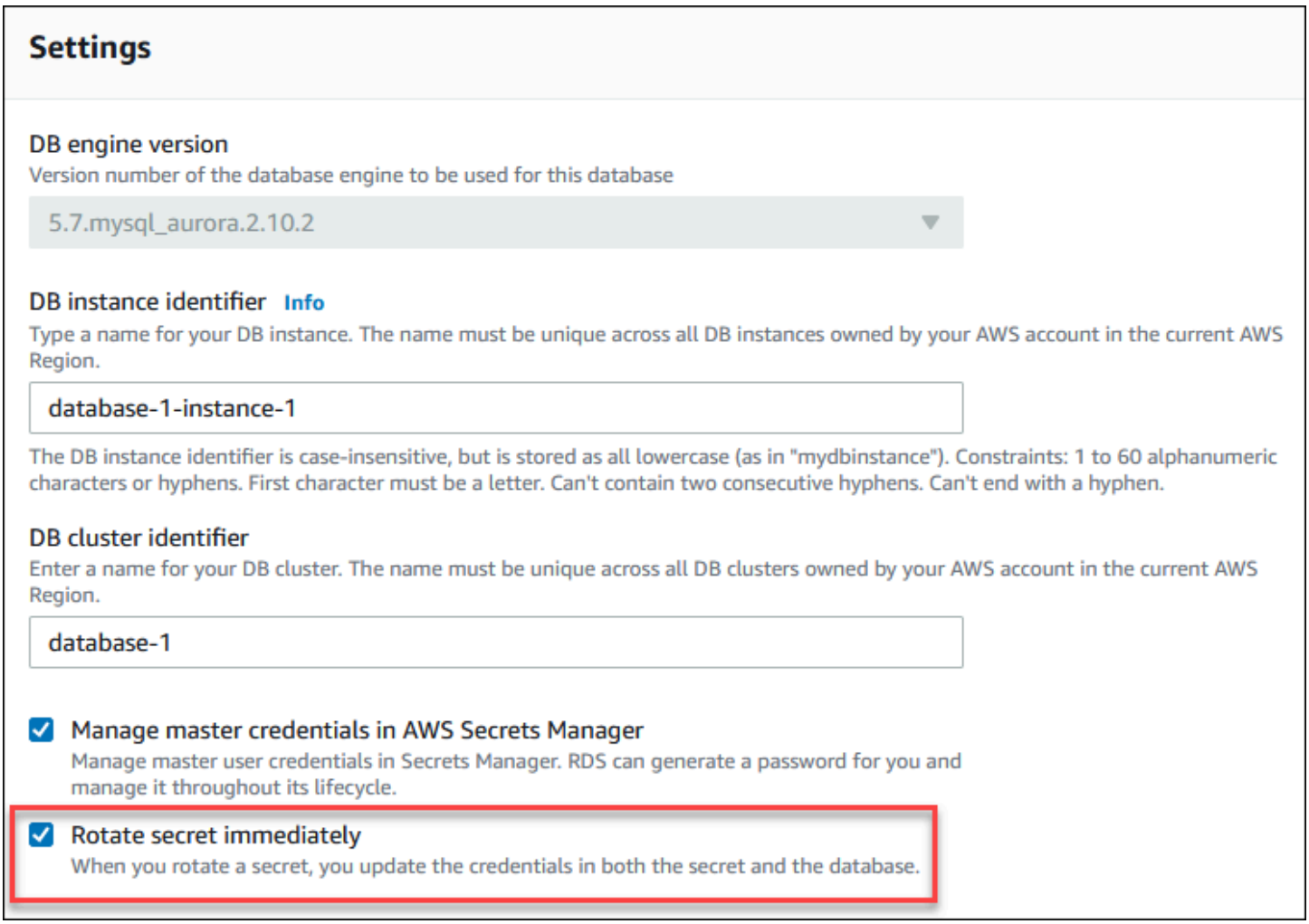
Wenn Aurora ein Hauptbenutzerpasswort-Secret rotiert, generiert Secrets Manager eine neue Secret-Version für das vorhandene Secret. Die neue Secret-Version enthält das neue Hauptbenutzerpasswort. Aurora ändert das Hauptbenutzerpasswort für den -DB-Cluster so, dass es mit dem Passwort für die neue Secret-Version übereinstimmt.

Sie können ein Secret sofort rotieren, anstatt auf eine geplante Rotation zu warten. Wenn Sie das Hauptbenutzerpasswort-Secret in Secrets Manager rotieren möchten, ändern Sie den DB-Cluster. Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Sie können ein geheimes Masterbenutzerkennwort sofort mit der RDS-Konsole AWS CLI, der oder der RDS-API rotieren. Das neue Passwort ist immer 28 Zeichen lang und enthält mindestens einen Groß- und Kleinbuchstaben, eine Zahl und ein Interpunktionszeichen.

Konsole

Wenn Sie das Hauptbenutzerpasswort-Secret mithilfe der RDS-Konsole rotieren möchten, ändern Sie den DB-Cluster und wählen Sie die Option Rotate secret immediately (Sofortige Secret-Drehung) unter Settings (Einstellungen) aus.



Folgen Sie den Anweisungen zum Ändern eines DB-Clusters mit der RDS-Konsole in Ändern des DB-Clusters über die Konsole, die CLI und die API. Sie müssen auf der Bestätigungsseite die Option Apply immediately (Sofort anwenden) auswählen.

AWS CLI

Verwenden Sie den modify-db-clusterBefehl und geben Sie die `--rotate-master-user-password` Option an AWS CLI, um ein geheimes Masterbenutzer-Passwort mithilfe von zu wechseln. Sie müssen die `--apply-immediately`-Option angeben, wenn Sie das Hauptpasswort rotieren.

In diesem Beispiel wird ein Hauptbenutzerpasswort-Secret rotiert.

Example

Für Linux, macOS oder Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --rotate-master-user-password \
    --apply-immediately
```

Für Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

RDS-API

Sie können ein geheimes Masterbenutzerkennwort rotieren, indem Sie den DBCluster Vorgang Ändern verwenden und den `RotateMasterUserPassword` Parameter auf setzen`true`. Sie müssen den `ApplyImmediately`-Parameter auf `true` festlegen, wenn Sie das Hauptpasswort rotieren.

Anzeigen der Details zu einem Secret für einen DB-Cluster

Sie können Ihre Secrets mit der Konsole (https://console.aws.amazon.com/secretsmanager/) oder dem AWS CLI (get-secret-valueSecrets Manager Manager-Befehl) abrufen.

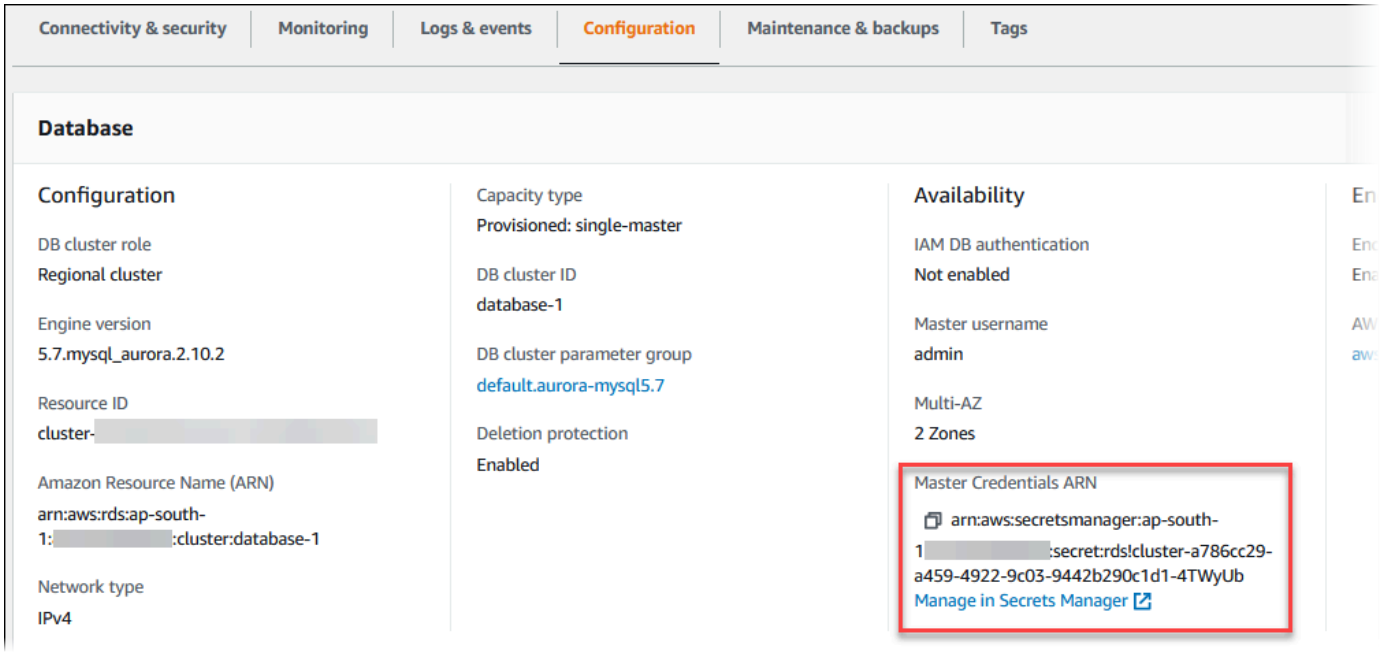
Sie finden den Amazon-Ressourcennamen (ARN) eines von Aurora verwalteten Secrets im Secrets Manager mit der RDS-Konsole AWS CLI, der oder der RDS-API.

Konsole

So zeigen Sie die Details zu einem von Aurora verwalteten Secret in Secrets Manager an

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie den Namen des DB-Clusters aus, um dessen Details anzuzeigen.
4. Wählen Sie die Registerkarte Konfiguration aus.

   Unter Master Credentials ARN (ARN der Hauptanmeldeinformationen) können Sie den geheimen ARN einsehen.



Sie können dem Link Manage in Secrets Manager (In Secrets Manager verwalten) folgen, um das Secret in der Secrets-Manager-Konsole anzuzeigen und zu verwalten.

AWS CLI

Sie können den AWS CLI describe-db-clustersBefehl RDS verwenden, um die folgenden Informationen zu einem Secret zu finden, das von Aurora in Secrets Manager verwaltet wird:

- `SecretArn` – Der ARN des Secrets
- `SecretStatus` – Der Status des Secrets

  Mögliche Werte für den Status sind u. a. folgende:

  - `creating` – Das Secret wird erstellt.
  - `active` – Das Secret ist für den normalen Gebrauch und die Rotation verfügbar.
  - `rotating` – Das Secret wird rotiert.
  - `impaired` – Das Secret kann für den Zugriff auf Datenbankanmeldeinformationen verwendet werden, es kann jedoch nicht rotiert werden. Ein Secret kann diesen Status haben, wenn beispielsweise die Berechtigungen geändert werden, sodass RDS nicht mehr auf das Secret oder den KMS-Schlüssel für das Secret zugreifen kann.

    Wenn ein Secret diesen Status hat, können Sie die Bedingung korrigieren, die den Status verursacht hat. Wenn Sie die Bedingung korrigieren, die den Status verursacht hat, behält

der Status bis zur nächsten Rotation den Wert `impaired`. Alternativ können Sie den DB-Cluster ändern, um die automatische Verwaltung von Datenbankanmeldeinformationen zu deaktivieren, und dann den DB-Cluster erneut ändern, um die automatische Verwaltung von Datenbankanmeldeinformationen zu aktivieren. Verwenden Sie die `--manage-master-user-password` Option im modify-db-clusterBefehl, um den DB-Cluster zu ändern.

- `KmsKeyId` – Der ARN des KMS-Schlüssels, der verwendet wird, um das Secret zu verschlüsseln

Geben Sie die `--db-cluster-identifier`-Option an, um die Ausgabe für einen bestimmten DB-Cluster anzuzeigen. Dieses Beispiel zeigt die Ausgabe für ein Secret, das von einem DB-Cluster verwendet wird.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

Das folgende Beispiel zeigt die Ausgabe für ein Secret:

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

Wenn Sie über den geheimen ARN verfügen, können Sie mit dem get-secret-valueSecrets Manager-CLI-Befehl Details zum Secret Manager anzeigen.

Dieses Beispiel zeigt die Details für das Secret in der vorherigen Beispielausgabe.

Example

Für Linux, macOS oder Unix:

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

Für Windows:

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

RDS-API

Sie können den ARN, den Status und den KMS-Schlüssel für ein von Aurora verwaltetes Geheimnis im Secrets Manager anzeigen, indem Sie den Vorgang DBClustersDescribe RDS verwenden und den `DBClusterIdentifier` Parameter auf eine DB-Cluster-ID setzen. Details zum Secret sind in der Ausgabe enthalten.

Wenn Sie über den geheimen ARN verfügen, können Sie mithilfe des GetSecretValueSecrets Manager-Vorgangs Details zu dem Secret anzeigen.

Datenschutz in Amazon RDS

Das AWS Modell der geteilten Verantwortung wird auch auf den Datenschutz in Amazon Relational Database Service angewendet. Wie in diesem Modell beschrieben, ist AWS verantwortlich für den Schutz der globalen Infrastruktur, in der die gesamte AWS Cloud ausgeführt wird. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS-Modell der geteilten Verantwortung und in der DSGVO im AWS-Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, AWS-Konto-Anmeldeinformationen zu schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einzurichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Verwenden Sie SSL/TLS für die Kommunikation mit AWS-Ressourcen. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit AWS CloudTrail ein. Informationen zur Verwendung von CloudTrail-Trails zur Erfassung von AWS-Aktivitäten finden Sie unter Arbeiten mit CloudTrail-Trails im AWS CloudTrail-Benutzerhandbuch.
- Verwenden Sie AWS-Verschlüsselungslösungen zusammen mit allen Standardsicherheitskontrollen in AWS-Services.

- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff auf AWS über eine Befehlszeilenschnittstelle oder über eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit Amazon RDS oder anderen AWS-Services über die Konsole, API, AWS CLI oder AWS SDKs arbeiten. Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Themen

- Datenschutz durch Verschlüsselung
- Richtlinie für den Datenverkehr zwischen Netzwerken

Datenschutz durch Verschlüsselung

Sie können die Verschlüsselung für Datenbankressourcen aktivieren. Sie können auch Verbindungen zu DB-Clustern verschlüsseln.

Themen

- Verschlüsseln von Amazon Aurora-Ressourcen
- AWS KMS key-Verwaltung
- Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln
- Rotieren Ihrer SSL/TLS-Zertifikate

Verschlüsseln von Amazon Aurora-Ressourcen

Amazon Aurora kann Ihre Amazon Aurora-DB-Cluster verschlüsseln. Daten, die im Ruhezustand verschlüsselt werden, umfassen den zugehörigen Speicherplatz von DB-Clustern sowie deren automatisierte Backups, Lesereplikate und Snapshots.

Amazon Aurora-verschlüsselte DB-Cluster verwenden den standardmäßig in der Branche verwendeten AES-256-Verschlüsselungsalgorithmus, um Ihre Daten auf dem Server zu verschlüsseln, der Ihre Amazon Aurora-DB-Cluster hostet. Sobald Sie die Daten verschlüsselt haben, übernimmt Amazon Aurora die Authentifizierung des Zugriffs und die Entschlüsselung Ihrer Daten auf transparente Art und Weise und mit minimaler Auswirkung auf die Leistung. Sie müssen Ihre Datenbank-Client-Anwendungen nicht ändern, um Verschlüsselung anzuwenden.

> **Note**
>
> Bei verschlüsselten und unverschlüsselten DB--Clustern werden Daten zwischen der Quelle und den Lesereplikaten verschlüsselt, auch wenn die Replikation AWS-regionsübergreifend erfolgt.

Themen

- Übersicht über die Verschlüsselung von Amazon Aurora-Ressourcen
- Verschlüsseln eines Amazon-Aurora-DB-Clusters
- Bestimmen, ob die Verschlüsselung für einen DB-Cluster aktiviert ist
- Verfügbarkeit der Amazon Aurora-Verschlüsselung
- Verschlüsselung während der Übertragung
- Einschränkungen von Amazon Aurora-verschlüsselten DB-Clustern

Übersicht über die Verschlüsselung von Amazon Aurora-Ressourcen

Amazon Aurora-verschlüsselte DB-Cluster bieten zusätzlichen Datenschutz, indem Sie Ihre Daten vor unautorisiertem Zugriff auf den zugehörigen Speicherplatz sichern. Sie können die Amazon Aurora-Verschlüsselung verwenden, um den Datenschutz für Ihre in der Cloud bereitgestellten Anwendungen zu erhöhen und die Compliance-Anforderungen bei der Verschlüsselung von Daten im Ruhezustand zu erfüllen. Bei einem mit Amazon Aurora verschlüsselten DB-Cluster werden alle DB-Instances, Protokolle, Backups und Snapshots verschlüsselt. Weitere Informationen zur Verfügbarkeit und zu den Einschränkungen der Verschlüsselung finden Sie unter Verfügbarkeit der Amazon Aurora-Verschlüsselung undEinschränkungen von Amazon Aurora-verschlüsselten DB-Clustern.

Amazon Aurora verwendet einen AWS Key Management Service-Schlüssel für die Verschlüsselung dieser Ressourcen. AWS KMS kombiniert sichere, hoch verfügbare Hard- und Software, um ein

System für die Schlüsselverwaltung bereitzustellen, das für die Cloud skaliert ist. Sie können eine Von AWS verwalteter Schlüssel verwenden, oder Sie können kundenverwaltete Schlüssel erstellen.

Wenn Sie einen verschlüsselten DB-Cluster erstellen, können Sie einen vom Kunden verwalteten Schlüssel oder den Von AWS verwalteter Schlüssel für Amazon Aurora zur Verschlüsselung Ihres DB-Clusters wählen. Wenn Sie den Schlüsselbezeichner für einen vom Kunden verwalteten Schlüssel nicht angeben, verwendet Amazon Aurora die Von AWS verwalteter Schlüssel für Ihren neuen DB-Cluster. Amazon Aurora erstellt eine Von AWS verwalteter Schlüssel für Amazon Aurora für Ihr AWS Konto. Ihr AWS Konto hat eine andere Von AWS verwalteter Schlüssel für Amazon Aurora für jede AWS Region.

Zur Verwaltung der vom Kunden verwalteten Schlüssel, die zum Ver- und Entschlüsseln Ihrer Amazon-Aurora-Ressourcen dienen, verwenden Sie die AWS Key Management Service (AWS KMS).

Mit AWS KMS können Sie vom Kunden verwaltete Schlüssel erstellen und die Richtlinien definieren, um die diese vom Kunden verwalteten Schlüssel zu steuern. AWS KMS unterstützt CloudTrail, sodass Sie die Verwendung von KMS-Schlüsseln überprüfen können, um sicherzustellen, dass die vom Kunden verwalteten Schlüssel ordnungsgemäß verwendet werden. Sie können Ihre vom Kunden verwalteten Schlüssel mit Amazon Aurora und unterstützten AWS Services wie Amazon S3, Amazon EBS und Amazon Redshift verwenden. Eine Liste der Services, die in AWS KMS integriert sind, finden Sie unter AWS-Service-Integration. Einige Überlegungen zur Verwendung von KMS-Schlüsseln:

- Sobald Sie eine verschlüsselte DB-Instance erstellt haben, können Sie den von dieser DB-Instance verwendeten KMS-Schlüssel nicht mehr ändern. Stellen Sie daher sicher, dass Sie Ihre KMS-Schlüsselanforderungen bestimmen, bevor Sie Ihre verschlüsselte DB-Instance erstellen.

  Wenn Sie den Verschlüsselungsschlüssel für Ihren DB-Cluster ändern müssen, erstellen Sie einen manuellen Snapshot Ihres Clusters und aktivieren Sie die Verschlüsselung beim Kopieren des Snapshots. Weitere Informationen finden Sie im re:Post-Knowledge-Artikel.
- Wenn Sie einen verschlüsselten Snapshot kopieren, können Sie zum Verschlüsseln des Ziel-Snapshots einen anderen KMS-Schlüssel verwenden als den, der zum Verschlüsseln des Quell-Snapshots verwendet wurde.
- Sie können einen Snapshot der verschlüsselt wurde, nicht mit dem Von AWS verwalteter Schlüssel der AWS Konten freigeben, die den Schnappschuss freigegeben haben.
- Jede DB-Instance im DB-Cluster wird mit demselben KMS-Schlüssel wie der DB-Cluster verschlüsselt.

- Lesereplikate von Amazon-Aurora-verschlüsselten Clustern können ebenfalls verschlüsselt werden.

Important

Amazon Aurora kann den Zugriff auf den KMS-Schlüssel für einen DB-Cluster verlieren, wenn Sie den KMS-Schlüssel deaktivieren. In diesen Fällen geht der verschlüsselte DB-Cluster in den `inaccessible-encryption-credentials-recoverable`-Zustand. Der DB-Cluster behält diesen Status sieben Tage lang bei, währenddessen die Instance angehalten wird. API-Aufrufe, die während dieser Zeit an den DB-Cluster getätigt wurden, sind möglicherweise nicht erfolgreich. Um den DB-Cluster wiederherzustellen, aktivieren Sie den KMS-Schlüssel und starten Sie diesen DB-Cluster neu. Aktivieren Sie den KMS-Schlüssel über die AWS-Managementkonsole, AWS CLI oder RDS-API. Starten Sie den DB-Cluster mit dem AWS CLI-Befehl start-db-cluster oder AWS-Managementkonsole neu.
Der Status `inaccessible-encryption-credentials-recoverable` gilt nur für DB-Cluster, die angehalten werden können. Dieser wiederherstellbare Status gilt nicht für Instances, die nicht angehalten werden können, wie Cluster mit regionsübergreifenden Lesereplikaten. Weitere Informationen finden Sie unter the section called “Einschränkungen”.
Wenn der DB-Cluster nicht innerhalb von sieben Tagen wiederhergestellt wird, wechselt er in den Terminalstatus `inaccessible-encryption-credentials`. In diesem Status ist der DB-Cluster nicht mehr nutzbar und kann nur über eine Sicherung wiederhergestellt werden. Wir empfehlen dringend, für verschlüsselte DB-Cluster immer Sicherungen zu aktivieren, um sich vor dem Verlust verschlüsselter Daten in Ihren Datenbanken zu schützen.
Bei der Erstellung eines DB-Clusters überprüft Aurora, ob der aufrufende Prinzipal Zugriff auf den KMS-Schlüssel hat, und generiert aus dem KMS-Schlüssel eine Berechtigung, die für die gesamte Lebensdauer des DB-Clusters verwendet wird. Das Widerrufen des Zugriffs des aufrufenden Prinzipals auf den KMS-Schlüssel hat keine Auswirkungen auf eine laufende Datenbank. Wenn KMS-Schlüssel in kontoübergreifenden Szenarien verwendet werden, z. B. beim Kopieren eines Snapshots in ein anderes Konto, muss der KMS-Schlüssel für das andere Konto freigegeben werden. Wenn Sie aus dem Snapshot einen DB-Cluster erstellen, ohne einen anderen KMS-Schlüssel anzugeben, verwendet der neue Cluster den KMS-Schlüssel aus dem Quellkonto. Das Widerrufen des Zugriffs auf den Schlüssel nach dem Erstellen des DB-Clusters hat keine Auswirkungen auf den Cluster. Die Deaktivierung des Schlüssels wirkt sich jedoch auf alle DB-Cluster aus, die mit diesem Schlüssel verschlüsselt

wurden. Um dies zu verhindern, geben Sie während des Snapshot-Kopiervorgangs einen anderen Schlüssel an.

Weitere Informationen über KMS-Schlüssel finden Sie unter AWS KMS keys im Entwicklerhandbuch zu AWS Key Management Service und in AWS KMS key-Verwaltung.

Verschlüsseln eines Amazon-Aurora-DB-Clusters

Wählen Sie Enable encryption (Verschlüsselung aktivieren) in der Konsole aus, um einen neuen DB-Cluster zu verschlüsseln. Weitere Informationen zum Erstellen eines DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Wenn Sie den AWS CLI-Befehl create-db-cluster verwenden, um einen verschlüsselten DB-Cluster zu erstellen, legen Sie den Parameter `--storage-encrypted` fest. Wenn Sie die API-Operation CreateDBCluster verwenden, legen Sie den Parameter `StorageEncrypted` auf „true“ fest.

Sobald Sie einen verschlüsselten DB-Cluster erstellt haben, können Sie den von diesem DB-Cluster verwendeten KMS-Schlüssel nicht mehr ändern. Stellen Sie daher sicher, dass Sie Ihre KMS-Schlüsselanforderungen bestimmen, bevor Sie Ihren verschlüsselten DB-Cluster erstellen.

Wenn Sie den Befehl AWS CLI `create-db-cluster` verwenden, um einen verschlüsselten DB-Cluster mit einem vom Kunden verwalteten Schlüssel zu erstellen, setzen Sie den Parameter `--kms-key-id` auf einen beliebigen Schlüsselbezeichner für den KMS-Schlüssel. Wenn Sie den Vorgang Amazon RDS API `CreateDBInstance` verwenden, setzen Sie den Parameter `KmsKeyId` auf einen beliebigen Schlüsselbezeichner für den KMS-Schlüssel. Um einen vom Kunden verwalteten Schlüssel in einem anderen AWS-Konto zu verwenden, geben Sie die Schlüssel-ARN oder Alias-ARN an.

Bestimmen, ob die Verschlüsselung für einen DB-Cluster aktiviert ist

Sie können die AWS-Managementkonsole, AWS CLI, oder RDS-API verwenden, um zu bestimmen, ob die Verschlüsselung im Ruhezustand für einen DB-Cluster aktiviert ist.

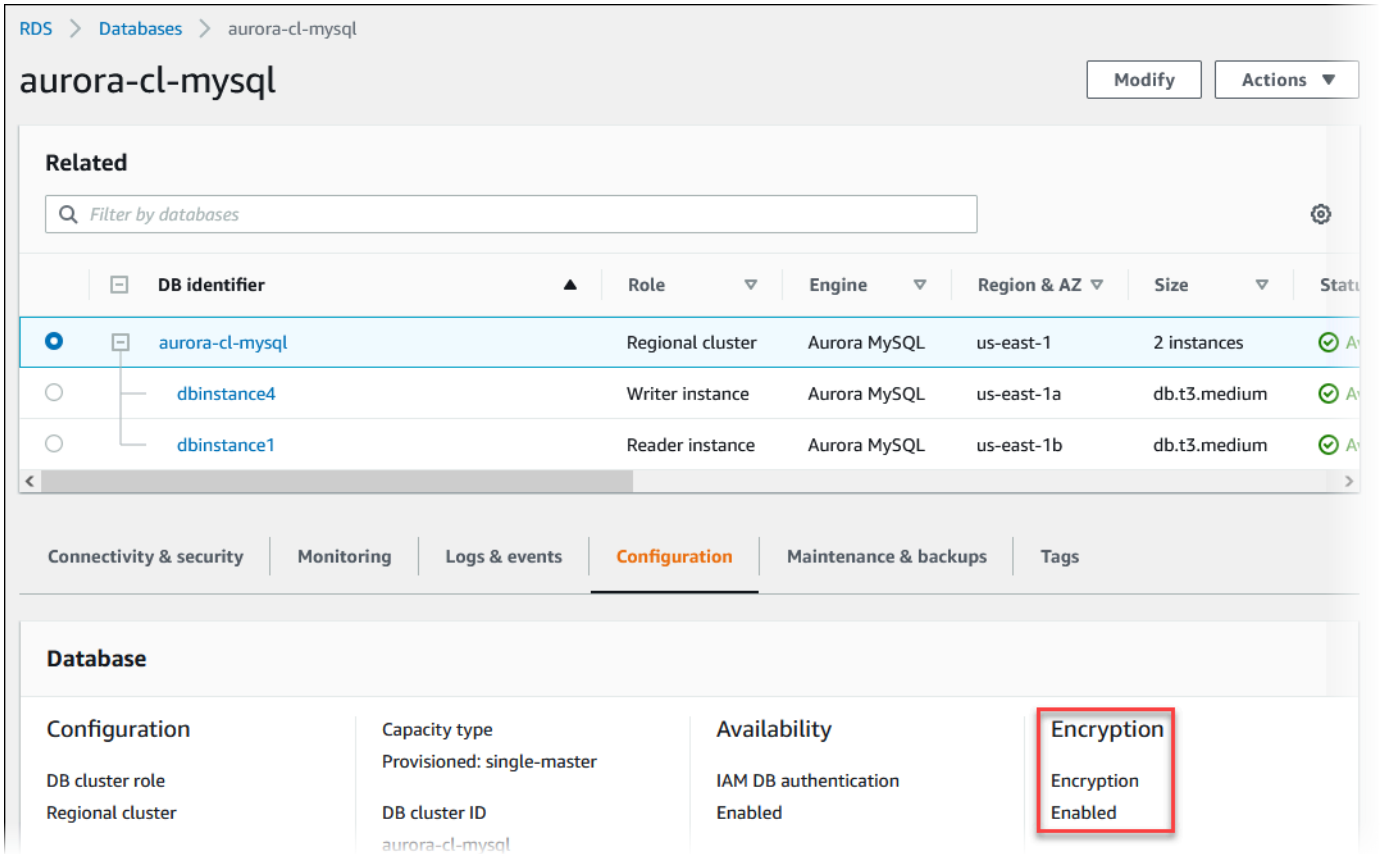
Konsole

So ermitteln Sie, ob die Verschlüsselung im Ruhezustand für einen DB-Cluster aktiviert ist

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

2. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
3. Wählen Sie den Namen des DB-Clusters aus, den Sie überprüfen möchten, um dessen Details anzuzeigen.
4. Wählen Sie die Registerkarte Konfiguration aus und überprüfen Sie den Wert für die Verschlüsselung.

   Es zeigt entweder Aktiviert oder Nicht aktiviert.



AWS CLI

Um festzustellen, ob die Verschlüsselung im Ruhezustand für einen DB-Cluster mithilfe von AWS CLI aktiviert ist, rufen Sie den Befehl describe-db-clusters mit der folgenden Option auf:

- `--db-cluster-identifier` – der Name des DB-Clusters.

Im folgenden Beispiel wird eine Abfrage verwendet, um entweder `TRUE` oder `FALSE` bezüglich der Verschlüsselung im Ruhezustand für den `mydb` DB-Cluster zurückzugeben.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

RDS-API

Um zu ermitteln, ob die Verschlüsselung im Ruhezustand für einen DB-Cluster mithilfe der Amazon-RDS-API aktiviert ist, rufen Sie die Operation DescribeDBClusters mit dem folgenden Parameter auf:

- `DBClusterIdentifier` – der Name des DB-Clusters.

Verfügbarkeit der Amazon Aurora-Verschlüsselung

Die Amazon Aurora-Verschlüsselung ist aktuell für alle Datenbank-Engines und Speichertypen verfügbar.

> **Note**
>
> Die Amazon Aurora-Verschlüsselung ist für die DB-Instance-Klasse db.t2.micro nicht verfügbar.

Verschlüsselung während der Übertragung

Verschlüsselung auf physischer Ebene

Alle Daten, die über AWS-Regionen-Regionen über das globale AWS-Netzwerk übertragen werden, werden automatisch auf der physischen Ebene verschlüsselt, bevor sie gesicherte AWS-Einrichtungen verlassen. Der gesamte Datenverkehr zwischen AZs ist verschlüsselt. Zusätzliche Verschlüsselungsebenen, einschließlich der in diesem Abschnitt aufgeführten, bieten möglicherweise zusätzlichen Schutz.

Verschlüsselung bereitgestellt durch Amazon-VPC-Peering und regionsübergreifendem Peering in Transit Gateway

Der gesamte regionsübergreifende Datenverkehr, der Amazon-VPC- und Transit-Gateway-Peering verwendet, wird automatisch massenverschlüsselt, wenn er eine Region verlässt. Auf der physischen Ebene wird automatisch eine zusätzliche Verschlüsselungsebene für den gesamten Datenverkehr bereitgestellt, bevor er AWS gesicherte Einrichtungen verlässt.

Verschlüsselung zwischen den Instances

AWS bietet sichere und private Konnektivität zwischen DB-Instances aller Typen. Darüber hinaus verwenden einige Instance-Typen die Offload-Funktionen der zugrunde liegenden Nitro-System-Hardware, um den Datenverkehr während der Übertragung zwischen Instances automatisch zu verschlüsseln. Diese Verschlüsselung verwendet AEAD-Algorithmen (Authenticated Encryption with Associated Data) mit 256-Bit-Verschlüsselung. Es gibt keine Auswirkungen auf die Netzwerkleistung. Um diese zusätzliche Verschlüsselung des Datenverkehrs während der Übertragung zwischen Instances zu unterstützen, müssen die folgenden Anforderungen erfüllt sein:

- Die Instances verwenden die folgenden Instance-Typen:
  - Allgemeiner Zweck: M6i, M6id, M6in, M6idn, M7g
  - Speicheroptimiert: R6i, R6id, R6in, R6idn, R7g, X2idn, X2iedn, X2iezn
- Die Instances befinden sich in derselben AWS-Region.
- Die Instances befinden sich in derselben VPC oder in per Peering verbundenen VPCs und der Datenverkehr wird nicht durch ein virtuelles Netzwerkgerät, z. B. einen Load Balancer oder ein Transit Gateway, geleitet.

Einschränkungen von Amazon Aurora-verschlüsselten DB-Clustern

Folgende Einschränkungen bestehen für Amazon Aurora-verschlüsselte DB-Cluster:

- Sie können die Verschlüsselung für einen verschlüsselten DB-Cluster nicht deaktivieren.
- Sie können keinen verschlüsselten Snapshot einer/eines unverschlüsselten DB-Clusters erstellen.
- Ein Snapshot eines verschlüsselten DB-clusters muss mit demselben KMS-Schlüssel verschlüsselt werden wie der DB-cluster.
- Unverschlüsselte DB-Cluster können nicht in verschlüsselte umgewandelt werden. Sie können jedoch einen unverschlüsselten Snapshot in einen verschlüsselten Aurora-DB-Cluster wiederherstellen. Geben Sie dazu bei der Wiederherstellung aus dem unverschlüsselten Snapshot einen KMS-Schlüssel an.
- Sie können keine verschlüsselte Aurora-Replica aus einem unverschlüsselten Aurora-DB-Cluster erstellen. Sie können keine unverschlüsselte Aurora-Replica aus einem verschlüsselten Aurora-DB-Cluster erstellen.

- Um einen verschlüsselten Snapshot von einer AWS Region in eine andere zu kopieren, müssen Sie den KMS-Schlüssel in der Zielregion AWS angeben. Dies liegt daran, dass KMS-Schlüssel spezifisch für die AWS Region sind, in der sie erstellt werden.

  Der Quell-Snapshot bleibt den gesamten Kopiervorgang über verschlüsselt. Amazon Aurora verwendet Envelope-Verschlüsselung, um Daten während des Kopiervorgangs zu schützen. Weitere Informationen zur Envelope-Verschlüsselung finden Sie unter Envelope-Verschlüsselung im AWS Key Management Service-Entwicklerhandbuch.
- Sie können eine(n) verschlüsselte(n) DB-Cluster nicht entschlüsseln. Sie können jedoch Daten aus einer/einem verschlüsselten DB-Cluster exportieren und die Daten in eine(n) unverschlüsselte(n) DB-Cluster importieren.

AWS KMS key-Verwaltung

Amazon Aurora wird automatisch zur Schlüsselverwaltung in AWS Key Management Service (AWS KMS) integriert. Amazon Aurora verwendet eine Envelope-Verschlüsselung. Weitere Informationen zur Envelope-Verschlüsselung finden Sie unter Envelope-Verschlüsselung im AWS Key Management Service-Entwicklerhandbuch.

Sie können zwei Typen von AWS KMS-Schlüsseln verwenden, um Ihre DB-Cluster zu verschlüsseln.

- Wenn Sie die volle Kontrolle über einen KMS-Schlüssel haben möchten, müssen Sie einen vom Kunden verwalteten Schlüssel erstellen. Weitere Informationen über kundenverwaltete Schlüssel finden Sie unter Kundenverwaltete Schlüssel im AWS Key Management Service Developer Guide.
- Von AWS verwaltete Schlüssel sind KMS-Schlüssel in Ihrem Konto, die in Ihrem Namen von einem AWS Dienst erstellt, verwaltet und verwendet werden, der mit AWS KMS. Standardmäßig wird der RDS-Von AWS verwalteter Schlüssel (`aws/rds`) für die Verschlüsselung verwendet. Sie können den RDS-Von AWS verwalteter Schlüssel nicht verwalten, rotieren oder löschen. Weitere Informationen zu Von AWS verwaltete Schlüssel finden Sie unter Von AWS verwaltete Schlüssel im AWS Key Management Service-Entwickler-Leitfaden.

Zur Verwaltung von KMS-Schlüsseln für verschlüsselte DB-Cluster von Amazon Aurora verwenden Sie den AWS Key Management Service (AWS KMS) in der AWS KMS-Konsole, die AWS CLI oder die AWS KMS-API. Verwenden Sie zur Anzeige von Audit-Protokollen für jede Aktion, die mit einem AWS-verwalteten oder kundenverwalteten Schlüssel durchgeführt wurde AWS CloudTrail. Weitere Informationen zum Rotieren der Schlüssel finden Sie unter Rotieren von AWS KMS-Schlüsseln.

Autorisieren der Verwendung eines kundenverwalteten Schlüssels

Wenn Aurora einen vom Kunden verwalteten Schlüssel für kryptografische Vorgänge verwendet, handelt es im Namen des Benutzers, der die Aurora-Ressource erstellt oder ändert.

Wenn Sie eine Aurora-Ressource mit einem vom Kunden verwalteten Schlüssel erstellen möchten, müssen Sie über die Berechtigung verfügen, die folgenden Vorgänge für den vom Kunden verwalteten Schlüssel aufzurufen:

- `kms:CreateGrant`
- `kms:DescribeKey`

Sie können diese erforderlichen Berechtigungen in einer Schlüsselrichtlinie oder in einer IAM-Richtlinie angeben, wenn die Schlüsselrichtlinie dies zulässt.

> **Important**
>
> Wenn Sie explizite Zugriffsverweigerungsanweisungen für alle Ressourcen (*) in AWS KMS-Schlüsselrichtlinien mit verwalteten Services wie Amazon RDS verwenden, müssen Sie eine Bedingung angeben, damit das Konto als Eigentümer der Ressource agieren kann. Ohne diese Bedingung können Vorgänge fehlschlagen, auch wenn die Zugriffsverweigerungsregel Ausnahmen für Ihren IAM-Benutzer enthält.

> **Tip**
>
> Um den Grundsatz der Erteilung der geringsten erforderlichen Berechtigungen zu befolgen, lassen Sie den vollständigen Zugriff auf `kms:CreateGrant` nicht zu. Verwenden Sie stattdessen den kms:ViaService-Bedingungsschlüssel, um dem Benutzer die Erstellung von Erteilungen auf dem KMS-Schlüssel nur dann zu erlauben, wenn die Erteilungen im Namen des Benutzers von einem AWS-Service erstellt wird.

Sie können die IAM-Richtlinie auf verschiedene Weise strikter gestalten. Um beispielsweise zu erlauben, dass der vom Kunden verwaltete Schlüssel nur für Anfragen verwendet wird, die von Aurora ausgehen, können Sie den kms:ViaService-Bedingungsschlüssel mit dem Wert `rds.<region>.amazonaws.com` verwenden. Sie können auch die Schlüssel oder Werte im

Amazon-RDS-Verschlüsselungskontext als Bedingung für die Verwendung des vom Kunden verwalteten Schlüssels für die Verschlüsselung verwenden.

Weitere Informationen finden Sie unter Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben im AWS Key Management Service-Entwicklerhandbuch und unter Schlüsselrichtlinien in AWS KMS.

Amazon-RDS-Verschlüsselungskontext

Wenn Aurora Ihren KMS-Schlüssel verwendet oder Amazon EBS den KMS-Schlüssel im Namen von Aurora verwendet, gibt der Service einen Verschlüsselungskontext an. Der Verschlüsselungskontext enthält zusätzliche authentifizierte Daten (AAD), anhand derer AWS KMS die Datenintegrität sicherstellt. Wenn für eine Verschlüsselungsoperation ein Verschlüsselungskontext angegeben wird, muss der Service denselben Verschlüsselungskontext auch für die Entschlüsselungsoperation angeben. Andernfalls schlägt die Entschlüsselung fehl. Der Verschlüsselungskontext wird zudem in Ihre AWS CloudTrail-Protokolle geschrieben, sodass Sie jederzeit nachvollziehen können, warum ein bestimmter KMS-Schlüssel verwendet wurde. Ihre CloudTrail-Protokolle können sehr viele Einträge zur Verwendung eines KMS-Schlüssels enthalten. Der Verschlüsselungskontext in den einzelnen Protokolleinträgen kann Ihnen jedoch helfen, herauszufinden, warum der KMS-Schlüssel zu einem gegebenen Zeitpunkt verwendet wurde.

Aurora gibt als Verschlüsselungskontext immer mindestens die ID der DB-des CB-Clusters an, wie im folgenden Beispiel im JSON-Format gezeigt:

```
{ "aws:rds:dbc-id": "cluster-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

Anhand dieses Verschlüsselungskontexts können Sie herausfinden, für welchen DB-Cluster der KMS-Schlüssel verwendet wurde.

Wird Ihr KMS-Schlüssel für einen bestimmten DB-Cluster und ein bestimmtes Amazon-EBS-Volume verwendet, werden die ID des DB-Clusters und die ID des EBS-Volumes als Verschlüsselungskontext angegeben, wie im folgenden Beispiel im JSON-Format gezeigt:

```
{
  "aws:rds:dbc-id": "cluster-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln

Sie können Secure Socket Layer (SSL) oder Transport Layer Security (TLS) aus Ihrer Anwendung verwenden, um eine Verbindung zu einem DB-Cluster zu verschlüsseln, der mit Aurora MySQL oder Aurora PostgreSQL läuft.

SSL/TLS-Verbindungen bieten eine Sicherheitsebene, indem Daten verschlüsselt werden, die zwischen Ihrem Client und DB -Clustern verschoben werden. Optional kann Ihre SSL/TLS Verbindung die Serveridentität überprüfen, indem das in Ihrer Datenbank installierte Serverzertifikat validiert wird. Gehen Sie wie folgt vor, um eine Überprüfung der Serveridentität vorzuschreiben:

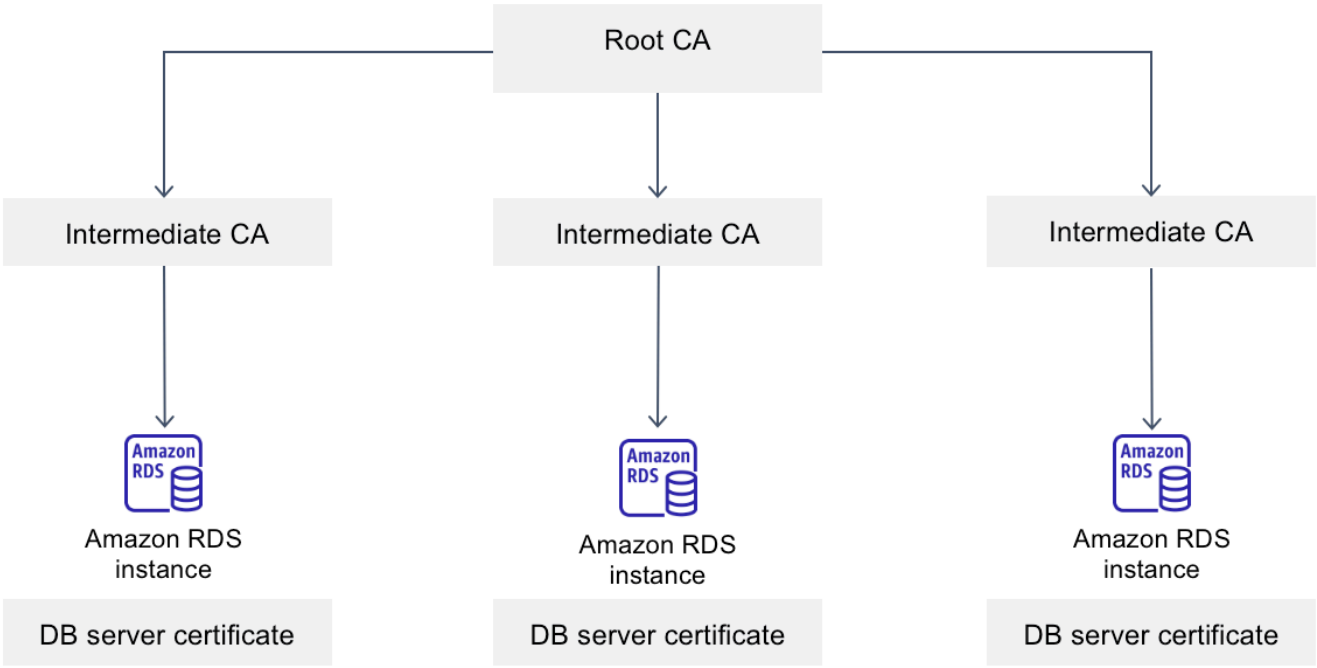
1. Wählen Sie die Zertifizierungsstelle (Certificate Authority, CA) aus, die das DB-Serverzertifikat für Ihre Datenbank zertifiziert. Weitere Informationen zu Zertifizierungsstellen finden Sie unterZertifizierungsstellen.
2. Laden Sie ein Zertifikatpaket herunter, das verwendet werden soll, wenn Sie eine Verbindung zur Datenbank herstellen. Informationen zum Herunterladen eines Zertifikatspakets finden Sie unter Zertifikatspakete von AWS-Region .

   > **Note**
   >
   > Alle Zertifikate stehen nur über SSL/TLS-Verbindungen zum Download zur Verfügung.

3. Connect Sie mithilfe des Prozesses Ihrer DB-Engine zur Implementierung von SSL/TLS Verbindungen eine Verbindung zur Datenbank her. Jede DB-Engine hat ihren eigenen Prozess für die Implementierung SSL/TLS. To learn how to implement SSL/TLS für Ihre Datenbank. Folgen Sie dem Link, der Ihrer DB-Engine entspricht:
   - Sicherheit in Amazon Aurora MySQL
   - Sicherheit in Amazon Aurora PostgreSQL

Zertifizierungsstellen

Die Zertifizierungsstelle (CA) ist das Zertifikat, das die Stamm-CA an der Spitze der Zertifikatskette identifiziert. Die CA signiert das DB-Serverzertifikat. Dies ist das Serverzertifikat, das auf jeder DB-Instance installiert ist. Das DB-Serverzertifikat identifiziert die DB-Instance als vertrauenswürdigen Server.



Amazon RDS bietet Folgendes CAs , um das DB-Serverzertifikat für eine Datenbank zu signieren.

Zertifizierungsstelle (Certificate authority, CA)	Description	Allgemeiner Name (CN)
rds-ca-rsa2048-g1	Verwendet in den meisten Fällen eine Zertifizierungsstelle mit einem RSA 2048-Algorithmus für private Schlüssel und SHA256 einem Signaturalgorithmus. AWS-Regionen In der AWS GovCloud (US) Regions verwendet diese Zertifizierungsstelle eine Zertifizierungsstelle mit einem RSA 2048-Algorithmus für private Schlüssel und einem Signaturalgorithmus. SHA384 Diese CA unterstützt die automatische Rotation von Serverzertifikaten.	Amazon RDS *region-id entifier* Root CA RSA2 048 G1

Zertifizierungsstelle (Certificate authority, CA)	Description	Allgemeiner Name (CN)
rds-ca-rsa4096-g1	Verwendet eine Zertifizierungsstelle mit RSA 4096-Algorithmus und Signaturalgorithmus für private Schlüssel. SHA384 Diese CA unterstützt die automatische Rotation von Serverzertifikaten.	Amazon RDS *region-identifier* Root CA RSA4 096 G1
rds-ca-ecc384-g1	Verwendet eine Zertifizierungsstelle mit ECC 384-Algorithmus für private Schlüssel und Signaturalgorithmus. SHA384 Diese CA unterstützt die automatische Rotation von Serverzertifikaten.	Amazon RDS *region-identifier* Root CA ECC384 G1

> **Note**
>
> Wenn Sie das verwenden AWS CLI, können Sie die Gültigkeiten der oben aufgeführten Zertifizierungsstellen mithilfe von describe-certificates überprüfen.

Diese CA-Zertifikate sind im regionalen und globalen Zertifikatpakets enthalten. Wenn Sie die Zertifizierungsstelle rds-ca-rsa 2048-g1, rds-ca-rsa 4096-g1 oder rds-ca-ecc 384-g1 mit einer Datenbank verwenden, verwaltet RDS das DB-Serverzertifikat in der Datenbank. RDS rotiert das DB-Serverzertifikat automatisch, bevor es abläuft.

Einstellung der CA für Ihre Datenbank

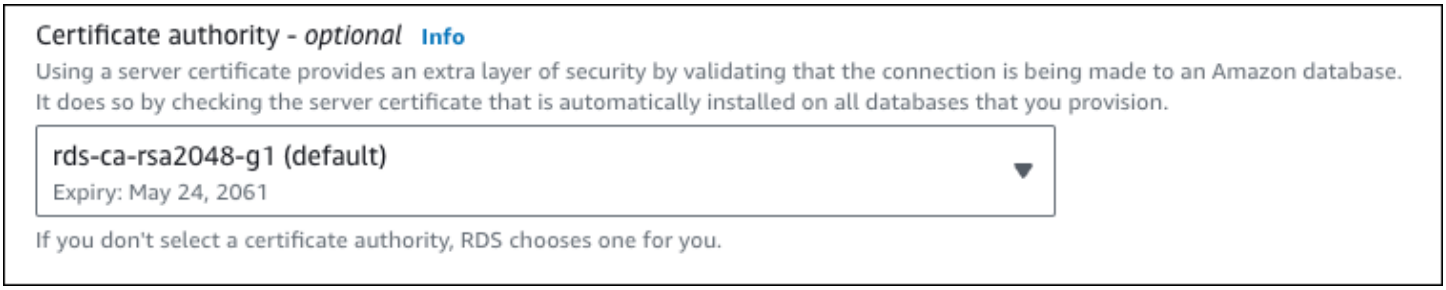
Sie können die CA für eine Datenbank einstellen, wenn Sie die folgenden Aufgaben ausführen:

- Einen Aurora-DB-Cluster erstellen — Sie können die CA für eine DB-Instance in einem Aurora-Cluster festlegen, wenn Sie die erste DB-Instance im DB-Cluster mithilfe der AWS CLI oder RDS-API erstellen. Derzeit können Sie die CA für die DB-Instances in einem DB-Cluster nicht einstellen, wenn Sie den DB-Cluster über die RDS-Konsole erstellen. Anleitungen finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

- Eine DB-Instance ändern – Sie können die CA für eine DB-Instance in einem DB-Cluster festlegen, indem Sie sie ändern. Anleitungen finden Sie unter Ändern einer DB-Instance in einem DB-Cluster.

> **Note**
>
> Die Standard-CA ist auf rds-ca-rsa 2048-g1 festgelegt. Sie können die Standard-CA für Sie überschreiben, AWS-Konto indem Sie den Befehl modify-certificates verwenden.

Die verfügbaren Optionen CAs hängen von der DB-Engine und der DB-Engine-Version ab. Wenn Sie die verwenden AWS-Managementkonsole, können Sie die Zertifizierungsstelle mithilfe der Einstellung Zertifizierungsstelle auswählen, wie in der folgenden Abbildung dargestellt.



In der Konsole werden nur die angezeigt CAs , die für die DB-Engine und die DB-Engine-Version verfügbar sind. Wenn Sie die verwenden AWS CLI, können Sie die CA für eine DB-Instance mit dem modify-db-instanceBefehl create-db-instanceor festlegen.

Wenn Sie den verwenden AWS CLI, können Sie mit dem Befehl describe-certificates die CAs für Ihr Konto verfügbaren Werte anzeigen. Dieser Befehl zeigt in der Ausgabe auch das Ablaufdatum für jede CA in `ValidTill` an. Mithilfe des Befehls können Sie herausfinden CAs , welche für eine bestimmte DB-Engine und DB-Engine-Version verfügbar sind. describe-db-engine-versions

Das folgende Beispiel zeigt die für die Standardversion der RDS for PostgreSQL DB-Engine CAs verfügbare Version.

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

Ihre Ausgabe sieht Folgendem ähnlich. Die verfügbaren CAs sind unter aufgeführt. `SupportedCACertificateIdentifiers` Die Ausgabe zeigt auch, ob die Version der DB-Engine das Rotieren des Zertifikats ohne Neustart in `SupportsCertificateRotationWithoutRestart` unterstützt.

```
{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

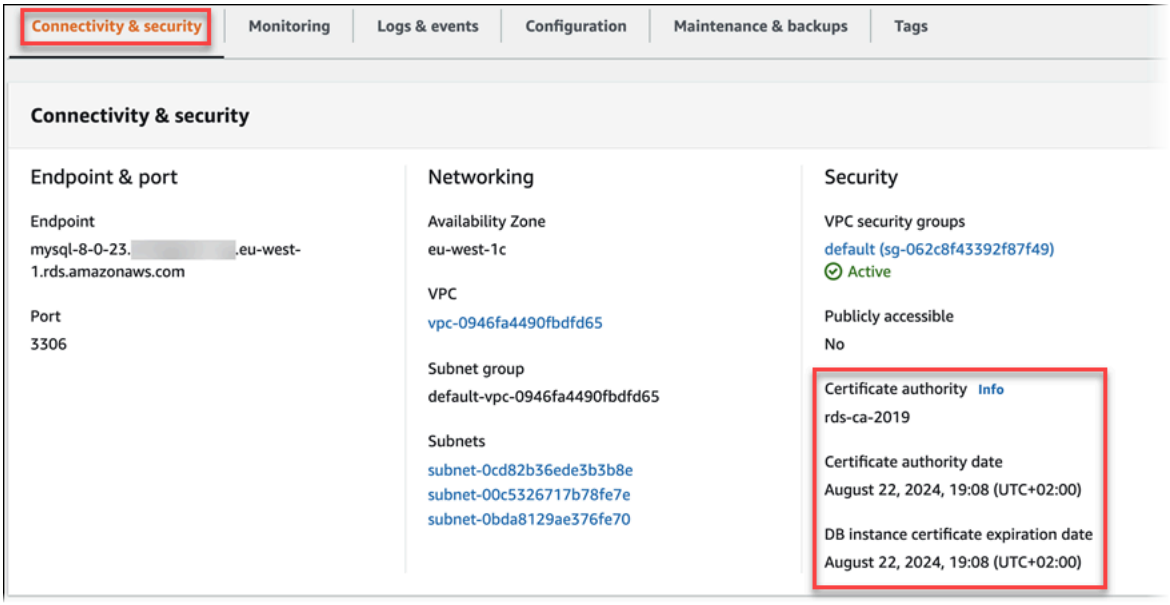

Gültigkeiten von DB-Serverzertifikaten

Die Gültigkeit des DB-Serverzertifikats hängt von der DB-Engine und der Version der DB-Engine ab. Wenn die Version der DB-Engine das Rotieren des Zertifikats ohne Neustart unterstützt, beträgt die Gültigkeit des DB-Serverzertifikats 1 Jahr. Andernfalls beträgt die Gültigkeit 3 Jahre.

Weitere Informationen zur Rotation des DB-Serverzertifikats finden Sie unter Automatische Rotation von Serverzertifikaten.

Anzeigen der CA für Ihre DB-Instance

Sie können die Details zur CA für eine Datenbank anzeigen, indem Sie die Registerkarte Konnektivität und Sicherheit in der Konsole aufrufen, wie in der folgenden Abbildung dargestellt.



Wenn Sie die verwenden AWS CLI, können Sie die Details zur CA für eine DB-Instance mithilfe des describe-db-instancesBefehls anzeigen.

Zertifikatspakete für Aurora herunterladen

Wenn Sie mit SSL oder TLS eine Verbindung zu Ihrer Datenbank herstellen, benötigt die Datenbank-Instance ein Vertrauenszertifikat von Amazon RDS. Wählen Sie den entsprechenden Link in der folgenden Tabelle aus, um das Paket herunterzuladen, das der AWS-Region entspricht, in der Sie Ihre Datenbank hosten.

Zertifikatspakete von AWS-Region

Die Zertifikatspakete für alle Regionen AWS-Regionen und GovCloud (USA) enthalten die folgenden Root-CA-Zertifikate:

- `rds-ca-rsa2048-g1`
- `rds-ca-rsa4096-g1`
- `rds-ca-ecc384-g1`

Die Zertifikate `rds-ca-rsa4096-g1` und `rds-ca-ecc384-g1` sind in den folgenden Regionen nicht verfügbar:

- Asien-Pazifik (Mumbai)
- Asien-Pazifik (Melbourne)

- Kanada West (Calgary)
- Europa (Zürich)
- Europa (Spain)
- Israel (Tel Aviv)

Ihr Application Trust Store muss nur das Root-CA-Zertifikat registrieren. Registrieren Sie die CA-Zwischenzertifikate nicht in Ihrem Trust Store, da dies zu Verbindungsproblemen führen kann, wenn RDS Ihr DB-Serverzertifikat automatisch rotiert.

> **Note**
>
> Amazon RDS Proxy and Zertifikate von AWS Certificate Manager (ACM). Wenn Sie RDS-Proxy verwenden, müssen Sie keine Amazon-RDS-Zertifikate herunterladen oder Anwendungen aktualisieren, die RDS-Proxy-Verbindungen verwenden. Weitere Informationen finden Sie unter Verwendung TLS/SSL mit RDS Proxy.
> Wenn Sie Aurora Serverless v1 verwenden, ist das Herunterladen von Amazon-RDS-Zertifikaten nicht erforderlich. Weitere Informationen finden Sie unter Verwenden von TLS/SSL mit Aurora Serverless v1.

Um ein Zertifikatspaket für einen herunterzuladen AWS-Region, wählen Sie in der folgenden Tabelle den Link für den Server aus AWS-Region , der Ihre Datenbank hostet.

AWS Region	Zertifikat-Paket (PEM)	Zertifikatspaket (PKCS7)
Irgendein kommerzieller AWS-Region	global-bundle.pem	global-bundle.p7b
USA Ost (Nord-Virginia)	us-east-1-bundle.pem	us-east-1-bundle.p7b
US East (Ohio)	us-east-2-bundle.pem	us-east-2-bundle.p7b
USA West (Nordkalifornien)	us-west-1-bundle.pem	us-west-1-bundle.p7b
USA West (Oregon)	us-west-2-bundle.pem	us-west-2-bundle.p7b
Africa (Cape Town)	af-south-1-bundle.pem	af-south-1-bundle.p7b

AWS Region	Zertifikat-Paket (PEM)	Zertifikatspaket (PKCS7)
Asia Pacific (Hong Kong)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
Asien-Pazifik (Hyderabad)	ap-south-2-bundle.pem	ap-south-2-bundle.p7b
Asien-Pazifik (Jakarta)	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
Asien-Pazifik (Malaysia)	ap-southeast-5-bundle.pem	ap-southeast-5-bundle.p7b
Asien-Pazifik (Melbourne)	ap-southeast-4-bundle.pem	ap-southeast-4-bundle.p7b
Asien-Pazifik (Mumbai)	ap-south-1-bundle.pem	ap-south-1-bundle.p7b
Asia Pacific (Osaka)	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b
Asien-Pazifik (Thailand)	ap-southeast-7-bundle.pem	ap-southeast-7-bundle.p7b
Asien-Pazifik (Tokio)	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
Asia Pacific (Seoul)	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
Asien-Pazifik (Singapur)	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
Asien-Pazifik (Sydney)	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b
Canada (Central)	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
Kanada West (Calgary)	ca-west-1-bundle.pem	ca-west-1-bundle.p7b
Europa (Frankfurt)	eu-central-1-bundle.pem	eu-central-1-bundle.p7b
Europa (Irland)	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
Europe (London)	eu-west-2-bundle.pem	eu-west-2-bundle.p7b
Europe (Milan)	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
Europe (Paris)	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
Europa (Spain)	eu-south-2-bundle.pem	eu-south-2-bundle.p7b

AWS Region	Zertifikat-Paket (PEM)	Zertifikatspaket (PKCS7)
Europa (Stockholm)	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
Europa (Zürich)	eu-central-2-bundle.pem	eu-central-2-bundle.p7b
Israel (Tel Aviv)	il-central-1-bundle.pem	il-central-1-bundle.p7b
Mexiko (Zentral)	mx-central-1-bundle.pem	mx-central-1-bundle.p7b
Middle East (Bahrain)	me-south-1-bundle.pem	me-south-1-bundle.p7b
Naher Osten (VAE)	me-central-1-bundle.pem	me-central-1-bundle.p7b
Südamerika (São Paulo)	sa-east-1-bundle.pem	sa-east-1-bundle.p7b
Irgendein s AWS GovCloud (US) Region	global-bundle.pem	global-bundle.p7b
AWS GovCloud (US-Ost)	us-gov-east-1-bundle.pem	us-gov-east-1-Bundle.p7b
AWS GovCloud (US-West)	us-gov-west-1-bundle.pem	us-gov-west-1-Bundle.p7b

Anzeigen des Inhalts Ihres CA-Zertifikats

Verwenden Sie den folgenden Befehl, um den Inhalt Ihres CA-Zertifikatpakets zu überprüfen:

```
keytool -printcert -v -file global-bundle.pem
```

Rotieren Ihrer SSL/TLS-Zertifikate

Die Zertifikate der Amazon-RDS-Zertifizierungsstelle rds-ca-2019 sind im August 2024 abgelaufen. Wenn Sie Secure Sockets Layer (SSL) oder Transport Layer Security (TLS) mit Zertifikatsüberprüfung verwenden oder dies planen, um eine Verbindung zu Ihren RDS-DB-Instances herzustellen, sollten Sie die Verwendung eines der neuen CA-Zertifikate rds-ca-rsa 2048-g1, rds-ca-rsa 4096-g1 oder 384-g1 in Betracht ziehen. rds-ca-ecc Wenn Sie derzeit keine Zertifikatsverifizierung verwenden SSL/TLS , haben Sie möglicherweise immer noch ein abgelaufenes CA-Zertifikat und müssen es auf ein neues CA-Zertifikat aktualisieren, wenn Sie es SSL/TLS mit der Zertifikatsverifizierung verwenden möchten, um eine Verbindung zu Ihren RDS-Datenbanken herzustellen.

Amazon RDS bietet neue CA-Zertifikate als bewährte AWS Sicherheitsmethode. Informationen zu den neuen Zertifikaten und den unterstützten AWS Regionen finden Sie unterWird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

Verwenden Sie die folgenden Methoden, um das CA-Zertifikat für Ihre Datenbank zu aktualisieren:

- Aktualisieren des CA-Zertifikats durch Ändern der DB-Instance
- Aktualisieren des CA-Zertifikats durch Anwenden der Wartung

Bevor Sie Ihre DB-Instances zur Verwendung des neuen CA-Zertifikats aktualisieren, stellen sie sicher, dass Sie Ihre Clients oder Anwendungen aktualisieren, die eine Verbindung mit Ihren RDS-Datenbanken herstellen.

Überlegungen zur Rotation von Zertifikaten

Beachten Sie die folgenden Situationen, bevor Sie Ihr Zertifikat rotieren:

- Amazon RDS Proxy and Zertifikate von AWS Certificate Manager (ACM). Wenn Sie RDS Proxy verwenden, müssen Sie beim Rotieren Ihres SSL/TLS Zertifikats keine Anwendungen aktualisieren, die RDS-Proxy-Verbindungen verwenden. Weitere Informationen finden Sie unter Verwendung TLS/SSL mit RDS Proxy.
- Wenn Sie Aurora Serverless v1 verwenden, ist das Herunterladen von Amazon-RDS-Zertifikaten nicht erforderlich. Weitere Informationen finden Sie unter Verwenden von TLS/SSL mit Aurora Serverless v1.
- Wenn Sie eine Go-Anwendung der Version 1.15 mit einer DB-Instance verwenden, die vor dem 28. Juli 2020 erstellt oder auf das Zertifikat rds-ca-2019 aktualisiert wurde, müssen Sie das Zertifikat erneut aktualisieren.

  Verwenden Sie den Befehl `modify-db-instance` unter Verwendung der neuen CA-Zertifikat-ID. Sie können die, die für eine bestimmte DB-Engine und DB-Engine-Version verfügbar sind CAs , mit dem Befehl finden. `describe-db-engine-versions`

  Wenn Sie Ihre Datenbank nach dem 28. Juli 2020 erstellt oder das entsprechende Zertifikat aktualisiert haben, ist keine Aktion erforderlich. Weitere Informationen finden Sie in GitHub Go-Ausgabe #39568.

Aktualisieren des CA-Zertifikats durch Ändern der DB-Instance

Im folgenden Beispiel wird Ihr CA-Zertifikat von rds-ca-2019 auf 2048-g1 aktualisiert. rds-ca-rsa Sie können ein anderes Zertifikat wählen. Weitere Informationen finden Sie unterZertifizierungsstellen.

Aktualisieren Sie den Trust Store Ihrer Anwendung, um Ausfallzeiten im Zusammenhang mit der Aktualisierung Ihres CA-Zertifikats zu reduzieren. Weitere Informationen zu Neustarts in Zusammenhang mit der CA-Zertifikatrotation finden Sie unter Automatische Rotation von Serverzertifikaten.

So aktualisieren Sie das CA-Zertifikat durch Ändern der DB-Instance

1. Laden Sie das neue SSL/TLS Zertifikat herunter, wie unter beschriebenWird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.
2. Aktualisieren Sie Ihre Anwendungen zur Verwendung der neuen SSL/TLS-Zertifikate.

   Die Methoden zur Aktualisierung von Anträgen auf neue SSL/TLS Zertifikate hängen von Ihren spezifischen Anwendungen ab. Arbeiten Sie mit Ihren Anwendungsentwicklern zusammen, um die SSL/TLS Zertifikate für Ihre Anwendungen zu aktualisieren.

   Informationen zum Prüfen von SSL/TLS Verbindungen und zum Aktualisieren von Anwendungen für die einzelnen DB-Engines finden Sie in den folgenden Themen:

   - Aktualisieren von Anwendungen, um Verbindungen mit DB-Clustern von Aurora MySQL mithilfe neuer TLS-Zertifikate herzustellen
   - Aktualisieren von Anwendungen für die Verbindung zu Aurora-PostgreSQL-DB-Clustern mit neuen SSL/TLS-Zertifikaten

   Ein Beispielskript, das einen Trust Store für ein Linux-Betriebssystem aktualisiert, finden Sie unterBeispielskript für den Import von Zertifikaten in Ihren Trust Store.

   > **Note**
   >
   > Das Zertifikatpaket enthält Zertifikate für die neue und die alte CA, damit Sie Ihre Anwendung sicher aktualisieren und die Verbindung während der Übergangsphase aufrecht erhalten können. Wenn Sie das verwenden, um eine Datenbank AWS Database Migration Service zu einer zu migrieren, empfehlen wir die Verwendung des Zertifikatspakets, um die Konnektivität während der Migration sicherzustellen.

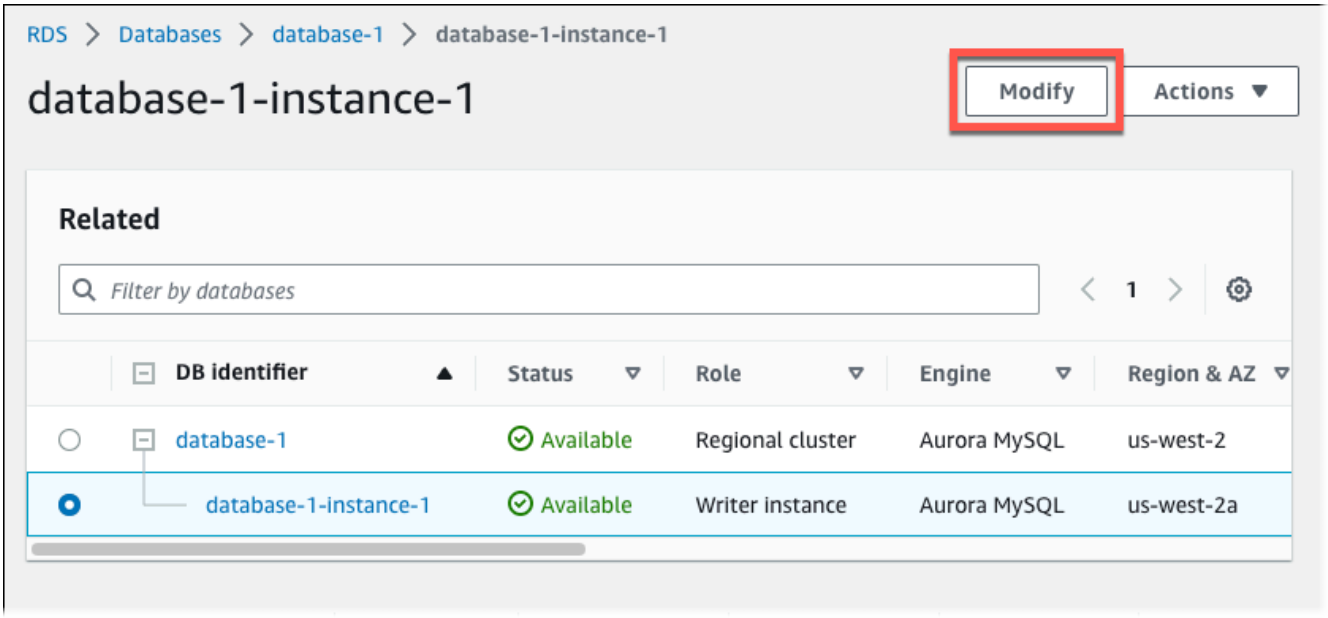
3. Ändern Sie die DB-Instance , um die CA von rds-ca-2019 auf 2048-g1 zu ändern. rds-ca-rsa Um zu überprüfen, ob Ihre Datenbank zum Aktualisieren der CA-Zertifikate neu gestartet werden muss, verwenden Sie den Befehl und überprüfen Sie das Flag. describe-db-engine-versions`SupportsCertificateRotationWithoutRestart`

> **Note**
>
> Starten Sie Ihren Babelfish-Cluster neu, nachdem Sie Änderungen vorgenommen haben, um das CA-Zertifikat zu aktualisieren.

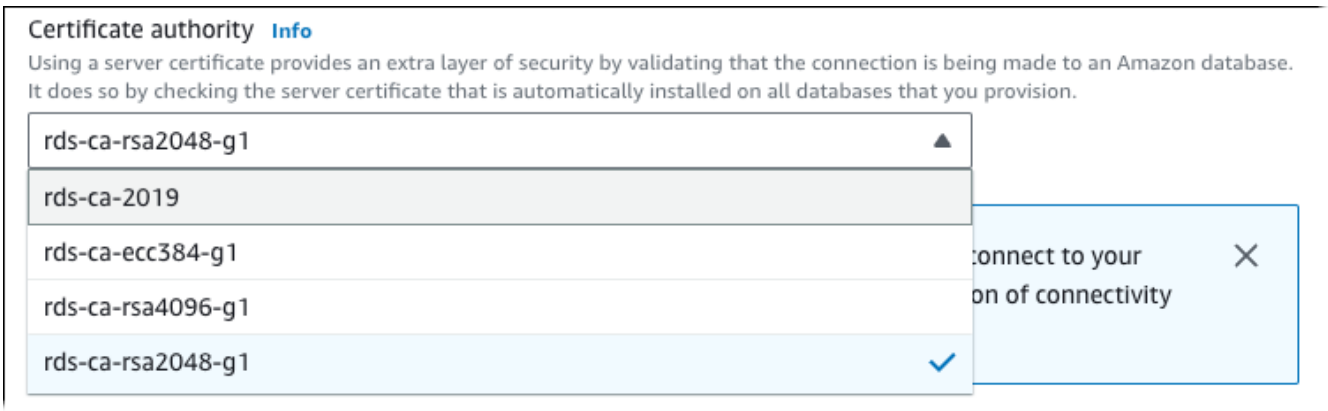
> **Important**
>
> Wenn nach dem Ablauf des Zertifikats Verbindungsprobleme auftreten, verwenden Sie die Option „Sofort anwenden“, indem Sie Sofort anwenden in der Konsole angeben oder die Option `--apply-immediately` mit der AWS CLI festlegen. Die Ausführung dieser Operation ist standardmäßig während Ihres nächsten Wartungsfensters eingeplant. Um eine Überschreibung für Ihre Cluster-CA festzulegen, die sich von der standardmäßigen RDS-CA unterscheidet, verwenden Sie den CLI-Befehl modify-certificates.

Console

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken und dann die DB-Instance für die Änderung aus.
3. Wählen Sie Ändern aus.



4. Wählen Sie im Bereich Konnektivität die Option rds-ca-rsa2048-g1 aus.



5. Klicken Sie auf Weiter und überprüfen Sie die Zusammenfassung aller Änderungen.
6. Wählen Sie Sofort anwenden aus, um die Änderungen sofort anzuwenden.
7. Überprüfen Sie auf der Bestätigungsseite Ihre Änderungen. Wenn sie korrekt sind, wählen Sie DB-Instance ändern aus, um Ihre Änderungen zu speichern.

> **Important**
>
> Wenn Sie diese Operation planen, stellen Sie sicher, dass Sie zuvor Ihren clientseitigen Vertrauensspeicher aktualisiert haben.

Oder klicken Sie auf Zurück, um Ihre Änderungen zu bearbeiten, oder auf Abbrechen, um Ihre Änderungen zu verwerfen.

AWS CLI

AWS CLI Um die CA für eine DB-Instance auf. modify-db-instancemodify-db-cluster Geben Sie die DB-Instance-Kennung und die Option `--ca-certificate-identifier` an.

Verwenden Sie den Parameter `--apply-immediately`, um das Update sofort anzuwenden. Die Ausführung dieser Operation ist standardmäßig während Ihres nächsten Wartungsfensters eingeplant.

> **Important**
>
> Wenn Sie diese Operation planen, stellen Sie sicher, dass Sie zuvor Ihren clientseitigen Vertrauensspeicher aktualisiert haben.

Example

Das folgende Beispiel ändert `mydbinstance`, indem das CA-Zertifikat auf `rds-ca-rsa2048-g1` festgelegt wird.

Für Linux, macOS oder Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

Für Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

> **Note**
>
> Wenn Ihre Instance neu gestartet werden muss, können Sie den modify-db-instanceCLI-Befehl verwenden und die `--no-certificate-rotation-restart` Option angeben.

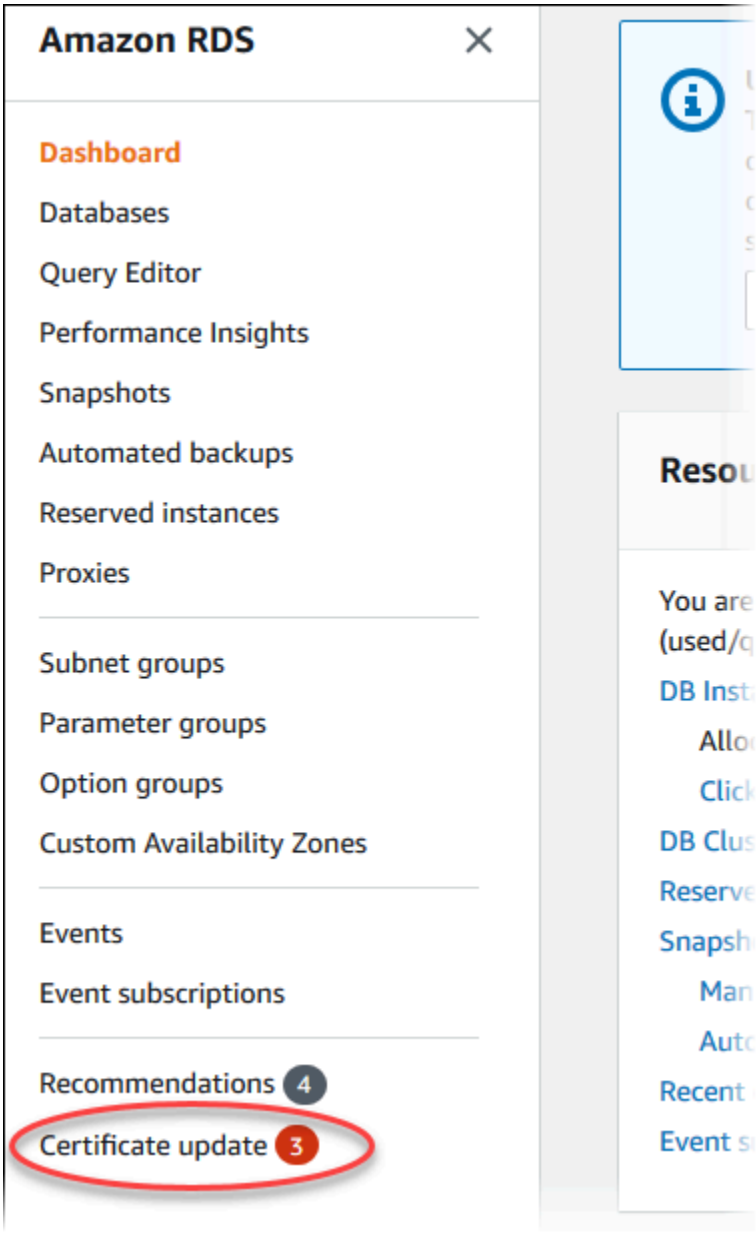
Aktualisieren des CA-Zertifikats durch Anwenden der Wartung

Führen Sie die folgenden Schritte aus, um Ihr CA-Zertifikat durch Anwendung der Wartung zu aktualisieren.

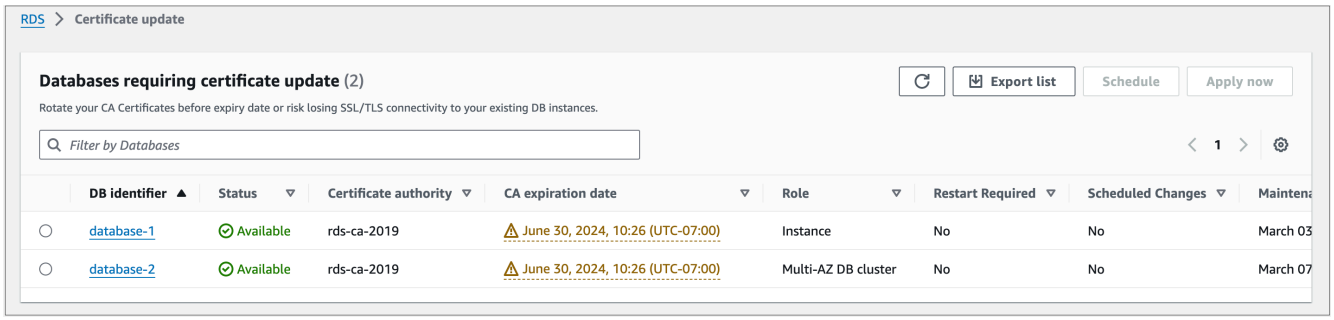
Console

So aktualisieren Sie Ihr CA-Zertifikat durch Anwendung der Wartung

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich die Option Zertifikataktualisierung.



Die Seite Datenbanken, die eine Zertifikataktualisierung erfordern wird angezeigt.



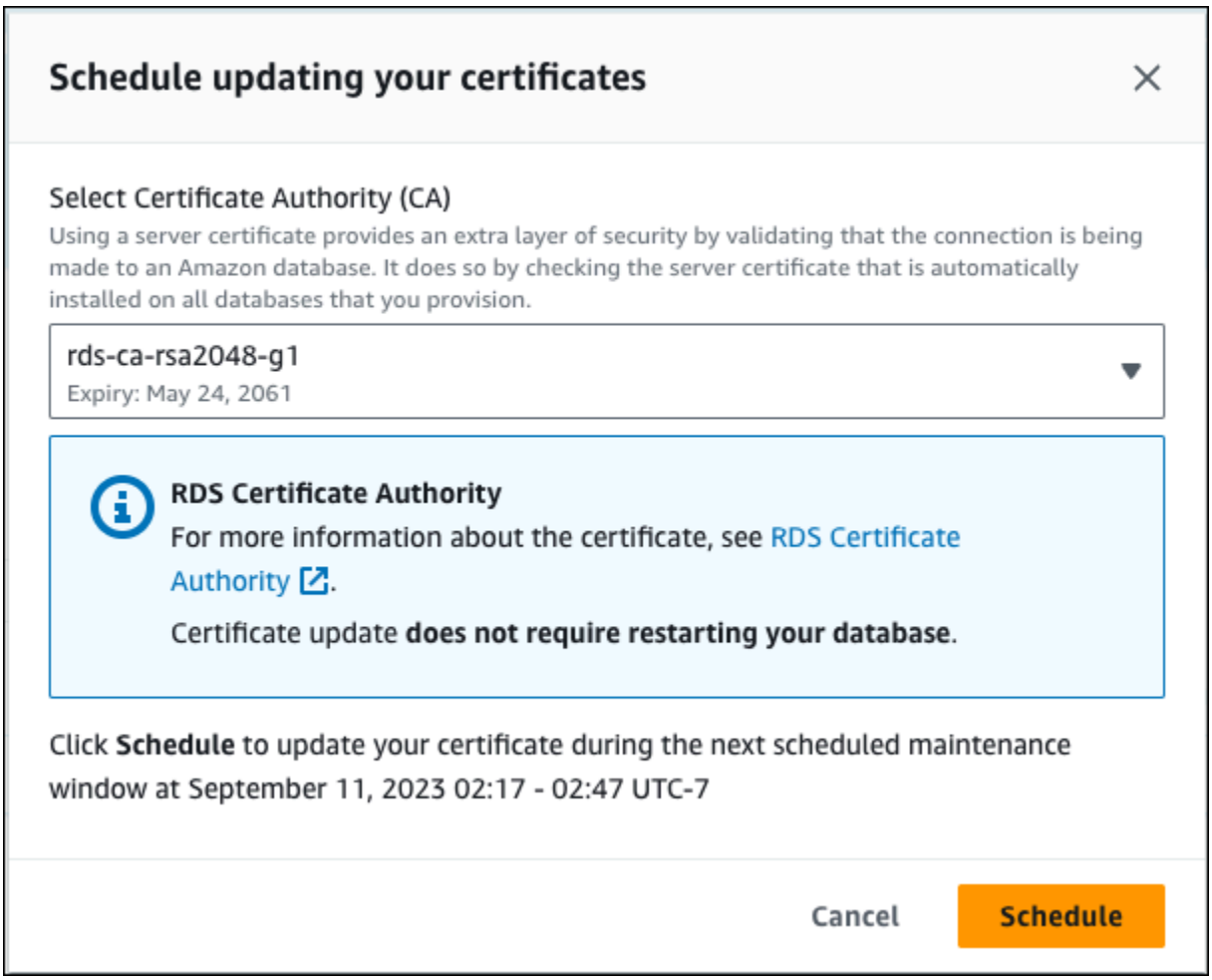
> **Note**
>
> Auf dieser Seite werden nur die aktuellen DB-Instances angezeigt AWS-Region. Wenn Sie Datenbanken in mehr als einer haben AWS-Region, überprüfen Sie diese Seite auf jeder Seite, AWS-Region um alle DB-Instances mit alten SSL/TLS Zertifikaten zu sehen.

3. Wählen Sie die DB-Instance für die Aktualisierung aus.

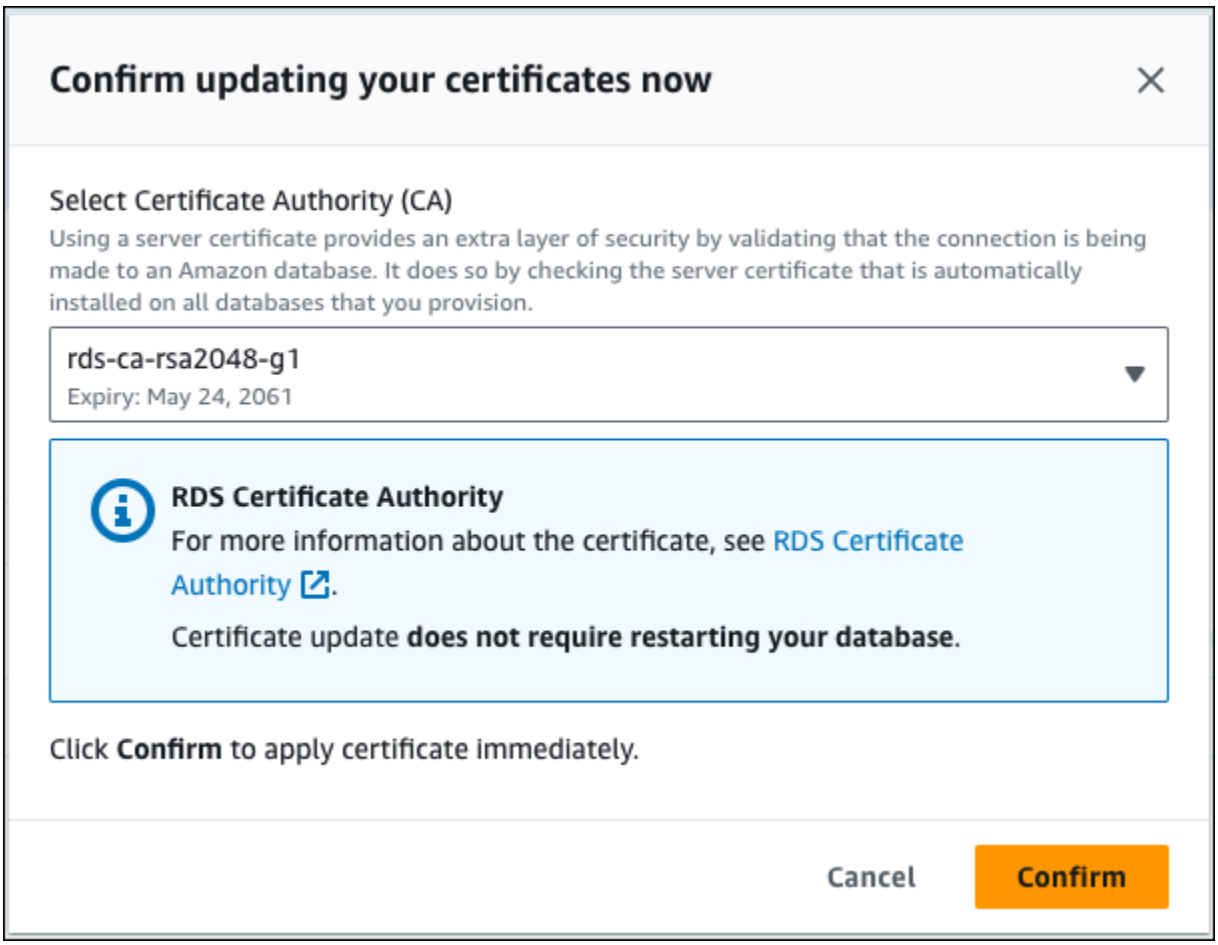
   Sie können die Zertifikatrotation für das nächste Wartungsfenster planen, indem Sie Zeitplan wählen. Wenden Sie die Rotation sofort an, indem Sie Jetzt anwenden wählen.

> **Important**
>
> Wenn nach Ablauf des Zertifikats Verbindungsprobleme auftreten, verwenden Sie die Option Jetzt anwenden.

4. a. Wenn Sie Zeitplan wählen, werden Sie aufgefordert, die Rotation der CA-Zertifikate zu bestätigen. In dieser Aufforderung wird auch das geplante Fenster für das Update angegeben.



b. Wenn Sie Jetzt anwenden wählen, werden Sie aufgefordert, die Rotation der CA-Zertifikate zu bestätigen.



> **Important**
>
> Bevor Sie die Rotation der CA-Zertifikate für Ihre Datenbank planen, aktualisieren Sie alle Client-Anwendungen, die das Serverzertifikat für die Verbindung verwenden SSL/TLS . Diese Updates sind spezifisch für Ihre DB-Engine. Nachdem Sie diese Clientanwendungen aktualisiert haben, können Sie die Rotation des CA-Zertifikats bestätigen.

Aktivieren Sie das Kontrollkästchen und klicken Sie dann auf Confirm (Bestätigen), um fortzufahren.

5. Wiederholen Sie Schritt 3 und 4 für alle DB-Instances, die aktualisiert werden sollen.

Automatische Rotation von Serverzertifikaten

Wenn Ihre Root-CA die automatische Rotation von Serverzertifikaten unterstützt, übernimmt RDS automatisch die Rotation des DB-Serverzertifikats. RDS verwendet dieselbe Stamm-CA für diese automatische Rotation, sodass Sie kein neues CA-Paket herunterladen müssen. Siehe Zertifizierungsstellen.

Die Rotation und Gültigkeit Ihres DB-Serverzertifikats hängen von Ihrer DB-Engine ab:

- Wenn Ihre DB-Engine die Rotation ohne Neustart unterstützt, rotiert RDS das DB-Serverzertifikat automatisch, ohne dass Sie etwas unternehmen müssen. RDS versucht, Ihr DB-Serverzertifikat in Ihrem bevorzugten Wartungsfenster nach der Halbwertszeit des DB-Serverzertifikats zu rotieren. Das neue DB-Serverzertifikat ist 12 Monate lang gültig.
- Wenn Ihre DB-Engine die Rotation ohne Neustart nicht unterstützt, macht Amazon RDS während der Halbwertszeit des Zertifikats oder mindestens 3 Monate vor Ablauf eine `server-certificate-rotation` ausstehende Wartungsaktion per Describe-pending-maintenance-actions API sichtbar. Sie können die Rotation mithilfe der apply-pending-maintenance-action API anwenden. Das neue DB-Serverzertifikat ist 36 Monate lang gültig.

Verwenden Sie den describe-db-engine-versionsBefehl und überprüfen Sie das `SupportsCertificateRotationWithoutRestart` Flag, um festzustellen, ob die DB-Engine-Version das Rotieren des Zertifikats ohne Neustart unterstützt. Weitere Informationen finden Sie unter Einstellung der CA für Ihre Datenbank.

Beispielskript für den Import von Zertifikaten in Ihren Trust Store

Nachfolgend sind Beispiel-Shell-Skripte aufgeführt, die das Zertifikatspaket in einen Vertrauensspeicher importieren.

Jedes Beispiel-Shell-Skript verwendet keytool, das Teil des Java Development Kits (JDK) ist. Weitere Informationen über die Installation von JDK finden Sie im JDK-Installationshandbuch.

Linux

Nachfolgend finden Sie ein Beispiel-Shell-Skript, das das Zertifikatpaket in einen Trust Store auf einem Linux-Betriebssystem importiert.

```
mydir=tmp/certs
```

```
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

macOS

Es folgt ein Beispiel für ein Shell-Skript, das das Zertifikatspaket in einen Vertrauensspeicher unter macOS importiert.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
```

```
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

Weitere bewährte Methoden zur Verwendung von SSL mit Amazon RDS finden Sie unter Best practices for successful SSL connections to Amazon RDS für Oracle.

Richtlinie für den Datenverkehr zwischen Netzwerken

Verbindungen werden zwischen Amazon Aurora und On-Premises-Anwendungen geschützt sowie zwischen Amazon Aurora und anderen AWS-Ressourcen innerhalb derselben AWS-Region.

Datenverkehr zwischen Service und On-Premises-Clients und -Anwendungen

Sie haben zwei Verbindungsoptionen zwischen Ihrem privaten Netzwerk und AWS:

- AWS Site-to-Site VPN-Verbindung Weitere Informationen finden Sie unter Was ist AWS Site-to-Site VPN?
- Direct Connect-Verbindung Weitere Informationen finden Sie unter Was ist Direct Connect?

Sie erhalten über das Netzwerk Zugriff auf Amazon Aurora, indem Sie von AWS veröffentlichte API-Operationen verwenden. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.

- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Identity and Access Management für Amazon Aurora

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf AWS Ressourcen sicher zu kontrollieren. IAM-Administratoren steuern, wer authenticated (angemeldet) und authorized (autorisiert) (im Besitz von Berechtigungen) ist, um Amazon-RDS-Ressourcen zu nutzen. IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe
- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- Funktionsweise von Amazon Aurora mit IAM
- Beispiele für identitätsbasierte Amazon-Aurora-Richtlinien
- AWS-verwaltete Richtlinien für Amazon RDS
- Änderungen von Amazon RDS an von AWS verwalteten Richtlinien
- Vermeidung des dienstübergreifenden Confused-Deputy-Problems
- IAM-Datenbankauthentifizierung
- Fehlerbehebung für Amazon Aurora-Identität und -Zugriff

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, die Sie in Aurora ausführen.

Service-Benutzer – Wenn Sie den Aurora-Service verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen bereit. Wenn Sie für Ihre Arbeit weitere Aurora-Funktionen ausführen, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle verstehen, kann Ihnen dies helfen, die richtigen Berechtigungen von Ihrem Administrator anzufordern. Unter Fehlerbehebung für Amazon Aurora-Identität und -Zugriff finden Sie nützliche Informationen für den Fall, dass Sie keinen Zugriff auf eine Funktion in Aurora haben.

Service-Administrator – Wenn Sie in Ihrem Unternehmen die Verantwortung für Aurora-Ressourcen haben, haben Sie wahrscheinlich vollständigen Zugriff auf Aurora. Ihre Aufgabe besteht darin, die Aurora-Funktionen und -Ressourcen festzulegen, auf die Mitarbeiter zugreifen können sollten. Sie müssen anschließend bei Ihrem -Administrator entsprechende Änderungen für die Berechtigungen Ihrer Service-Benutzer anfordern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen dazu, wie Ihr Unternehmen IAM mit Aurora verwenden kann, finden Sie unter Funktionsweise von Amazon Aurora mit IAM.

Administrator – Wenn Sie als Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf Aurora verfassen können. Beispiele für identitätsbasierte Aurora-Richtlinien, die Sie in IAM verwenden können, finden Sie unter Beispiele für identitätsbasierte Amazon-Aurora-Richtlinien.

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS Konto (Root-Benutzer)

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen aus einer Identitätsquelle zugreift. Verbundidentitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Verwenden Sie möglichst temporäre Anmeldeinformationen statt IAM-Benutzer mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu können.

Eine IAM-Gruppe gibt eine Sammlung von IAM-Benutzern an und vereinfacht die Verwaltung von Berechtigungen bei großer Benutzerzahl. Weitere Informationen finden Sie unter Use cases for IAM users im IAM-Benutzerhandbuch.

Sie können sich mit der IAM-Datenbankauthentifizierung bei Ihrem DB--Cluster authentifizieren.

Die IAM-Datenbank-Authentifizierung funktioniert mit Aurora. Weitere Informationen zur Authentifizierung bei Ihrem DB--Cluster mit IAM finden Sie unter IAM-Datenbankauthentifizierung.

IAM-Rollen

Eine IAM-Rolle ist eine Identität innerhalb Ihres Unternehmens, für AWS-Konto die bestimmte Berechtigungen gelten. Sie ist mit einem Benutzer vergleichbar, jedoch nicht mit einer bestimmten Person verknüpft. Sie können vorübergehend eine IAM-Rolle in der übernehmen, AWS-Managementkonsole indem Sie die Rollen wechseln. Sie können eine Rolle übernehmen, indem Sie eine AWS CLI oder AWS API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter Verwenden von IAM-Rollen im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- Temporäre Benutzerberechtigungen – Ein Benutzer kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- Verbundbenutzerzugriff – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter Erstellen von Rollen für externe Identitätsanbieter (Verbund) im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter Berechtigungssätze im AWS IAM Identity Center -Benutzerhandbuch.
- Kontoübergreifender Zugriff – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS-Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien im IAM-Benutzerhandbuch.
- Serviceübergreifender Zugriff — Einige AWS-Services verwenden Funktionen in anderen AWS-Services. Wenn Sie beispielsweise in einem Service einen Anruf tätigen, ist es üblich, dass dieser Service Anwendungen in Amazon ausführt EC2 oder Objekte in Amazon S3 speichert. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
  - Forward-Access-Sitzungen — Forward-Access-Sitzungen (FAS) verwenden die Berechtigungen des Principals, der einen aufruft AWS-Service, kombiniert mit der Anfrage, Anfragen AWS-Service an nachgelagerte Dienste zu stellen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.
  - Servicerolle – Eine Servicerolle ist eine IAM-Rolle, die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.
  - Dienstbezogene Rolle — Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem

Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

- Auf Amazon ausgeführte Anwendungen EC2 — Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2 Instance ausgeführt werden und AWS API-Anfragen stellen AWS CLI . Dies ist dem Speichern von Zugriffsschlüsseln innerhalb der EC2 Instance vorzuziehen. Um einer EC2 Instanz eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie ein Instanzprofil, das an die Instanz angehängt ist. Ein Instanzprofil enthält die Rolle und ermöglicht Programmen, die auf der EC2 Instanz ausgeführt werden, temporäre Anmeldeinformationen abzurufen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Verwenden einer IAM-Rolle, um Berechtigungen für Anwendungen zu gewähren, die auf EC2 Amazon-Instances ausgeführt werden.

Informationen dazu, wann Sie IAM-Rollen verwenden sollten, finden Sie unter Wann Sie eine IAM-Rolle (statt eines Benutzers) erstellen sollten im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an IAM-Identitäten oder -Ressourcen anhängen. AWS Eine Richtlinie ist ein Objekt, AWS das, wenn es einer Identität oder Ressource zugeordnet ist, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn eine Entität (Root-Benutzer, Benutzer oder IAM-Rolle) eine Anfrage stellt. Die Berechtigungen in den Richtlinien legen fest, ob eine Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Ein Administrator kann mithilfe von Richtlinien angeben, wer Zugriff auf AWS Ressourcen hat und welche Aktionen er mit diesen Ressourcen ausführen kann. Eine IAM-Entität (Berechtigungssatz oder Rolle) besitzt zunächst keine Berechtigungen. Anders ausgedrückt, können Benutzer standardmäßig keine Aktionen ausführen und nicht einmal ihr Passwort ändern. Um einem Benutzer die Berechtigung für eine Aktion zu erteilen, muss ein Administrator einem Benutzer eine Berechtigungsrichtlinie zuweisen. Alternativ kann der Administrator den Benutzer zu einer Gruppe hinzufügen, die über die gewünschten Berechtigungen verfügt. Wenn ein Administrator einer Gruppe Berechtigungen erteilt, erhalten alle Benutzer in dieser Gruppe diese Berechtigungen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Rolleninformationen von der AWS-Managementkonsole AWS CLI, der oder der AWS API abrufen.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. ein Berechtigungssatz oder eine Rolle. Diese Richtlinien steuern, welche Aktionen diese Identität für welche Ressourcen und unter welchen Bedingungen ausführen kann. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Erstellen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Berechtigungssatz oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Berechtigungssätzen und Rollen in Ihrem AWS Konto zuordnen können. Zu den verwalteten Richtlinien gehören AWS verwaltete Richtlinien und vom Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer eingebundenen Richtlinie wählen, finden Sie unter Auswahl zwischen verwalteten und eingebundenen Richtlinien im IAM-Benutzerhandbuch.

Informationen zu AWS verwalteten Richtlinien, die speziell für Aurora gelten, finden Sie unterAWS-verwaltete Richtlinien für Amazon RDS.

Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger verbreitete Richtlinientypen. Mit diesen Richtlinientypen können Sie die maximalen Berechtigungen festlegen, die Ihnen durch die gängigeren Richtlinientypen gewährt werden.

- Berechtigungsgrenzen – Eine Berechtigungsgrenze ist eine erweiterte Funktion, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (Berechtigungssatz oder Rolle) erteilen kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen stellen die Schnittmenge zwischen den identitätsbasierten Richtlinien der Entität und ihren Berechtigungsgrenzen dar. Ressourcenbasierte Richtlinien, die den Berechtigungssatz oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über

Berechtigungsgrenzen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im IAM-Benutzerhandbuch.

- Dienststeuerungsrichtlinien (SCPs) — SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OU) festlegen. AWS Organizations AWS Organizations ist ein Dienst zur Gruppierung und zentralen Verwaltung mehrerer AWS Konten, die Ihrem Unternehmen gehören. Wenn Sie alle Funktionen in einer Organisation aktivieren, können Sie Richtlinien zur Servicesteuerung (SCPs) auf einige oder alle Ihre Konten anwenden. Das SCP schränkt die Berechtigungen für Entitäten in Mitgliedskonten ein, einschließlich der einzelnen Root-Benutzer des AWS-Kontos Entitäten. Weitere Informationen zu Organizations und SCPs finden Sie unter So SCPs arbeiten Sie im AWS Organizations Benutzerhandbuch.
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Berechtigungssatzes oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die sich daraus ergebenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Funktionsweise von Amazon Aurora mit IAM

Bevor Sie IAM zum Verwalten des Zugriffs auf Amazon Aurora verwenden, sollten Sie verstehen, welche IAM-Funktionen für die Verwendung mit Aurora verfügbar sind.

In der folgenden Tabelle sind die IAM-Funktionen aufgelistet, die mit Amazon Aurora verwendet werden können:

IAM-Feature	Unterstützung von Amazon Aurora
Identitätsbasierte Richtlinien	Ja

IAM-Feature	Unterstützung von Amazon Aurora
Ressourcenbasierte Richtlinien	Nein
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Richtlinienbedingungsschlüssel (services pezifisch)	Ja
ACLs	Nein
Attributbasierte Zugriffssteuerung (Attribute-Based Access Control, ABAC) (Tags in Richtlinien)	Ja
Temporäre Anmeldeinformationen	Ja
Forward Access Sessions	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Ja

Einen allgemeinen Überblick darüber, wie , Amazon Aurora und andere AWS Services mit IAM zusammenarbeiten, finden Sie im IAM-Benutzerhandbuch unter AWS Services, die mit IAM funktionieren.

Themen

- Identitätsbasierte Aurora-Richtlinien
- Ressourcenbasierte Richtlinien in Aurora
- Richtlinienaktionen für Aurora
- Richtlinienressourcen für Aurora
- Richtlinien-Bedingungsschlüssel für Aurora
- Zugriffskontrolllisten (ACLs) in Aurora

- Attributbasierte Zugriffssteuerung (Attribute-Based Access Control, ABAC) in Richtlinien mit Aurora-Tags
- Verwenden temporärer Anmeldeinformationen mit Aurora
- Forward Access Sessions für Aurora
- Servicerollen für Aurora
- Serviceverknüpfte Rollen für Aurora

Identitätsbasierte Aurora-Richtlinien

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Aurora-Richtlinien

Beispiele für identitätsbasierte Aurora-Richtlinien finden Sie unter Beispiele für identitätsbasierte Amazon-Aurora-Richtlinien.

Ressourcenbasierte Richtlinien in Aurora

Unterstützt ressourcenbasierte Richtlinien: Nein

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen

in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Richtlinienaktionen für Aurora

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Richtlinienaktionen in Aurora verwenden das folgende Präfix vor der Aktion: `rds:`. Um beispielsweise jemandem die Berechtigung zu erteilen, DB-Instances mit der API-Operation Amazon RDS `DescribeDBInstances` zu beschreiben, nehmen Sie die Aktion `rds:DescribeDBInstances` in die Richtlinie auf. Richtlinienanweisungen müssen entweder ein `Action`- oder ein `NotAction`-Element enthalten. Aurora definiert eine eigene Gruppe von Aktionen, die Aufgaben beschreiben, die Sie mit diesem Service durchführen können.

Um mehrere -Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie folgendermaßen durch Kommas.

```
"Action": [
      "rds:action1",
      "rds:action2"
```

Sie können auch Platzhalter (*) verwenden, um mehrere Aktionen anzugeben. Beispielsweise können Sie alle Aktionen festlegen, die mit dem Wort `Describe` beginnen, einschließlich der folgenden Aktion:

```
"Action": "rds:Describe*"
```

Um eine Liste von Aurora-Aktionen finden Sie unter Von Amazon RDS definierte Aktionen in der Service-Autorisierungs-Referenz

Richtlinienressourcen für Aurora

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als bewährte Methode geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Die DB-Instance-Ressource hat den folgenden Amazon-Ressourcennamen (ARN).

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

Weitere Informationen zum Format von ARNs finden Sie unter Amazon Resource Names (ARNs) und AWS Service Namespaces.

Wenn Sie beispielsweise die `dbtest`-DB-Instance in Ihrer Anweisung angeben möchten, verwenden Sie den folgenden ARN.

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

Wenn Sie alle DB-Instances angeben möchten, die einem bestimmten Konto angehören, verwenden Sie den Platzhalter (*).

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

Einige RDS-API-Operationen, z. B. das Erstellen von Ressourcen, können nicht für eine bestimmte Ressource durchgeführt werden. Verwenden Sie in diesen Fällen den Platzhalter (*).

```
"Resource": "*"
```

Viele Amazon-RDS-API-Operationen umfassen mehrere Ressourcen. `CreateDBInstance` erstellt beispielsweise eine DB-Instance. Sie können festlegen, dass ein -Benutzer beim Erstellen einer DB-Instance eine bestimmte Sicherheitsgruppe und Parametergruppe verwenden muss. Um mehrere Ressourcen in einer einzigen Anweisung anzugeben, trennen Sie sie durch Kommas. ARNs

```
"Resource": [
      "resource1",
      "resource2"
```

Eine Liste der Aurora-Ressourcentypen und ihrer Eigenschaften ARNs finden Sie unter Von Amazon RDS definierte Ressourcen in der Service Authorization Reference. Informationen zu den Aktionen, mit denen Sie den ARN einzelner Ressourcen angeben können, finden Sie unter Von Amazon RDS definierte Aktionen.

Richtlinien-Bedingungsschlüssel für Aurora

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das `Condition`-Element legt fest, ob Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Aurora definiert einen eigenen Satz von Bedingungsschlüsseln und unterstützt auch einige globale Bedingungsschlüssel. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Alle RDS-API-Operationen unterstützen den Bedingungsschlüssel `aws:RequestedRegion`.

Um eine Liste von Aurora-Bedingungsschlüsseln finden Sie unter Bedingungsschlüssel für Amazon RDS in der Service-Autorisierungs-Referenz. Informationen dazu, mit welchen Aktionen und Ressourcen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Von Amazon RDS definierte Aktionen.

Zugriffskontrolllisten (ACLs) in Aurora

Unterstützt Zugriffskontrolllisten (ACLs): Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) Berechtigungen für den Zugriff auf eine Ressource haben. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Attributbasierte Zugriffssteuerung (Attribute-Based Access Control, ABAC) in Richtlinien mit Aurora-Tags

Unterstützt Tags für die attributbasierte Zugriffssteuerung (Attribute-Based Access Control, ABAC) in Richtlinien: Ja

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen, auch als Tags bezeichnet, definiert werden. Sie können Tags an IAM-Entitäten und AWS Ressourcen anhängen und dann ABAC-Richtlinien so entwerfen, dass Operationen möglich sind, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter Definieren von Berechtigungen mit ABAC-Autorisierung im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe Attributbasierte Zugriffskontrolle (ABAC) verwenden im IAM-Benutzerhandbuch.

Weitere Informationen über das Markieren von Aurora-Ressourcen mit Tags finden Sie unter Festlegen von Bedingungen: Verwenden von benutzerdefinierten Tags. Ein Beispiel für eine identitätsbasierte Richtlinie zur Einschränkung des Zugriffs auf eine Ressource auf der Grundlage der Tags dieser Ressource finden Sie unter Erteilen von Berechtigungen für Aktionen in einer Ressource mit einem bestimmten Tag und zwei verschiedenen Tag-Werten.

Verwenden temporärer Anmeldeinformationen mit Aurora

Unterstützt temporäre Anmeldeinformationen: Ja

Temporäre Anmeldeinformationen ermöglichen den kurzfristigen Zugriff auf AWS Ressourcen und werden automatisch erstellt, wenn Sie einen Verbund verwenden oder die Rollen wechseln. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporary security credentials in IAM und AWS-Services that work with IAM im IAM-Benutzerhandbuch.

Forward Access Sessions für Aurora

Unterstützt Forward Access Sessions (FAS): Ja

Forward Access Sessions (FAS) verwenden die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, kombiniert mit der Anforderung, Anfragen AWS-Service an nachgeschaltete Dienste zu stellen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter Zugriffssitzungen weiterleiten.

Servicerollen für Aurora

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine IAM-Rolle, die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

> **Warning**
>
> Das Ändern der Berechtigungen für eine Servicerolle könnte die Aurora-Funktionalität beeinträchtigen. Bearbeiten Sie Servicerollen nur, wenn Aurora dazu Anleitungen gibt.

Serviceverknüpfte Rollen für Aurora

Unterstützt serviceverknüpfte Rollen: Ja

Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

Details zum Verwenden von serviceverknüpften Aurora-Rollen finden Sie unter Verwenden von serviceverknüpften Rollen für Amazon Aurora.

Beispiele für identitätsbasierte Amazon-Aurora-Richtlinien

Standardmäßig besitzen Berechtigungssätze und Rollen keine Berechtigungen zum Erstellen oder Ändern von Aurora-Ressourcen. Sie können auch keine Aufgaben mit der AWS-Managementkonsole AWS CLI, oder AWS API ausführen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Berechtigungssätzen und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den Berechtigungssätzen oder Rollen anfügen, die diese Berechtigungen benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von Richtlinien auf der JSON-Registerkarte im IAM-Benutzerhandbuch.

Themen

- Bewährte Methoden für Richtlinien
- Verwenden der Aurora-Konsole
- Erforderliche Berechtigungen für die Verwendung der Konsole
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer
- Berechtigungsrichtlinien zum Erstellen, Ändern und Löschen von Ressourcen in Aurora
- Beispielrichtlinien: Verwenden von Bedingungsschlüsseln
- Festlegen von Bedingungen: Verwenden von benutzerdefinierten Tags
- Berechtigung zum Markieren von Aurora-Ressourcen während der Erstellung erteilen

Bewährte Methoden für Richtlinien

Identitätsbasierte Richtlinien können festlegen, ob jemand Amazon-RDS-Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder daraus löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Wenn Sie identitätsbasierte Richtlinien erstellen oder bearbeiten, befolgen Sie diese Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien

definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter AWS -verwaltete Richtlinien oder AWS -verwaltete Richtlinien für Auftrags-Funktionen im IAM-Benutzerhandbuch.

- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwenden der Aurora-Konsole

Um auf die Amazon Aurora-Konsole zuzugreifen, müssen Sie über einen Mindestsatz von Berechtigungen verfügen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den Aurora Aurora-Ressourcen in Ihrem aufzulisten und anzuzeigen AWS-Konto. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die Sie ausführen möchten.

Um sicherzustellen, dass diese Entitäten weiterhin die Aurora-Konsole verwenden können, fügen Sie den Entitäten auch die folgende AWS verwaltete Richtlinie hinzu.

```
AmazonRDSReadOnlyAccess
```

Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Erforderliche Berechtigungen für die Verwendung der Konsole

Damit ein Benutzer mit der Konsole arbeiten kann, muss dieser Benutzer über einen Minimumsatz an Berechtigungen verfügen. Diese Berechtigungen ermöglichen es dem Benutzer, die Aurora Aurora-Ressourcen für sein AWS Konto zu beschreiben und andere verwandte Informationen bereitzustellen, einschließlich EC2 Amazon-Sicherheits- und Netzwerkinformationen.

Wenn Sie eine IAM-Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Benutzer mit dieser IAM-Richtlinie. Um sicherzustellen, dass diese Benutzer die Konsole weiterhin verwenden können, fügen Sie dem Benutzer auch die verwaltete Richtlinie `AmazonRDSReadOnlyAccess` an. Einzelheiten dazu finden Sie unter Verwalten des Zugriffs mit Richtlinien.

Für Benutzer, die nur Aufrufe an die AWS CLI oder Amazon-RDS-API durchführen, müssen Sie keine Mindestberechtigungen in der Konsole erteilen.

Die folgende Richtlinie gewährt vollen Zugriff auf alle Aurora Aurora-Ressourcen für das AWS Root-Konto:

```
AmazonRDSFullAccess
```

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der API AWS CLI oder AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Berechtigungsrichtlinien zum Erstellen, Ändern und Löschen von Ressourcen in Aurora

Die folgenden Abschnitte enthalten Beispiele für Berechtigungsrichtlinien, mit denen der Zugriff auf Ressourcen gewährt und eingeschränkt werden kann:

Erlauben Sie einem Benutzer, DB-Instances in einem AWS Konto zu erstellen

Im Folgenden finden Sie ein Beispiel für eine Richtlinie, die es dem Konto mit der ID ermöglicht, DB-Instances für Ihr AWS Konto `123456789012` zu erstellen. Die Richtlinie setzt voraus, dass der Name der DB-Instance mit beginn `test`. Die neue DB-Instance muss auch die MySQL-Datenbank-Engine und die DB-Instance-Klasse `db.t2.micro` verwenden. Zusätzlich muss die neue DB-Instance eine Optionsgruppe und eine DB-Parametergruppe verwenden, die mit `default` beginnt und die Subnetzgruppe `default` verwendet.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:db:test*",
            "arn:aws:rds:*:123456789012:og:default*",
            "arn:aws:rds:*:123456789012:pg:default*",
            "arn:aws:rds:*:123456789012:subgrp:default"
         ],
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql",
               "rds:DatabaseClass": "db.t2.micro"
            }
         }
      }
   ]
```

```
}
```

Die Richtlinie ist ein einzelnes Statement, das die folgenden Berechtigungen für den -Benutzer bestimmt:

- Die Richtlinie ermöglicht es dem Konto, mithilfe des Vorgangs Create DBInstance API eine DB-Instance zu erstellen (dies gilt auch für den create-db-instance AWS CLI Befehl und den AWS-Managementkonsole).
- Das Element `Resource` gibt an, dass der Benutzer auf oder mit Ressourcen Aktionen ausführen kann. Sie geben Ressourcen über einen Amazon Resources Name (ARN) an. Dieser ARN umfasst den Namen des Dienstes, zu dem die Ressource gehört (`rds`), die AWS Region (`*`gibt in diesem Beispiel eine beliebige Region an), die AWS Kontonummer (`123456789012`ist in diesem Beispiel die Kontonummer) und den Ressourcentyp. Weitere Informationen zum Erstellen finden ARNs Sie unterAmazon-Ressourcennamen (ARN) in Amazon RDS.

  Das `Resource`-Element im Beispiel gibt für den Benutzer die folgenden richtlinienbezogenen Einschränkungen für die Ressourcen an:

  - Die DB-Instance-Kennung für die neue DB-Instance muss mit `test` beginnen (zum Beispiel `testCustomerData1`, `test-region2-data`).
  - Die Optionsgruppe für die neue DB-Instance muss mit beginne `default`.
  - Die DB-Parametergruppe für die neue DB-Instance muss mit beginne `default`.
  - Die Subnetzgruppe für die neue DB-Instance muss mit `default` beginnen.
- Das `Condition`-Element gibt an, dass die DB-Engine MySQL sein muss und die DB-Instance-Klasse `db.t2.micro` sein muss. Das `Condition`-Element bestimmt die Bedingungen, wann eine Richtlinie wirksam sein soll. Sie können zusätzliche Berechtigungen oder Einschränkungen hinzufügen, indem Sie das `Condition`-Element verwenden. Weitere Informationen zur Angabe von Bedingungen finden Sie unter Richtlinien-Bedingungsschlüssel für Aurora. Dieses Beispiel zeigt die Bedingungen `rds:DatabaseEngine` und `rds:DatabaseClass`. Informationen zu den gültigen Bedingungswerten für `rds:DatabaseEngine` finden Sie in der Liste unter dem `Engine` Parameter in Create DBInstance. Informationen zu den gültigen Bedingungswerten für `rds:DatabaseClass` finden Sie unter Unterstützte DB-Engines für DB-Instance-Klassen.

Das Element `Principal` ist in der Richtlinie nicht angegeben, da in identitätsbasierten Richtlinien die Angabe des Prinzipals als Empfänger der Berechtigung nicht erforderlich ist. Wenn Sie einem Benutzer eine Richtlinie zuweisen, ist der Benutzer automatisch der Prinzipal. Wird die

Berechtigungsrichtlinie einer IAM-Rolle zugewiesen, erhält der in der Vertrauensrichtlinie der Rolle angegebene Prinzipal die Berechtigungen.

Um eine Liste von Aurora-Aktionen finden Sie unter Von Amazon RDS definierte Aktionen in der Service-Autorisierungs-Referenz

Einem Benutzer eine beliebige Beschreibungsaktion für eine beliebige RDS-Ressource erlauben

Die folgende Berechtigungsrichtlinie gewährt Berechtigungen für einen Benutzer, alle Aktionen auszuführen, die mit beginne `Describe`. Diese Aktionen zeigen Informationen zu einer RDS-Ressource, z. B. eine DB-Instance. Das Platzhalterzeichen (*) im `Resource`-Element zeigt an, dass die Aktionen für alle Amazon Aurora-Ressourcen erlaubt sind, die dem Konto gehören.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

Einem Benutzer erlauben, eine DB-Instance zu erstellen, die spezifische DB-Parametergruppe und Subnetzgruppe verwendet.

Die folgenden Berechtigungsrichtlinien erteilen einem Benutzer die Erlaubnis, ausschließlich eine DB-Instance mit der DB-Parametergruppe `mydbpg` und der DB-Subnetzgruppe `mydbsubnetgroup` zu erstellen.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
```

```
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": [
            "arn:aws:rds:*:*:pg:mydbpg",
            "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
         ]
      }
   ]
}
```

Erteilen von Berechtigungen für Aktionen in einer Ressource mit einem bestimmten Tag und zwei verschiedenen Tag-Werten

Sie können in Ihrer identitätsbasierten Richtlinie Bedingungen für die Steuerung des Zugriffs auf Aurora-Ressourcen auf der Basis von Tags verwenden. Die folgende Richtlinie erteilt die Berechtigung, die API-Operation `CreateDBSnapshot` auf DB-Instances durchzuführen, bei denen das Tag `stage` entweder auf `development` oder `test` festgelegt ist.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
```

```
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      ]
}
```

Die folgende Richtlinie erteilt die Berechtigung, die API-Operation `ModifyDBInstance` auf DB-Instances durchzuführen, bei denen das Tag `stage` entweder auf `development` oder `test` festgelegt ist.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
```

```
                "development",
                "test"
            ]
         }
      }
   ]
}
```

Verhindern, dass ein Benutzer eine DB-Instance löscht

Die folgenden Berechtigungsrichtlinien erteilen Berechtigungen, um einen Benutzer davon abzuhalten, eine bestimmte DB-Instance zu löschen. Beispielsweise möchten Sie jedem Benutzer, der kein Administrator ist, verbieten, Ihre Produktions-DB-Instances zu löschen.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

Verweigern des gesamten Zugriffs auf eine Ressource

Sie können den Zugriff auf eine Ressource explizit verweigern. Verweigerungsrichtlinien haben Vorrang vor Zulassungsrichtlinien. Die folgende Richtlinie verweigert einem Benutzer explizit die Möglichkeit, eine Ressource zu verwalten:

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": "rds:*",
         "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
      }
   ]
}
```

Beispielrichtlinien: Verwenden von Bedingungsschlüsseln

Im Folgenden finden Sie Beispiele für die Verwendung von Bedingungsschlüsseln in den IAM-Berechtigungsrichtlinien von Amazon Aurora.

Beispiel 1: Erteilen der Berechtigung zum Erstellen einer DB-Instance mit einer bestimmten DB-Engine ohne Multi-AZ-Bereitstellung

Die folgende Richtlinie nutzt einen RDS-Bedingungsschlüssel und legt fest, dass ein Benutzer nur DB-Instances erstellen darf, die die MySQL-Datenbank-Engine und keine Multi-AZ-Bereitstellung verwenden. Das Element `Condition` gibt die Voraussetzung an, dass es sich um eine MySQL-Datenbank-Engine handeln muss.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
```

```
            "Bool": {
               "rds:MultiAz": false
            }
         }
      }
   ]
}
```

Beispiel 2: Explizites Verweigern der Berechtigung, DB-Instances für bestimmte DB-Instance-Klassen sowie DB-Instances, die bereitgestellte IOPS verwenden, zu erstellen

Die folgende Richtlinie verweigert explizit die Berechtigung, DB-Instances für die DB-Instance-Klassen `r3.8xlarge` und `m4.10xlarge` zu erstellen, die zu den größten und teuersten DB-Instance-Klassen gehören. Diese Richtlinie hält Benutzer ebenfalls davon ab, DB-Instances zu erstellen, die bereitgestellte IOPS verwenden, durch die zusätzliche Kosten entstehen.

Eine explizit verweigerte Berechtigung überschreibt alle anderen erteilten Berechtigungen. So wird sichergestellt, dass Identitäten nicht aus Versehen eine Berechtigung erhalten, die Sie nie erteilen wollten.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
```

```
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
               "rds:Piops": "0"
            }
         }
      }
   ]
}
```

Beispiel 3: Einschränken des Satzes von Tag-Schlüsseln und Werten, mit dem eine Ressource mit einem Tag versehen werden kann

Die folgende Richtlinie verwendet einen RDS-Bedingungsschlüssel und erlaubt es, ein Tag mit dem Schlüssel `stage` einer Ressource mit den Werten `test`, `qa` und `production` hinzuzufügen.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagEdits",
      "Effect": "Allow",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource": "arn:aws:rds:us-east-1:123456789012:db:db-123456",
      "Condition": {
        "StringEquals": {
          "rds:req-tag/stage": [
            "test",
            "qa",
            "production"
          ]
        }
      }
```

```
    }
  ]
}
```

Festlegen von Bedingungen: Verwenden von benutzerdefinierten Tags

Amazon Aurora bietet Unterstützung für das Festlegen von Bedingungen in einer IAM-Richtlinie mithilfe von benutzerdefinierten Tags.

Angenommen, Sie fügen Ihren DB-Instances ein Tag namens `environment` mit Werten wie `beta`, `staging`, `production` und so weiter hinzu. Wenn dies der Fall ist, können Sie eine Richtlinie erstellen, die bestimmte Benutzer basierend auf dem Tag-Wert `environment` auf DB-Instances beschränkt.

> **Note**
>
> Bei benutzerdefinierten Tag-Kennungen muss auf Groß- und Kleinschreibung geachtet werden.

In der folgenden Tabelle werden die RDS-Tag-Kennungen aufgeführt, die Sie in einem `Condition`-Element verwenden können.

RDS-Tag-ID	Gilt für
`db-tag`	DB-Instances einschließlich Lesereplikaten
`snapshot-tag`	DB-Snapshots
`ri-tag`	Reservierte DB-Instances
`og-tag`	DB-Optionsgruppen
`pg-tag`	DB-Parametergruppen
`subgrp-tag`	DB-Subnetzgruppen
`es-tag`	Ereignisabonnements

RDS-Tag-ID	Gilt für
`cluster-tag`	DB-Cluster
`cluster-pg-tag`	DB-Cluster-Parametergruppen
`cluster-snapshot-tag`	DB-Cluster-Snapshots

Die Syntax für eine benutzerdefinierte Tag-Bedingung sieht wie folgt aus:

`"Condition":{"StringEquals":{"rds:`*`rds-tag-identifier`*`/`*`tag-name`*`": ["`*`value`*`"]} }`

Beispielsweise gilt das folgende `Condition`-Element für DB-Instances mit dem Tag `environment` und dem Tag-Wert `production`.

`"Condition":{"StringEquals":{"rds:db-tag/`**`environment`**`": ["`**`production`**`"]} }`

Weitere Informationen über die Erstellung von Tags finden Sie unter Taggen von Amazon-Aurora- und Amazon-RDS-Ressourcen.

Important

Wenn Sie den Zugriff auf Ihre RDS-Ressourcen mithilfe von Tags verwalten, empfehlen wir, den Zugriff auf die Tags Ihrer RDS-Ressourcen zu sichern. Sie können den Zugriff auf Tags verwalten, indem Sie Richtlinien für die Aktionen `AddTagsToResource` und `RemoveTagsFromResource` erstellen. Beispielsweise verweigert die folgende Richtlinie den Benutzern das Hinzufügen und Entfernen von Tags für alle Ressourcen. Sie können anschließend Richtlinien erstellen, die es bestimmten Benutzern ermöglichen, Tags hinzuzufügen oder zu entfernen.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyTagUpdates",
         "Effect":"Deny",
```

```
            "Action":[
               "rds:AddTagsToResource",
               "rds:RemoveTagsFromResource"
            ],
            "Resource":"*"
         }
      ]
}
```

Um eine Liste von Aurora-Aktionen finden Sie unter Von Amazon RDS definierte Aktionen in der Service-Autorisierungs-Referenz

Beispielrichtlinien: Verwenden von benutzerdefinierten Tags

Im Folgenden finden Sie Beispiele für die Verwendung von benutzerdefinierten Tags in den IAM-Berechtigungsrichtlinien in Amazon Aurora. Weitere Informationen zum Hinzufügen von Tags zu einer Amazon Aurora-Ressource finden Sie unter Amazon-Ressourcennamen (ARN) in Amazon RDS.

> **Note**
>
> Alle Beispiele verwenden die Region US-West-2 und enthalten ein fiktives Konto. IDs

Beispiel 1: Erteilen von Berechtigungen für Aktionen in einer Ressource mit einem bestimmten Tag und zwei verschiedenen Tag-Werten

Die folgende Richtlinie erteilt die Berechtigung, die API-Operation `CreateDBSnapshot` auf DB-Instances durchzuführen, bei denen das Tag `stage` entweder auf `development` oder `test` festgelegt ist.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
```

```
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Die folgende Richtlinie erteilt die Berechtigung, die API-Operation `ModifyDBInstance` auf DB-Instances durchzuführen, bei denen das Tag `stage` entweder auf `development` oder `test` festgelegt ist.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
            ],
          "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
```

```
            "arn:aws:rds:*:123456789012:og:*"
            ]
      },
      {
        "Sid":"AllowDevTestToModifyInstance",
        "Effect":"Allow",
        "Action":[
           "rds:ModifyDBInstance"
           ],
        "Resource":"arn:aws:rds:*:123456789012:db:*",
        "Condition":{
           "StringEquals":{
              "rds:db-tag/stage":[
                 "development",
                 "test"
                 ]
              }
           }
      }
   ]
}
```

Bespiel 2: Explizites Verweigern der Berechtigung zum Erstellen einer DB-Instance, die bestimmte DB-Parametergruppen verwendet

Die folgende Richtlinie verweigert explizit die Berechtigung, eine DB-Instance zu erstellen, die DB-Parametergruppen mit bestimmten Tag-Werten verwendet. Sie können diese Richtlinie anwenden, wenn stets eine bestimmte kundenseitig erstellte DB-Parametergruppe beim Erstellen von DB-Instances verwendet werden muss. Die Richtlinien mit `Deny` werden meist genutzt, um den von einer allgemeineren Richtlinie erteilten Zugriff einzuschränken.

Eine explizit verweigerte Berechtigung überschreibt alle anderen erteilten Berechtigungen. So wird sichergestellt, dass Identitäten nicht aus Versehen eine Berechtigung erhalten, die Sie nie erteilen wollten.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
```

```
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
         "Condition":{
            "StringEquals":{
               "rds:pg-tag/usage":"prod"
            }
         }
      }
   ]
}
```

Beispiel 3: Erteilen von Berechtigungen für Aktionen auf einer DB-Instance mit einem Instance-Namen, der den Benutzernamen als Präfix enthält

Die folgende Richtlinie erteilt die Berechtigung, eine beliebige API (mit Ausnahme von `AddTagsToResource` oder `RemoveTagsFromResource`) auf einer DB-Instance aufzurufen, deren Instance-Name den Benutzernamen als Präfix aufweist und das Tag `stage` mit dem Wert `devo` oder kein Tag `stage` enthält.

Die Zeile `Resource` in der Richtlinie kennzeichnet eine Ressource durch den Amazon-Ressourcennamen (ARN). Weitere Informationen zur Verwendung von Amazon Aurora-Ressourcen ARNs mit Amazon finden Sie unterAmazon-Ressourcennamen (ARN) in Amazon RDS.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
         "Condition":{
            "StringEqualsIfExists":{
```

```
                "rds:db-tag/stage":"devo"
            }
        }
    }
  ]
}
```

Berechtigung zum Markieren von Aurora-Ressourcen während der Erstellung erteilen

Einige RDS-API-Vorgänge erlauben es Ihnen, bei der Erstellung der Ressource Tags anzugeben. Sie können Resource-Tags (Markierungen) verwenden, um eine attributbasierte Steuerung (ABAC) zu implementieren. Weitere Informationen finden Sie unter Wozu dient ABAC? AWS und Steuern des Zugriffs auf AWS Ressourcen mithilfe von Tags.

Damit Benutzer Ressourcen bei der Erstellung kennzeichnen können, benötigen sie die Berechtigungen zum Verwenden der Aktion, die die Ressource wie `rds:CreateDBCluster` erstellt. Wenn Tags in der Erstellungsaktion angegeben werden, führt RDS eine zusätzliche Autorisierung für die Aktion `rds:AddTagsToResource` aus, um die Berechtigungen der Benutzer zum Erstellen von Tags zu überprüfen. Daher benötigen die Benutzer außerdem die expliziten Berechtigungen zum Verwenden der `rds:AddTagsToResource`-Aktion.

In der IAM-Richtliniendefinition für die `rds:AddTagsToResource`-Aktion können Sie mit dem Bedingungsschlüssel `aws:RequestTag` Tags in einer Anforderung zum Kennzeichnen einer Ressource anfordern.

Mithilfe der folgenden Richtlinie können Benutzer beispielsweise DB-Instances erstellen und Tags während der DB-Instance-Erstellung anwenden, jedoch nur mit bestimmten Tag-Schlüsseln (`environment` oder `project`):

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBInstance"
           ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rds:AddTagsToResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/environment": ["production", "development"],
                    "aws:RequestTag/project": ["dataanalytics", "webapp"]
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": ["environment", "project"]
                }
            }
        }
    ]
}
```

Diese Richtlinie lehnt jede Anforderung zum Erstellen einer DB-Instance ab, die andere Tags als `environment` oder `project` enthält oder keines dieser Tags angibt. Darüber hinaus müssen Benutzer Werte für die Tags angeben, die den zulässigen Werten in der Richtlinie entsprechen.

Mit der folgenden Richtlinie können Benutzer DB-Cluster erstellen und dabei beliebige Tags anwenden, mit Ausnahme des Tags `environment=prod`:

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBCluster"
           ],
           "Resource": "*"
       },
       {
```

```
            "Effect": "Allow",
            "Action": [
                "rds:AddTagsToResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestTag/environment": "prod"
                }
            }
        }
    ]
}
```

Unterstützte RDS-API-Aktionen für das Tagging bei der Erstellung

Die folgenden RDS-API-Aktionen unterstützen Tagging beim Erstellen einer Ressource. Für diese Aktionen können Sie beim Erstellen der Ressource Tags angeben:

- `CreateBlueGreenDeployment`
- `CreateCustomDBEngineVersion`
- `CreateDBCluster`
- `CreateDBClusterEndpoint`
- `CreateDBClusterParameterGroup`
- `CreateDBClusterSnapshot`
- `CreateDBInstance`
- `CreateDBInstanceReadReplica`
- `CreateDBParameterGroup`
- `CreateDBProxy`
- `CreateDBProxyEndpoint`
- `CreateDBSecurityGroup`
- `CreateDBShardGroup`
- `CreateDBSnapshot`
- `CreateDBSubnetGroup`
- `CreateEventSubscription`

- CreateGlobalCluster
- CreateIntegration
- CreateOptionGroup
- CreateTenantDatabase
- CopyDBClusterParameterGroup
- CopyDBClusterSnapshot
- CopyDBParameterGroup
- CopyDBSnapshot
- CopyOptionGroup
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime
- PurchaseReservedDBInstancesOffering

Wenn Sie die API AWS CLI oder verwenden, um eine Ressource mit Tags zu erstellen, wird der `Tags` Parameter verwendet, um während der Erstellung Tags auf Ressourcen anzuwenden.

Wenn das Tagging bei diesen API-Aktionen fehlschlägt, wird die Ressource nicht erstellt und die Anforderung schlägt mit einem Fehler fehl. Dadurch wird sichergestellt, dass Ressourcen entweder mit Tags oder gar nicht erstellt werden, was wiederum verhindert, dass Ressourcen ohne die vorgesehenen Tags erstellt werden.

AWS-verwaltete Richtlinien für Amazon RDS

Wenn Sie Berechtigungssätzen und Rollen Berechtigungen hinzufügen möchten, ist es einfacher, von AWS verwaltete Richtlinien zu verwenden, als selbst Richtlinien zu schreiben. Es erfordert Zeit und Fachwissen, um von Kunden verwaltete IAM-Richtlinien zu erstellen, die Ihrem Team nur die benötigten Berechtigungen bieten. Um schnell loszulegen, können Sie unsere von AWS verwalteten Richtlinien verwenden. Diese Richtlinien decken allgemeine Anwendungsfälle ab und sind in Ihrem AWS-Konto verfügbar. Weitere Informationen zu verwalteten AWS-Richtlinien finden Sie unter Verwaltete AWS-Richtlinien im IAM-Leitfaden.

AWS-Services pflegen und aktualisieren AWS-verwaltete Richtlinien. Die Berechtigungen in von AWS verwalteten Richtlinien können nicht geändert werden. Services fügen einer von AWS verwalteten Richtlinien gelegentlich zusätzliche Berechtigungen hinzu, um neue Features zu unterstützen. Diese Art von Update betrifft alle Identitäten (Berechtigungssätze und Rollen), denen die Richtlinie angehängt ist. Services aktualisieren eine von AWS verwaltete Richtlinie am ehesten, ein neues Feature gestartet wird oder neue Vorgänge verfügbar werden. Services entfernen keine Berechtigungen aus einer von AWS verwalteten Richtlinie, so dass Richtlinien-Aktualisierungen Ihre vorhandenen Berechtigungen nicht beeinträchtigen.

Darüber hinaus unterstützt AWS verwaltete Richtlinien für Auftragsfunktionen, die mehrere Services umfassen. So bietet die AWS-verwaltete Richtlinie `ReadOnlyAccess` beispielsweise einen schreibgeschützten Zugriff auf alle AWS-Services und Ressourcen. Wenn ein Service ein neues Feature startet, fügt AWS schreibgeschützte Berechtigungen für neue Vorgänge und Ressourcen hinzu. Eine Liste und Beschreibungen der Richtlinien für Auftragsfunktionen finden Sie in Verwaltete AWS-Richtlinien für Auftragsfunktionen im IAM-Leitfaden.

Themen

- Von AWS verwaltete Richtlinie: AmazonRDSReadOnlyAccess
- Von AWS verwaltete Richtlinie: AmazonRDSFullAccess
- Von AWS verwaltete Richtlinie: AmazonRDSDataFullAccess
- Von AWS verwaltete Richtlinie: AmazonRDSEnhancedMonitoringRole
- Von AWS verwaltete Richtlinie: AmazonRDSPerformanceInsightsReadOnly
- AWS-verwaltete Richtlinie: AmazonRDSPerformanceInsightsFullAccess
- Von AWS verwaltete Richtlinie: AmazonRDSDirectoryServiceAccess
- Von AWS verwaltete Richtlinie: AmazonRDSServiceRolePolicy

- Von AWS verwaltete Richtlinie: AmazonRDSPreviewServiceRolePolicy
- AWS-verwaltete Richtlinie: AmazonRDSBetaServiceRolePolicy

Von AWS verwaltete Richtlinie: AmazonRDSReadOnlyAccess

Diese Richtlinie gewährt schreibgeschützten Zugriff auf Amazon RDS über die AWS-Managementkonsole.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `rds` – Ermöglicht es Prinzipalen, Amazon-RDS-Ressourcen zu beschreiben und die Tags für Amazon-RDS-Ressourcen aufzulisten.
- `cloudwatch` – Ermöglicht es Prinzipalen, Amazon-CloudWatch-Metrikstatistiken abzurufen.
- `ec2` – Ermöglicht es Prinzipalen, Availability Zones und Netzwerkressourcen zu beschreiben.
- `logs` – Ermöglicht es Prinzipalen, Protokoll-Streams von CloudWatch Logs von Protokollgruppen zu beschreiben und CloudWatch-Logs-Protokollereignisse abzurufen.
- `devops-guru` – Ermöglicht Prinzipalen, Ressourcen zu beschreiben, die über Abdeckung durch Amazon DevOps Guru verfügen, was entweder durch CloudFormation-Stacknamen oder Ressourcen-Tags angegeben wird.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSReadOnlyAccess im Referenzleitfaden zur AWS-verwalteten Richtlinie.

Von AWS verwaltete Richtlinie: AmazonRDSFullAccess

Diese Richtlinie bietet Vollzugriff auf Amazon RDS über die AWS-Managementkonsole.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `rds` – Ermöglicht Prinzipalen Vollzugriff auf alle Amazon RDS.
- `application-autoscaling` – Ermöglicht es Prinzipalen, Ziele und Richtlinien zur Skalierung der automatischen Anwendungsskalierung zu beschreiben und zu verwalten.

- `cloudwatch` – Ermöglicht es Prinzipalen, CloudWatch-Metrik-Statiken abzurufen und CloudWatch-Alarme zu verwalten.
- `ec2` – Ermöglicht es Prinzipalen, Availability Zones und Netzwerkressourcen zu beschreiben.
- `logs` – Ermöglicht es Prinzipalen, Protokoll-Streams von CloudWatch Logs von Protokollgruppen zu beschreiben und CloudWatch-Logs-Protokollereignisse abzurufen.
- `outposts` – Ermöglicht es Prinzipalen, AWS Outposts-Instance-Typen abzurufen.
- `pi` – Ermöglicht es Prinzipalen, Performance-Insights-Metriken abzurufen.
- `sns` – Ermöglicht es Prinzipalen, Amazon Simple Notification Service (Amazon SNS)-Abonnements und -Themen zu abonnieren und Amazon-SNS-Nachrichten zu veröffentlichen.
- `devops-guru` – Ermöglicht Prinzipalen, Ressourcen zu beschreiben, die über Abdeckung durch Amazon DevOps Guru verfügen, was entweder durch CloudFormation-Stacknamen oder Ressourcen-Tags angegeben wird.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSFullAccess im Referenzleitfaden zur AWS-verwalteten Richtlinie.

Von AWS verwaltete Richtlinie: AmazonRDSDataFullAccess

Diese Richtlinie erlaubt Vollzugriff, um die Daten-API und den Abfrage-Editor auf Aurora Serverless-Clustern in einem bestimmten AWS-Konto zu verwenden. Diese Richtlinie erlaubt dem AWS-Konto, den Wert eines Secrets von AWS Secrets Manager abzurufen.

Sie können die `AmazonRDSDataFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `dbqms` – Ermöglicht es Prinzipalen, auf Abfragen zuzugreifen, Abfragen zu erstellen, zu löschen, zu beschreiben und zu aktualisieren. Der Database Query Metadata Service (`dbqms`) ist ein reiner Dienst für interne Daten. Er stellt Ihre aktuellen und gespeicherten Abfragen für den Abfrage-Editor auf AWS-Managementkonsole für mehrere AWS-Services bereit, einschließlich Amazon RDS.
- `rds-data` – Ermöglicht es Prinzipalen, SQL-Anweisungen in Aurora Serverless-Datenbanken auszuführen.
- `secretsmanager` – Ermöglicht es Prinzipalen, den Wert eines Secrets von AWS Secrets Manager abzurufen.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSDataFullAccess im Referenzleitfaden zur AWS-verwalteten Richtlinie.

Von AWS verwaltete Richtlinie: AmazonRDSEnhancedMonitoringRole

Diese Richtlinie ermöglicht den Zugriff auf Amazon CloudWatch Logs für Amazon RDS Enhanced Monitoring.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `logs` – Ermöglicht es Prinzipalen, CloudWatch-Logs-Protokollgruppen und -Aufbewahrungsrichtlinien zu erstellen und CloudWatch-Logs-Protokollstreams von Protokollgruppen zu erstellen und zu beschreiben. Außerdem können Prinzipale, CloudWatch-Log-Protokollereignisse ablegen und abrufen.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSEnhancedMonitoringRole im Referenzleitfaden zur AWS-verwalteten Richtlinie.

Von AWS verwaltete Richtlinie: AmazonRDSPerformanceInsightsReadOnly

Diese Richtlinie bietet schreibgeschützten Zugriff auf Erkenntnisse zur Amazon-RDS-Leistung für Amazon-RDS-DB-Instances und Amazon-Aurora-DB-Cluster.

Die Richtlinie enthält jetzt `Sid` (Anweisungs-ID) als Bezeichner für die Richtlinienanweisung.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `rds` – Ermöglicht es Prinzipalen, Amazon-RDS-DB-Instances und Amazon-Aurora-DB-Cluster zu beschreiben.
- `pi` – Ermöglicht es Prinzipalen, Aufrufe an die Amazon-RDS-Performance-Insights-API zu tätigen und auf Performance-Insights-Metriken zuzugreifen

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSPerformanceInsightsReadOnly im Referenzleitfaden zur AWS-verwalteten Richtlinie.

AWS-verwaltete Richtlinie: AmazonRDSPerformanceInsightsFullAccess

Diese Richtlinie bietet Vollzugriff auf Erkenntnisse zur Amazon-RDS-Leistung für DB-Instances von Amazon RDS und DB-Clustern von Amazon Aurora.

Die Richtlinie enthält jetzt `Sid` (Anweisungs-ID) als Bezeichner für die Richtlinienanweisung.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `rds` – Ermöglicht es Prinzipalen, Amazon-RDS-DB-Instances und Amazon-Aurora-DB-Cluster zu beschreiben.
- `pi` – Ermöglicht es Prinzipalen, die API von Erkenntnissen zur Amazon-RDS-Leistung aufzurufen und Leistungsanalyseberichte zu erstellen, anzusehen und zu löschen.
- `cloudwatch` – Ermöglicht es Prinzipalen, alle Amazon-CloudWatch-Metriken aufzulisten sowie Metrikdaten und Statistiken abzurufen.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSPerformanceInsightsFullAccess im Referenzleitfaden zur AWS-verwalteten Richtlinie.

Von AWS verwaltete Richtlinie: AmazonRDSDirectoryServiceAccess

Diese Richtlinie ermöglicht es Amazon RDS, Aufrufe an Directory Service zu tätigen.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgende Berechtigung:

- `ds` – Ermöglicht es Prinzipalen, Directory Service-Verzeichnisse zu beschreiben und die Autorisierung für Directory Service-Verzeichnisse zu kontrollieren.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSDirectoryServiceAccess im Referenzleitfaden zur AWS-verwalteten Richtlinie.

Von AWS verwaltete Richtlinie: AmazonRDSServiceRolePolicy

Sie können die `AmazonRDSServiceRolePolicy`-Richtlinie Ihren IAM-Entitäten nicht anfügen. Diese Richtlinie ist mit einer servicegebundenen Rolle verknüpft, die es Amazon RDS ermöglicht,

Aktionen in Ihrem Namen durchzuführen. Weitere Informationen finden Sie unter Berechtigungen von serviceverknüpften Rollen für Amazon Aurora.

Von AWS verwaltete Richtlinie: AmazonRDSPreviewServiceRolePolicy

Sie sollten `AmazonRDSPreviewServiceRolePolicy` nicht an Ihre IAM-Entitäten anhängen. Diese Richtlinie ist an eine servicegebundenen Rolle angefügt, die es Amazon RDS ermöglicht, AWS-Services im Namen Ihrer RDS-DB-Ressourcen durchzuführen. Weitere Informationen finden Sie unter Serviceverknüpfte Rolle für Amazon RDS Preview.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `ec2` – Ermöglicht es Prinzipalen, Availability Zones und Netzwerkressourcen zu beschreiben.
- `secretsmanager` – Ermöglicht es Prinzipalen, den Wert eines Secrets von AWS Secrets Manager abzurufen.
- `cloudwatch`, `logs` – Ermöglicht Amazon RDS das Hochladen von DB-Instance-Metriken und -Protokollen über den CloudWatch-Agenten auf CloudWatch.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSPreviewServiceRolePolicy im Referenzleitfaden zur AWS-verwalteten Richtlinie.

AWS-verwaltete Richtlinie: AmazonRDSBetaServiceRolePolicy

Sie sollten `AmazonRDSBetaServiceRolePolicy` nicht an Ihre IAM-Entitäten anhängen. Diese Richtlinie ist an eine servicegebundenen Rolle angefügt, die es Amazon RDS ermöglicht, AWS-Services im Namen Ihrer RDS-DB-Ressourcen durchzuführen. Weitere Informationen finden Sie unter Serviceverknüpfte Rollenberechtigungen für Amazon RDS Beta.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `ec2` – Ermöglicht Amazon RDS die Durchführung von Backup-Vorgängen auf der DB-Instance, die Point-in-Time-Wiederherstellungsfunktionen bietet.
- `secretsmanager` – Ermöglicht Amazon RDS die Verwaltung von DB-Instance-spezifischen Secrets, die von Amazon RDS erstellt wurden.

- `cloudwatch`, `logs` – Ermöglicht Amazon RDS das Hochladen von DB-Instance-Metriken und -Protokollen über den CloudWatch-Agenten auf CloudWatch.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSBetaServiceRolePolicy im Referenzleitfaden zur AWS-verwalteten Richtlinie.

Änderungen von Amazon RDS an von AWS verwalteten Richtlinien

Anzeigen von Details zu Aktualisierungen für AWS-verwaltete Richtlinien für Amazon RDS, seit dieser Dienst mit der Verfolgung dieser Änderungen begonnen hat. Um automatische Warnungen über Änderungen an dieser Seite zu erhalten, abonnieren Sie den RSS-Feed auf der Amazon-RDS-Dokumentverlauf-Seite.

Änderung	Beschreibung	Datum
Von AWS verwaltete Richtlinie: AmazonRDSPreviewServiceRolePolicy – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat die Berechtigung `sns:Publish` aus der `AmazonRDSPreviewServiceRolePolicy` der serviceverknüpften Rolle `AWSServiceRoleForRDSPreview` entfernt. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinie: AmazonRDSPreviewServiceRolePolicy.	7. August 2024
AWS-verwaltete Richtlinie: AmazonRDSBetaServiceRolePolicy – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat die Berechtigung `sns:Publish` aus der `AmazonRDSBetaServiceRolePolicy` der serviceverknüpften Rolle `AWSServiceRoleForRDSBeta` entfernt. Weitere Informationen finden Sie unter AWS-verwaltete Richtlinie: AmazonRDSBetaServiceRolePolicy.	7. August 2024
Von AWS verwaltete Richtlinie: AmazonRDSServiceRo	Amazon RDS hat die Berechtigung `sns:Publi`	2. Juli 2024

Änderung	Beschreibung	Datum
lePolicy – Aktualisierung auf eine bestehende Richtlinie	sh aus der AmazonRDS ServiceRolePolicy der serviceverknüpften Rolle AWSServiceRoleForRDS entfernt. Weitere Informati onen finden Sie unter Von AWS verwaltete Richtlini e: AmazonRDSServiceRo lePolicy.	
AWS-verwaltete Richtlinien für Amazon RDS – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat der AmazonRDSCustomSer viceRolePolicy der serviceverknüpften Rolle AWSServiceRoleForR DSCustom eine neue Berechtigung hinzugefü gt, damit RDS Custom für SQL Server den zugrunde liegenden Datenbank-Host-Ins tance-Typ ändern kann. RDS hat außerdem die Berechtig ung ec2:DescribeInstan ceTypes hinzugefügt, Informationen zum Instance-Typ für den Datenbank-Host abzurufen. Weitere Informati onen finden Sie unter AWS-verwaltete Richtlinien für Amazon RDS.	8. April 2024

Änderung	Beschreibung	Datum
AWS-verwaltete Richtlinien für Amazon RDS – Neue Richtlinie	Amazon RDS hat die neue verwaltete Richtlinie `AmazonRDSCustomInstanceProfileRolePolicy` hinzugefügt, mit der RDS Custom Automatisierungsaktionen und Datenbankverwaltungsaufgaben über ein EC2-Instance-Profil ausführen kann. Weitere Informationen finden Sie unter AWS-verwaltete Richtlinien für Amazon RDS.	27. Februar 2024
Berechtigungen von serviceverknüpften Rollen für Amazon Aurora – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat neue Anweisungs-IDs zur `AmazonRDSServiceRolePolicy` der serviceverknüpftem Rolle `AWSServiceRoleForRDS` hinzugefügt. Weitere Informationen finden Sie unter Berechtigungen von serviceverknüpften Rollen für Amazon Aurora.	19. Januar 2024

Änderung	Beschreibung	Datum
AWS-verwaltete Richtlinien für Amazon RDS – Aktualisierung auf bestehende Richtlinien	Die von `AmazonRDSPerformanceInsightsReadOnly` und `AmazonRDSPerformanceInsightsFullAccess` verwalteten Richtlinien enthalten jetzt `Sid` (Statement-ID) als Bezeichner in der Richtlinienerklärung. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinie: AmazonRDSPerformanceInsightsReadOnly und AWS-verwaltete Richtlinie: AmazonRDSPerformanceInsightsFullAccess	23. Oktober 2023

Änderung	Beschreibung	Datum
AWS-verwaltete Richtlinien für Amazon RDS – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat der verwalteten `AmazonRDSFullAccess` -Richtlinie neue Berechtigungen hinzugefügt. Mit den Berechtigungen können Sie den Leistungsanalysebericht für einen bestimmten Zeitraum erstellen, anzeigen und löschen. Weitere Informationen zur Konfiguration von Zugriffsrichtlinien für Performance Insights finden Sie unter Konfigurieren von Zugriffsrichtlinien für Performance Insights.	17. August 2023

Änderung	Beschreibung	Datum
AWS-verwaltete Richtlinien für Amazon RDS – Neue Richtlinie und Aktualisierung der bestehenden Richtlinie	Amazon RDS hat der verwalteten `AmazonRDSPerformanceInsightsReadOnly` -Richtlinie neue Berechtigungen und eine neue verwaltete Richtlinie mit dem Namen `AmazonRDSPerformanceInsightsFullAccess` hinzugefügt. Diese Berechtigungen ermöglichen es Ihnen, Performance Insights für einen bestimmten Zeitraum zu analysieren, sich die Analyseergebnisse zusammen mit den Empfehlungen anzusehen und die Berichte zu löschen. Weitere Informationen zur Konfiguration von Zugriffsrichtlinien für Performance Insights finden Sie unter Konfigurieren von Zugriffsrichtlinien für Performance Insights.	16. August 2023

Änderung	Beschreibung	Datum
AWS-verwaltete Richtlinien für Amazon RDS – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat `AmazonRDSFullAccess` und `AmazonRDSReadOnlyAccess` einen neuen Amazon-CloudWatch-Namespace `ListMetrics` hinzugefügt. Dieser Namespace ist erforderlich, damit Amazon RDS spezifische Metriken zur Ressourcennutzung auflisten kann. Weitere Informationen finden Sie unter Übersicht über die Verwaltung von Zugriffsberechtigungen für Ihre CloudWatch-Ressourcen im Amazon-CloudWatch-Benutzerhandbuch.	4. April 2023

Änderung	Beschreibung	Datum
Berechtigungen von serviceverknüpften Rollen für Amazon Aurora – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat neue Berechtigungen zur `AmazonRDSServiceRolePolicy` der servicevernüpftem Rolle `AWSServiceRoleForRDS` zur Integration in AWS Secrets Manager hinzugefügt. RDS erfordert die Integration mit Secrets Manager für die Verwaltung von Hauptbenutzerpasswörtern in Secrets Manager. Das Secret verwendet eine reservierte Namenskonvention und schränkt Kundenaktualisierungen ein. Weitere Informationen finden Sie unter Passwortverwaltung mit , Amazon Aurora und AWS Secrets Manager.	22. Dezember 2022

Änderung	Beschreibung	Datum
AWS-verwaltete Richtlinien für Amazon RDS – Aktualisierung auf bestehende Richtlinien	Amazon RDS hat den verwalteten Richtlinien `AmazonRDSFullAccess` und `AmazonRDSReadOnlyAccess` eine neue Berechtigung hinzugefügt, mit der Sie Amazon DevOps Guru in der RDS-Konsole aktivieren können. Diese Berechtigung ist erforderlich, um zu überprüfen, ob DevOps Guru aktiviert ist. Weitere Informationen finden Sie unter Konfiguration von IAM-Zugriffsrichtlinien für Guru for RDS DevOps.	19. Dezember 2022

Änderung	Beschreibung	Datum
Berechtigungen von serviceverknüpften Rollen für Amazon Aurora – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat einen neuen Amazon-CloudWatch-Namespace für `PutMetricData` zu `AmazonRDSPreviewServiceRolePolicy` hinzugefügt. Dieser Namespace ist erforderlich, damit Amazon RDS Metriken zur Ressourcennutzung veröffentlichen kann. Weitere Informationen finden Sie unter Verwenden von Bedingungsschlüsseln zum Einschränken des Zugriffs auf CloudWatch-Namespaces im Amazon CloudWatch-Benutzerhandbuch.	7. Juni 2022

Änderung	Beschreibung	Datum
Berechtigungen von serviceverknüpften Rollen für Amazon Aurora – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat einen neuen Amazon-CloudWatch-Namespace für `PutMetricData` zu `AmazonRDSBetaServiceRolePolicy` hinzugefügt. Dieser Namespace ist erforderlich, damit Amazon RDS Metriken zur Ressourcennutzung veröffentlichen kann. Weitere Informationen finden Sie unter Verwenden von Bedingungsschlüsseln zum Einschränken des Zugriffs auf CloudWatch-Namespaces im Amazon CloudWatch-Benutzerhandbuch.	7. Juni 2022

Änderung	Beschreibung	Datum
Berechtigungen von serviceverknüpften Rollen für Amazon Aurora – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat einen neuen Amazon-CloudWatch-Namespace für `PutMetricData` zu `AWSServiceRoleForRDS` hinzugefügt. Dieser Namespace ist erforderlich, damit Amazon RDS Metriken zur Ressourcennutzung veröffentlichen kann. Weitere Informationen finden Sie unter Verwenden von Bedingungsschlüsseln zum Einschränken des Zugriffs auf CloudWatch-Namespaces im Amazon CloudWatch-Benutzerhandbuch.	22. April 2022
AWS-verwaltete Richtlinien für Amazon RDS – Neue Richtlinie	Amazon RDS hat eine neue verwaltete Richtlinie namens `AmazonRDSPerformanceInsightsReadOnly` hinzugefügt, um Amazon RDS zu erlauben, AWS-Services im Namen Ihrer DB-Instances aufzurufen. Weitere Informationen zur Konfiguration von Zugriffsrichtlinien für Performance Insights finden Sie unter Konfigurieren von Zugriffsrichtlinien für Performance Insights.	10. März 2022

Änderung	Beschreibung	Datum
Berechtigungen von serviceverknüpften Rollen für Amazon Aurora – Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat neue Amazon-CloudWatch-Namespaces für `PutMetricData` zu `AWSServiceRoleForRDS` hinzugefügt. Diese Namespaces sind erforderlich, damit Amazon DocumentDB (mit MongoDB-Kompatibilität) und Amazon Neptune CloudWatch-Metriken veröffentlichen können. Weitere Informationen finden Sie unter Verwenden von Bedingungsschlüsseln zum Einschränken des Zugriffs auf CloudWatch-Namespaces im Amazon CloudWatch-Benutzerhandbuch.	4. März 2022
Amazon RDS hat mit der Verfolgung von Änderungen begonnen	Amazon RDS hat mit der Verfolgung von Änderungen für seine AWS-verwaltete Richtlinien begonnen.	26. Oktober 2021

Vermeidung des dienstübergreifenden Confused-Deputy-Problems

Das Confused-Deputy-Problem ist ein Sicherheitsproblem, bei dem eine Entität, die nicht über die Berechtigung zum Ausführen einer Aktion verfügt, eine Entität mit größeren Rechten zwingen kann, die Aktion auszuführen. In AWS kann ein dienstübergreifendes Identitätswechsels zu einem Problem mit dem verwirrten Stellvertreter führen.

Ein serviceübergreifender Identitätswechsel kann auftreten, wenn ein Service (der Anruf-Service) einen anderen Service anruft (den aufgerufenen Service). Der aufrufende Dienst kann so manipuliert werden, dass er seine Berechtigungen verwendet, um auf die Ressourcen eines anderen Kunden zu reagieren, auf die er sonst nicht zugreifen dürfte. Um dies zu verhindern, AWS bietet Tools, mit denen Sie Ihre Daten für alle Dienste mit Dienstprinzipalen schützen können, denen Zugriff auf Ressourcen in Ihrem Konto gewährt wurde. Weitere Informationen finden Sie unter Das Problem des verwirrten Stellvertreters im IAM-Benutzerhandbuch.

Wir empfehlen die Verwendung der globalen Bedingungskontextschlüssel `aws:SourceArn` und `aws:SourceAccount` in ressourcenbasierten Richtlinien, um die Berechtigungen, die Amazon RDS einem anderen Service erteilt, auf eine bestimmte Ressource zu beschränken.

In einigen Fällen enthält der `aws:SourceArn`-Wert nicht die Konto-ID, z. B. wenn Sie den Amazon-Ressourcennamen (ARN) für einen Amazon-S3-Bucket verwenden. Stellen Sie in diesen Fällen sicher, dass Sie beide globalen Kontextschlüssel für die Bedingung verwenden, um Berechtigungen einzuschränken. In einigen Fällen verwenden Sie beide globalen Bedingungskontextschlüssel und der `aws:SourceArn`-Wert enthält die Konto-ID. Stellen Sie in diesen Fällen sicher, dass der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn` dieselbe Konto-ID verwenden, wenn sie in derselben Richtlinienanweisung verwendet werden. Wenn Sie nur eine Ressource mit dem betriebsübergreifenden Zugriff verknüpfen möchten, verwenden Sie `aws:SourceArn`. Wenn Sie zulassen möchten, dass eine Ressource im angegebenen AWS Konto der dienstübergreifenden Nutzung zugeordnet wird, verwenden Sie. `aws:SourceAccount`

Stellen Sie sicher, dass der Wert von `aws:SourceArn` ein ARN für einen Amazon-RDS-Ressourcentyp ist. Weitere Informationen finden Sie unter Amazon-Ressourcennamen (ARN) in Amazon RDS.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontextschlüssels `aws:SourceArn` mit dem vollständigen ARN der Ressource. In einigen Fällen kennen Sie möglicherweise den vollständigen ARN der Ressource nicht oder Sie geben möglicherweise mehrere Ressourcen an. Verwenden Sie in diesen Fällen die

globalen `aws:SourceArn`-Kontextbedingungsschlüssel mit Platzhaltern (*) für die unbekannten Teile des ARN. Ein Beispiel ist `arn:aws:rds:*:123456789012:*`.

Das folgende Beispiel zeigt, wie Sie die globalen Bedingungskontextschlüssel `aws:SourceArn` und `aws:SourceAccount` für Amazon RDS verwenden können, um das Confused-Deputy-Problem zu verhindern.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Weitere Beispiele für Richtlinien, die die globalen Bedingungskontextschlüssel `aws:SourceArn` und `aws:SourceAccount` verwenden, finden Sie in den folgenden Abschnitten:

- Erteilen von Berechtigungen zum Veröffentlichen von Benachrichtigungen in einem Amazon-SNS-Thema
- Einrichten des Zugriffs auf einen Amazon-S3-Bucket (PostgreSQL-Import)
- Einrichten des Zugriffs auf einen Amazon-S3-Bucket (PostgreSQL-Export)

IAM-Datenbankauthentifizierung

Sie können sich bei Ihrem mithilfe der AWS Identity and Access Management (IAM-) Datenbankauthentifizierung authentifizieren. Die IAM-Datenbankauthentifizierung funktioniert mit Aurora MySQL und Aurora PostgreSQL. Mit dieser Authentifizierungsmethode benötigen Sie kein Passwort, um eine Verbindung mit einer DB-Cluster herzustellen. Stattdessen verwenden Sie ein Authentifizierungstoken.

Ein Authentifizierungstoken ist eine eindeutige Zeichenfolge, die von Amazon Aurora auf Anforderung erzeugt wird. Authentifizierungstoken werden mit AWS Signature Version 4 generiert. Jedes Token verfällt 15 Minuten nach seiner Erzeugung. Da die Authentifizierung mithilfe von IAM extern verwaltet wird, ist es nicht erforderlich, Benutzeranmeldeinformationen in der Datenbank zu speichern. Sie können weiterhin auch die Standard-Datenbank-Authentifizierung verwenden. Das Token wird nur zur Authentifizierung verwendet und wirkt sich nicht auf die Sitzung aus, nachdem es eingerichtet wurde.

Die IAM-Datenbank-Authentifizierung bietet die folgenden Vorteile:

- Der Netzwerkverkehr zur und von der Datenbank wird mit Secure Socket Layer (SSL) oder Transport Layer Security (TLS) verschlüsselt. Weitere Informationen zur Verwendung von Amazon Aurora SSL/TLS mit Amazon finden Sie unterWird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.
- Sie können IAM verwenden, um den Zugriff auf Ihre Datenbankressourcen zentral zu verwalten, anstatt den Zugriff individuell auf jedem DB-Cluster zu verwalten.
- Für Anwendungen, die auf Amazon ausgeführt werden EC2, können Sie für Ihre EC2 Instance spezifische Profilanmeldedaten anstelle eines Kennworts verwenden, um die Sicherheit zu erhöhen.

Erwägen Sie im Allgemeinen, die IAM-Datenbankauthentifizierung zu verwenden, wenn Ihre Anwendungen weniger als 200 Verbindungen pro Sekunde erstellen und Sie Benutzernamen und Passwörter nicht direkt in Ihrem Anwendungscode verwalten möchten.

Der JDBC-Treiber von Amazon Web Services (AWS) unterstützt die IAM-Datenbankauthentifizierung. Weitere Informationen finden Sie unter AWS IAM-Authentifizierungs-Plug-In im Amazon Web Services (AWS) JDBC-Treiber-Repository. GitHub

Der Python-Treiber von Amazon Web Services (AWS) unterstützt die IAM-Datenbankauthentifizierung. Weitere Informationen finden Sie unter AWS IAM-Authentifizierungs-Plug-In im GitHubPython-Treiber-Repository von Amazon Web Services (AWS).

Navigieren Sie durch die folgenden Themen, um zu erfahren, wie IAM für die DB-Authentifizierung eingerichtet wird:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung
- Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung

Verfügbarkeit von Regionen und Versionen

Die Verfügbarkeit von Funktionen und der Support variieren zwischen bestimmten Versionen der einzelnen Aurora-Datenbank-Engines und in allen AWS-Regionen. Weitere Informationen zur Verfügbarkeit von Versionen und Regionen mit Aurora- und IAM-Datenbankauthentifizierung finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für die IAM-Datenbankauthentifizierung.

Für Aurora MySQL unterstützen alle unterstützten DB-Instance-Klassen die IAM-Datenbank-Authentifizierung, mit Ausnahme von db.t2.small und db.t3.small. Informationen zu den unterstützten DB-Instance-Klassen finden Sie unter Unterstützte DB-Engines für DB-Instance-Klassen.

CLI- und SDK-Unterstützung

Die IAM-Datenbankauthentifizierung ist für die AWS CLIund für die folgenden sprachspezifischen Sprachen verfügbar: AWS SDKs

- AWS SDK für .NET
- AWS SDK für C++
- AWS SDK für Go
- AWS SDK für Java
- AWS SDK für JavaScript
- AWS SDK für PHP
- AWS SDK for Python (Boto3)
- AWS SDK für Ruby

Einschränkungen der IAM-Datenbank-Authentifizierung

Beachten Sie bei der Verwendung der IAM-Datenbankauthentifizierung die folgenden Einschränkungen:

- Derzeit unterstützt die IAM-Datenbankauthentifizierung nicht alle globalen Bedingungskontextschlüssel.

  Weitere Informationen über globale Bedingungskontextschlüssel finden Sie unter Globale AWS - Bedingungskontextschlüssel im IAM-Benutzerhandbuch.
- Wenn die IAM-Rolle (`rds_iam`) einem Benutzer (einschließlich RDS-Hauptbenutzer) hinzugefügt wird, hat bei PostgreSQL die IAM-Authentifizierung Vorrang vor der Passwort-Authentifizierung, so dass sich der Benutzer als IAM-Benutzer anmelden muss.
- Sie können keinen benutzerdefinierten Route-53-DNS-Datensatz anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.
- CloudWatch und protokollieren Sie die CloudTrail IAM-Authentifizierung nicht. Diese Services verfolgen keine API-Aufrufe des Typs `generate-db-auth-token`, die die IAM-Rolle autorisieren, eine Datenbankverbindung zu aktivieren.
- Die IAM-DB-Authentifizierung erfordert Rechenressourcen auf dem Cluster der Datenbank. Für eine zuverlässige Verbindung benötigen Sie zwischen 300 und 1 000 MiB zusätzlichen Speicher in Ihrer Datenbank. Um den für Ihren Workload benötigten Speicher zu ermitteln, vergleichen Sie die RES-Spalte für RDS-Prozesse in der processlist für „Enhanced Monitoring“ vor und nach der Aktivierung der IAM-DB-Authentifizierung. Siehe Anzeigen von Betriebssystem-Metriken in der RDS-Konsole.

  Wenn Sie eine Burstable-Instance-Klasse verwenden, vermeiden Sie einen Speichermangel, indem Sie den von anderen Parametern wie Puffern und Cache belegten Speicher um denselben Betrag reduzieren.
- Für Aurora MySQL können Sie keine passwortbasierte Authentifizierung für einen Datenbankbenutzer verwenden, den Sie mit IAM-Authentifizierung konfigurieren.
- Die IAM-DB-Authentifizierung wird für RDS auf Outposts für keine Engine unterstützt.

Empfehlungen für die IAM-Datenbankauthentifizierung

Bei Verwendung der IAM-Datenbankauthentifizierung empfehlen wir Folgendes:

- Verwenden Sie die IAM-Datenbankauthentifizierung, wenn Ihre Anwendung weniger als 200 neue IAM-Datenbankauthentifizierungsverbindungen pro Sekunde benötigt.

  Die Datenbank-Engines, die mit Amazon Aurora kompatibel sind, setzen den Authentifizierungsversuchen pro Sekunde keine Grenzen. Wenn Sie jedoch die IAM-Datenbank-Authentifizierung verwenden, muss Ihre Anwendung ein Authentifizierungstoken erzeugen. Ihre Anwendung verwendet dann dieses Token, um eine Verbindung mit dem DB-Cluster herzustellen. Wenn Sie die Höchstanzahl neuer Verbindungen pro Sekunde überschreiten, kann der zusätzliche Overhead der IAM-Datenbankauthentifizierung zur Verbindungsablehnung führen.

  Erwägen Sie die Verwendung von Verbindungspooling in Ihren Anwendungen, um den ständigen Verbindungsaufbau zu begrenzen. Dadurch lässt sich der Aufwand für die IAM-DB-Authentifizierung reduzieren und Ihre Anwendungen können bestehende Verbindungen wiederverwenden. Erwägen Sie alternativ die Verwendung von RDS-Proxy für diese Anwendungsfälle. RDS-Proxy ist mit zusätzlichen Kosten verbunden. Weitere Informationen finden Sie unter RDS-Proxy – Preise.
- Die Größe eines IAM-Datenbankauthentifizierungstokens hängt von vielen Faktoren ab, einschließlich der Anzahl der IAM-Tags, IAM-Servicerichtlinien, ARN-Längen sowie andere IAM- und Datenbankeigenschaften. Die Mindestgröße dieses Tokens beträgt im Allgemeinen etwa 1 KB, kann aber durchaus größer sein. Da dieses Token als Passwort in der Verbindungszeichenfolge zur Datenbank verwendet wird, die die IAM-Authentifizierung verwendet, sollten Sie sicherstellen, dass Ihr Datenbanktreiber (z. B. ODBC) und and/or alle Tools dieses Token nicht aufgrund seiner Größe einschränken oder anderweitig kürzen. Ein verkürztes Token führt dazu, dass die von der Datenbank und IAM durchgeführte Authentifizierungsüberprüfung fehlschlägt.
- Wenn Sie beim Erstellen eines IAM-Datenbank-Authentifizierungstokens temporäre Anmeldeinformationen verwenden, müssen die temporären Anmeldeinformationen weiterhin gültig sein, wenn Sie das IAM-Datenbank-Authentifizierungstoken für eine Verbindungsanforderung verwenden.

Kontextschlüssel für globale Bedingungen werden nicht unterstützt AWS

Die IAM-Datenbankauthentifizierung unterstützt die folgende Teilmenge der AWS globalen Bedingungskontextschlüssel nicht.

- `aws:Referer`
- `aws:SourceIp`
- `aws:SourceVpc`

- `aws:SourceVpce`
- `aws:UserAgent`
- `aws:VpcSourceIp`

Weitere Informationen finden Sie unter Globale AWS -Bedingungskontextschlüssel im IAM-Benutzerhandbuch.

Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung

Die IAM-Datenbank-Authentifizierung ist für DB-Cluster standardmäßig deaktiviert. Sie können die IAM-Datenbankauthentifizierung mithilfe der AWS-Managementkonsole AWS CLI, oder der API aktivieren oder deaktivieren.

Sie können die IAM-Datenbankauthentifizierung aktivieren, wenn Sie eine der folgenden Aktionen ausführen:

- Informationen zum Erstellen eines neuen DB-Clusters mit aktivierter IAM-Datenbankauthentifizierung finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.
- Informationen zum Ändern eines DB-Clusters zur Aktivierung der IAM-Datenbankauthentifizierung finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.
- Informationen zum Wiederherstellen eines DB-Clusters aus einem Snapshot mit aktivierter IAM-Datenbankauthentifizierung finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot.
- Informationen zum Wiederherstellen eines DB-Clusters zu einem Zeitpunkt mit aktivierter IAM-Datenbankauthentifizierung finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit.

Konsole

Jeder Erstellungs- oder Änderungsworkflow verfügt über einen Abschnitt Database authentification (Datenbankauthentifizierung), in dem Sie die IAM-Datenbankauthentifizierung aktivieren oder deaktivieren können. Wählen Sie in diesem Abschnitt Password and IAM database authentication (Passwort- und IAM-Datenbankauthentifizierung), um die IAM-Datenbankauthentifizierung zu aktivieren.

So aktivieren bzw. deaktivieren die IAM-Datenbankauthentifizierung für eine vorhandene DB-einen vorhandenen DB-Cluster:

1. Öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Datenbanken aus.
3. Wählen Sie den DB-Cluster aus, die Sie ändern möchten.

   > **Note**
   >
   > Sie können die IAM-Authentifizierung nur aktivieren, wenn alle DB-Instances im Cluster mit IAM kompatibel sind. Überprüfen Sie die Kompatibilitätsanforderungen in Verfügbarkeit von Regionen und Versionen.

4. Wählen Sie Modify aus.
5. Wählen Sie im Abschnitt Datenbankauthentifizierung die Option aus, um die IAM-Datenbankauthentifizierung zu aktivieren. Wählen Sie Passwort-Authentifizierung oder Passwort- und Kerberos-Authentifizierung aus, um die IAM-Authentifizierung zu deaktivieren.
6. Sie können auch die Veröffentlichung von IAM-DB-Authentifizierungsprotokollen in Logs aktivieren. CloudWatch Wählen Sie unter Protokollexporte die Option iam-db-auth-error Protokoll aus. Das Veröffentlichen Ihrer CloudWatch Protokolle in Logs verbraucht Speicherplatz, und für diesen Speicherplatz fallen Gebühren an. Achten Sie darauf, alle CloudWatch Logs zu löschen, die Sie nicht mehr benötigen.
7. Klicken Sie auf Continue.
8. Um die Änderungen sofort anzuwenden, wählen Sie im Abschnitt Scheduling of modifications (Planen von Änderungen) die Option Immediately (Sofort).
9. Wählen Sie Cluster ändern.

AWS CLI

Verwenden Sie den Befehl, um mithilfe von einen neuen DB-Cluster mit IAM-Authentifizierung zu erstellen. AWS CLI`create-db-cluster` Legen Sie die Option `--enable-iam-database-authentication` fest.

Um einen bestehenden DB-Cluster mit oder ohne IAM-Authentifizierung zu aktualisieren, verwenden Sie den AWS CLI -Befehl `modify-db-cluster`. Sie müssen entweder die Option `--enable-iam-database-authentication` oder `--no-enable-iam-database-authentication` angeben.

> **Note**
>
> Sie können die IAM-Authentifizierung nur aktivieren, wenn alle DB-Instances im Cluster mit IAM kompatibel sind. Überprüfen Sie die Kompatibilitätsanforderungen in Verfügbarkeit von Regionen und Versionen.

Standardmäßig führt Aurora die Änderung während des nächsten Wartungsfensters durch. Wenn Sie diese Einstellung übergehen und die IAM-DB-Authentifizierung schnellstmöglich aktivieren möchten, verwenden Sie den Parameter `--apply-immediately`.

Wenn Sie einen wiederherstellen, verwenden Sie einen der folgenden AWS CLI Befehle:

- `restore-db-cluster-to-point-in-time`
- `restore-db-cluster-from-db-snapshot`

Die Voreinstellung der IAM-Datenbank-Authentifizierung entspricht der Einstellung des Quell-Snapshot. Wählen Sie die entsprechende Option `--enable-iam-database-authentication` oder `--no-enable-iam-database-authentication` aus.

RDS-API

Verwenden Sie die API-Operation , um mithilfe der API eine neue DB-Instance mit IAM-Authentifizierung zu erstellen `CreateDBCluster`. Stellen Sie den Parameter `EnableIAMDatabaseAuthentication` auf `true` ein.

Um einen bestehenden DB-Cluster mit oder ohne IAM-Authentifizierung zu aktualisieren, verwenden Sie die API-Operation `ModifyDBCluster`. Setzen Sie den Parameter `EnableIAMDatabaseAuthentication` auf `true`, um die IAM-Authentifizierung zu aktivieren, oder auf `false`, um sie zu deaktivieren.

> **Note**
>
> Sie können die IAM-Authentifizierung nur aktivieren, wenn alle DB-Instances im Cluster mit IAM kompatibel sind. Überprüfen Sie die Kompatibilitätsanforderungen in Verfügbarkeit von Regionen und Versionen.

Verwenden Sie eine der folgenden API-Operationen, wenn Sie einen DB--Cluster wiederherstellen.

- `RestoreDBClusterFromSnapshot`
- `RestoreDBClusterToPointInTime`

Die Voreinstellung der IAM-Datenbank-Authentifizierung entspricht der Einstellung des Quell-Snapshot. Zum Ändern dieser Einstellung setzen Sie den Parameter `EnableIAMDatabaseAuthentication` auf `true`, um die IAM-Authentifizierung zu aktivieren, oder auf `false`, um sie zu deaktivieren.

Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff

Sie müssen eine IAM-Richtlinie erstellen, um einem Benutzer oder einer Rolle zu erlauben, eine Verbindung mit Ihrem DB-Cluster herzustellen. Anschließend fügen Sie die Richtlinie an einen Berechtigungssatz oder eine Rolle an.

> **Note**
>
> Weitere Informationen zu IAM-Richtlinien finden Sie unter Identity and Access Management für Amazon Aurora.

Die folgende Beispielrichtlinie erlaubt einem Benutzer, eine Verbindung mit einem DB-Cluster mithilfe der IAM-Datenbank-Authentifizierung herzustellen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:db-
ABCDEFGHIJKL01234/db_user"
            ]
        }
    ]
```

```
}
```

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:cluster-
ABCDEFGHIJKL01234/db_user"
            ]
        }
    ]
}
```

Important

Ein Benutzer mit Administratorberechtigungen kann auf DB-Cluster zugreifen, ohne über eine IAM-Richtlinie explizite Berechtigungen zu erhalten. Wenn Sie den Administratorzugriff auf DB--Cluster einschränken möchten, können Sie eine IAM-Rolle mit geeigneten, weniger privilegierten Berechtigungen erstellen und diese dem Administrator zuweisen.

Note

Verwechseln Sie das Präfix `rds-db:` nicht mit anderen RDS-API-Operationspräfixen, die mit `rds:` beginnen. Das Präfix `rds-db:` und die Aktion `rds-db:connect` werden nur zur IAM-Datenbank-Authentifizierung verwendet. Sie sind in keinem anderen Kontext gültig.

Die Beispielrichtlinie enthält eine einzige Anweisung mit den folgenden Elementen:

- `Effect` – Geben Sie `Allow` an, um den Zugriff auf den DB-Cluster zu gewähren. Wenn Sie den Zugriff nicht ausdrücklich erlauben, wird er automatisch verweigert.
- `Action` – Geben Sie `rds-db:connect` an, um Verbindungen mit dem DB-Cluster zu erlauben.
- `Resource` – Geben Sie einen Amazon-Ressourcennamen (ARN) an, der ein Datenbankkonto auf einem DB-Cluster beschreibt. Das ARN-Format lautet folgendermaßen.

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

Ersetzen Sie in diesem Format Folgendes:

- *region*ist die AWS Region für den . In der Beispielrichtlinie ist die AWS Region`us-east-2`.
- *account-id*ist die AWS Kontonummer für den . In der Beispielrichtlinie lautet die Kontonummer `1234567890`. Der Benutzer muss demselben Konto wie das Konto für den DB--Cluster angehören.

  Erstellen Sie für einen kontoübergreifenden Zugriff eine IAM-Rolle mit der oben angegebenen Richtlinie in dem Konto für den DB--Cluster und erlauben Sie Ihrem anderen Konto, die Rolle zu übernehmen.
- *DbClusterResourceId* ist die Kennung des DB-Clusters. Diese Kennung ist für eine AWS Region eindeutig und ändert sich nie. In dieser Beispielrichtlinie lautet die Kennung `cluster-ABCDEFGHIJKL01234`.

  Um eine in Amazon Aurora AWS-Managementkonsole für Amazon zu finden, wählen Sie den aus, um dessen Details zu sehen. Wechseln Sie zur Registerkarte Konfiguration. Die Resource ID (Ressourcen-ID) wird im Abschnitt Configuration Details (Konfigurationsdetails) angezeigt.

  Alternativ können Sie den AWS CLI Befehl verwenden, um die Identifikatoren und Ressourcen IDs für Ihren gesamten in der aktuellen AWS Region aufzulisten, wie im Folgenden dargestellt.

```
aws rds describe-db-clusters --query "DBClusters[*].
[DBClusterIdentifier,DbClusterResourceId]"
```

> **Note**
>
> Wenn Sie über den RDS-Proxy eine Verbindung zu einer Datenbank herstellen, geben Sie die Proxy-Ressourcen-ID an, z. B. `prx-ABCDEFGHIJKL01234`. Hinweise zur Verwendung der IAM-Datenbankauthentifizierung mit RDS-Proxy finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

- *`db-user-name`* ist der Name des mit der IAM-Authentifizierung zu verknüpfenden Datenbankkontos. In der Beispielrichtlinie lautet das Datenbankkonto `db_user`.

Sie können andere erstellen ARNs , um verschiedene Zugriffsmuster zu unterstützen. Die folgende Richtlinie erlaubt den Zugriff auf zwei verschiedene Datenbankkonten auf einem DB-Cluster.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
mary_roe"
         ]
      }
   ]
}
```

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/mary_roe"
         ]
      }
   ]
}
```

Die folgende Richtlinie verwendet das Zeichen „*“, um allen und Datenbankkonten für ein bestimmtes AWS Konto und eine bestimmte AWS Region zuzuordnen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:*/*"
            ]
        }
    ]
}
```

Die folgende Richtlinie entspricht allen für ein bestimmtes AWS Konto und eine bestimmte AWS Region. Allerdings gewährt die Richtlinie nur den Zugriff auf DB-Cluster, die über ein `jane_doe`-Datenbankkonto verfügen.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

Der Benutzer oder die Rolle hat nur auf jene Datenbanken Zugriff, auf die der Datenbankbenutzer zugreifen kann. Nehmen wir beispielsweise an, dass Ihr DB-Cluster über eine Datenbank mit dem Namen dev und eine weitere Datenbank mit dem Namen test verfügt. Wenn der Datenbankbenutzer `jane_doe` nur auf dev zugreifen kann, haben auch alle anderen Benutzer oder Rollen, die auf diesen DB-Cluster mit dem Benutzer `jane_doe` zugreifen, lediglich die Berechtigung für den Zugriff auf dev. Diese Zugriffsbeschränkung gilt auch für andere Datenbankobjekte, wie z. B. Tabellen, Ansichten usw.

Ein Administrator muss IAM-Richtlinien erstellen, die Entitäten die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den Berechtigungssätzen oder Rollen anfügen, die diese Berechtigungen benötigen. Beispiele für Richtlinien finden Sie unter Beispiele für identitätsbasierte Amazon-Aurora-Richtlinien.

Anfügen einer IAM-Richtlinie an einen Berechtigungssatz oder eine Rolle

Nachdem Sie eine IAM-Richtlinie erstellt haben, um die Datenbank-Authentifizierung zu erlauben, müssen Sie die Richtlinie an einen Berechtigungssatz oder eine Rolle anfügen. Eine praktische Anleitung zu diesem Thema finden Sie unter Erstellen und Anfügen Ihrer ersten vom Kunden verwalteten Richtlinie im IAM-Benutzerhandbuch.

Wenn Sie die praktischen Anleitung durchgehen, können Sie eine der in diesem Abschnitt aufgeführten Beispielrichtlinien als Ausgangsbasis verwenden und auf Ihre Bedürfnisse anpassen. Am Ende des Tutorials verfügen Sie über einen Berechtigungssatz mit einer angefügten Richtlinie, der zur Durchführung der Aktion `rds-db:connect` berechtigt ist.

> **Note**
>
> Sie können mehrere Berechtigungssätze oder Rollen mit demselben Datenbank-Benutzerkonto verknüpfen. Nehmen wir beispielsweise an, dass Ihre IAM-Richtlinie folgenden Ressourcen-ARN enthält.
>
> ```
> arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/
> jane_doe
> ```
>
> Wenn Sie die Richtlinie an Jane, Bob und Diego anfügen, sind diese Benutzer in der Lage, eine Verbindung mit dem angegebenen DB-Cluster mithilfe des Datenbankkontos `jane_doe` herzustellen.

Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Mit der IAM-Datenbank-Authentifizierung müssen Sie den von Ihnen erstellten Benutzerkonten keine Datenbankpasswörter zuweisen. Wenn Sie einen mit dem Datenbankkonto verknüpften Benutzer entfernen, sollten Sie auch das Datenbankkonto mit der Anweisung `DROP USER` entfernen.

> **Note**
>
> Der für die IAM-Authentifizierung verwendete Benutzername muss mit der Schreibweise des Benutzernamens in der Datenbank übereinstimmen.

Themen

- Verwenden der IAM-Authentifizierung mit Aurora MySQL
- Verwenden der IAM-Authentifizierung mit Aurora PostgreSQL

Verwenden der IAM-Authentifizierung mit Aurora MySQL

Mit Aurora MySQL erfolgt die Authentifizierung durch `AWSAuthenticationPlugin` – ein von AWS bereitgestelltes Plugin, das reibungslos mit IAM zur Authentifizierung Ihrer Benutzer funktioniert. Stellen Sie als Hauptbenutzer oder als anderer Benutzer, der Benutzer erstellen und Berechtigungen gewähren kann, eine Verbindung mit dem DB-Cluster her. Geben Sie die `CREATE USER`-Anweisung aus, wie im folgenden Beispiel dargestellt.

```
CREATE USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

Die Klausel `IDENTIFIED WITH` ermöglicht Aurora MySQL die Verwendung von `AWSAuthenticationPlugin`, um das Datenbankkonto (`jane_doe`) zu authentifizieren. Die `AS 'RDS'`-Klausel bezieht sich auf die Authentifizierungsmethode. Stellen Sie sicher, dass der angegebene Datenbankbenutzername mit einer Ressource in der IAM-Richtlinie für den IAM-Datenbankzugriff identisch ist. Weitere Informationen finden Sie unter Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff.

> **Note**
>
> Wenn folgende Meldung erscheint, ist das von AWS bereitgestellte Plugin für den aktuellen DB-Cluster nicht verfügbar.
> `ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
> Stellen Sie sicher, dass Sie eine unterstützte Konfiguration verwenden und die IAM-Datenbankauthentifizierung auf Ihrem DB-Cluster aktiviert haben, um den Fehler zu beheben. Weitere Informationen erhalten Sie unter Verfügbarkeit von Regionen und Versionen und Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung.

Nachdem Sie ein Konto mithilfe von `AWSAuthenticationPlugin` erstellt haben, können Sie es in der gleichen Weise wie sonstige Datenbankkonten verwalten. Sie können beispielsweise Kontoberechtigungen mit den Anweisungen `GRANT` und `REVOKE` oder mehrere Kontoattribute mit der Anweisung `ALTER USER` ändern.

Der Datenbank-Netzwerkverkehr wird bei Verwendung von IAM mit SSL/TLS verschlüsselt. Ändern Sie mit dem folgenden Befehl das Benutzerkonto, um SSL-Verbindungen zuzulassen.

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

Verwenden der IAM-Authentifizierung mit Aurora PostgreSQL

Wenn Sie die IAM-Authentifizierung mit Aurora PostgreSQL verwenden möchten, stellen Sie als Hauptbenutzer oder als anderer Benutzer, der Benutzer erstellen und Berechtigungen gewähren kann, eine Verbindung mit dem DB-Cluster her. Nachdem die Verbindung hergestellt wurde, erstellen Sie Datenbankbenutzer und weisen ihnen die `rds_iam`-Rolle zu, wie im folgenden Beispiel dargestellt.

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

Stellen Sie sicher, dass der angegebene Datenbankbenutzername mit einer Ressource in der IAM-Richtlinie für den IAM-Datenbankzugriff identisch ist. Weitere Informationen finden Sie unter Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff. Sie müssen der `rds_iam`-Rolle gestatten, die IAM-Authentifizierung zu verwenden. Sie können auch verschachtelte Mitgliedschaften oder indirekte Zuweisungen der Rolle verwenden.

Beachten Sie, dass ein PostgreSQL-Datenbankbenutzer entweder die IAM- oder Kerberos-Authentifizierung verwenden kann, jedoch nicht beides, so dass dieser Benutzer nicht auch über die `rds_ad`-Rolle verfügen kann. Dies gilt auch für verschachtelte Mitgliedschaften. Weitere Informationen finden Sie unter Schritt 7: Erstellen von PostgreSQL-Benutzern für Ihre Kerberos-Prinzipale .

Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung

Bei einer IAM-Datenbankauthentifizierung verwenden Sie ein Authentifizierungstoken, wenn Sie sich mit Ihrem DB-Cluster verbinden. Ein Authentifizierungstoken ist eine Zeichenfolge, die Sie anstelle eines Passworts verwenden. Nachdem Sie ein Authentifizierungstoken erzeugt haben, ist es 15 Minuten lang gültig. Wenn Sie versuchen, sich mit einem verfallenen Token zu verbinden, wird die Verbindungsabfrage abgelehnt.

Jedes Authentifizierungstoken muss eine gültige Signatur unter Verwendung von AWS Signature Version 4 enthalten. (Weitere Informationen finden Sie unter Signaturprozess mit Signaturversion 4 in

der Allgemeine AWS-Referenz.) Die AWS CLI und ein AWS-SDK, z. B. AWS SDK für Java oder AWS SDK for Python (Boto3) kann jedes von Ihnen erstellte Token automatisch signieren.

Sie können ein Authentifizierungstoken verwenden, wenn Sie von einem anderen AWS-Service aus eine Verbindung zu Amazon Aurora einrichten, beispielsweise AWS Lambda. Durch Verwendung eines Tokens können Sie vermeiden, ein Passwort in Ihrem Code angeben zu müssen. Alternativ können Sie ein AWS SDK verwenden, um ein Authentifizierungstoken programmgesteuert zu erzeugen und programmgesteuert zu signieren.

Nachdem Sie über ein signiertes IAM-Authentifizierungstoken verfügen, können Sie eine Verbindung mit einem Aurora-DB-Cluster herstellen. Nachfolgend erfahren Sie, wie Sie dies mit einem Befehlszeilen-Tool oder mit einem AWS SDK, wie AWS SDK für Java oder AWS SDK for Python (Boto3), erreichen.

Weitere Informationen finden Sie in den folgenden Blogeinträgen:

- IAM-Authentifikation zum Verbinden mit von SQL Workbench/J mit Aurora MySQL oder Amazon RDS for MySQL verwenden
- Verwenden der IAM-Authentifizierung zum Verbinden mit pgAdmin Amazon Aurora PostgreSQL oder Amazon RDS for PostgreSQL

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Themen

- Herstellen einer Verbindung zu Ihrem DB-Cluster mithilfe von IAM-Authentifizierung für die AWS-Treiber
- Herstellen einer Verbindung mit Ihrem DB-Cluster mithilfe der IAM-Authentifizierung von der Befehlszeile aus: AWS CLI und mysql-Client
- Herstellen einer Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung von der Befehlszeile aus: AWS CLI und psql-Client

- Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK für .NET
- Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK für Go
- Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK für Java
- Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK for Python (Boto3)

Herstellen einer Verbindung zu Ihrem DB-Cluster mithilfe von IAM-Authentifizierung für die AWS-Treiber

Die AWS-Treibersuite wurde so konzipiert, dass sie schnellere Umstellungs- und Failover-Zeiten sowie die Authentifizierung mit AWS Secrets Manager, AWS Identity and Access Management (IAM) und Verbundidentität unterstützt. Die AWS-Treiber müssen den Status des DB-Clusters überwachen und die Cluster-Topologie kennen, um den neuen Writer zu ermitteln. Dieser Ansatz reduziert die Umstellungs- und Failover-Zeiten auf Werte im einstelligen Sekundenbereich, im Vergleich zu Werten im zweistelligen Bereich bei Open-Source-Treibern.

Weitere Informationen zu den AWS-Treibern finden Sie in der Dokumentation zum Treiber der entsprechenden Sprache für Ihren Aurora-MySQL- oder Aurora-PostgreSQL-DB-Cluster.

Herstellen einer Verbindung mit Ihrem DB-Cluster mithilfe der IAM-Authentifizierung von der Befehlszeile aus: AWS CLI und mysql-Client

Sie können über die Befehlszeile eine Verbindung mit einem Aurora-DB-Cluster mit der AWS CLI und dem `mysql`-Befehlszeilen-Tool herstellen, wie nachfolgend beschrieben.

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

> **Note**
>
> Informationen zum Herstellen einer Verbindung mit Ihrer Datenbank mithilfe von SQL Workbench/J mit IAM-Authentifizierung finden Sie im Blogbeitrag IAM-Authentifikation zum Verbinden von SQL Workbench/J Aurora MySQL oder Amazon RDS for MySQL verwenden.

Themen

- Generieren eines IAM-Authentifizierungstokens
- Herstellen der Verbindung zu einem DB--Cluster

Generieren eines IAM-Authentifizierungstokens

Im folgenden Beispiel wird gezeigt, wie Sie ein signiertes Authentifizierungstoken mithilfe der erhalte AWS CLI.

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

In diesem Beispiel lauten die Parameter folgendermaßen:

- `--hostname` – Der Hostname des DB--Clusters, auf den Sie zugreifen möchten.
- `--port` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `--region` – Die AWS-Region, in der der DB-Cluster ausgeführt wird.
- `--username` – Das Datenbankkonto, auf das Sie zugreifen möchten.

Die ersten Zeichen des Tokens sehen folgendermaßen aus.

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

Herstellen der Verbindung zu einem DB--Cluster

Das allgemeine Format zur Herstellung einer Verbindung wird nachfolgend dargestellt.

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-cleartext-plugin --user=userName --password=authToken
```

Dabei werden die folgenden Parameter verwendet:

- `--host` – Der Hostname des DB--Clusters, auf den Sie zugreifen möchten.
- `--port` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `--ssl-ca` – Die SSL-Zertifikatsdatei, die den öffentlichen Schlüssel enthält

  Weitere Informationen finden Sie unter TLS-Verbindungen mit DB-Clustern von Aurora MySQL.

  Zum Download eines SSL-Zertifikats siehe Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.
- `--enable-cleartext-plugin` – Ein Wert, der angibt, dass `AWSAuthenticationPlugin` für diese Verbindung zu verwenden ist.

  Wenn Sie einen MariaDB-Client verwenden, ist die `--enable-cleartext-plugin` Option nicht erforderlich.
- `--user` – Das Datenbankkonto, auf das Sie zugreifen möchten.
- `--password` – Ein signiertes IAM-Authentifizierungstoken.

Das Authentifizierungstoken besteht aus hunderten von Zeichen. Dessen Handhabung in der Befehlszeile kann unhandlich sein. Eine Möglichkeit zur Umgehung dieses Problems besteht darin, das Token in einer Umgebungsvariable zu speichern und dann bei der Verbindungsherstellung diese Variable zu verwenden. Im folgenden Beispiel ist eine Möglichkeit dargestellt, um dieses Problem

zu umgehen. Im Beispiel ist */sample_dir/* der vollständige Pfad zur SSL-Zertifikatsdatei, die den öffentlichen Schlüssel enthält.

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

Wenn Sie die Verbindung mithilfe von `AWSAuthenticationPlugin` herstellen, wird die Verbindung mit SSL geschützt. Geben Sie an der `mysql>`-Eingabeaufforderung Folgendes ein, um dies zu überprüfen.

```
show status like 'Ssl%';
```

In den folgenden Linien in der Ausgabe finden Sie weitere Details.

```
+---------------+-------------+
| Variable_name | Value

                                                                        |

+---------------+-------------+
| ...           | ...
| Ssl_cipher    | AES256-SHA

                                                                        |

| ...           | ...
| Ssl_version   | TLSv1.1

                                                                        |

| ...           | ...
+-----------------------------+
```

Informationen dazu, wie Sie über einen Proxy eine Verbindung mit einem DB-Cluster herstellen, finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

Herstellen einer Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung von der Befehlszeile aus: AWS CLI und psql-Client

Sie können über die Befehlszeile eine Verbindung mit einem Aurora-PostgreSQL-DB-Cluster mit der AWS CLI und dem psql-Befehlszeilen-Tool herstellen, wie nachfolgend beschrieben.

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

> **Note**
>
> Informationen zum Herstellen einer Verbindung mit Ihrer Datenbank mithilfe von pgAdmin mit IAM-Authentifizierung finden Sie im Blogbeitrag Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL.

Themen

- Generieren eines IAM-Authentifizierungstokens
- Verbinden mit einem und einem Aurora PostgreSQL-Cluster

Generieren eines IAM-Authentifizierungstokens

Das Authentifizierungstoken besteht aus hunderten von Zeichen, wodurch es in der Befehlszeile unhandlich wird. Eine Möglichkeit zur Umgehung dieses Problems besteht darin, das Token in einer Umgebungsvariable zu speichern und dann bei der Verbindungsherstellung diese Variable zu verwenden. Im folgenden Beispiel wird gezeigt, wie Sie die AWS CLI verwenden, um ein signiertes Authentifizierungstoken mithilfe des `generate-db-auth-token`-Befehls zu erzeugen und dieses anschließend in einer `PGPASSWORD`-Umgebungsvariablen zu speichern.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
```

```
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

In diesem Beispiel lauten die Parameter für den `generate-db-auth-token`-Befehl folgendermaßen:

- `--hostname`– Der Hostname des DB--Clusters (Cluster-Endpunkt), auf den Sie zugreifen möchten.
- `--port` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `--region` – Die AWS-Region, in der der DB-Cluster ausgeführt wird.
- `--username` – Das Datenbankkonto, auf das Sie zugreifen möchten.

Die ersten Zeichen des generierten Tokens sehen folgendermaßen aus.

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

Verbinden mit einem und einem Aurora PostgreSQL-Cluster

Das allgemeine Format zur Herstellung einer Verbindung mithilfe von psql wird nachfolgend dargestellt.

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

Dabei werden die folgenden Parameter verwendet:

- `host`– Der Hostname des DB--Clusters (Cluster-Endpunkt), auf den Sie zugreifen möchten.

- `port` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `sslmode` – Der zu verwendende SSL-Modus.

  Wenn Sie `sslmode=verify-full` verwenden, prüft die SSL-Verbindung den Endpunkt des DB-Clusters gegen den Endpunkt des SSL-Zertifikats.
- `sslrootcert` – Die SSL-Zertifikatsdatei, die den öffentlichen Schlüssel enthält

  Weitere Informationen finden Sie unter Sicherung von Aurora-PostgreSQL-Daten mit SSL/TLS.

  Zum Download eines SSL-Zertifikats siehe Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.
- `dbname` – Die Datenbank, auf die Sie zugreifen möchten.
- `user` – Das Datenbankkonto, auf das Sie zugreifen möchten.
- `password` – Ein signiertes IAM-Authentifizierungstoken.

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

Das folgende Beispiel zeigt die Verwendung von psql für die Verbindung. Im Beispiel verwendet psql die Umgebungsvariable `RDSHOST` für den Host und die Umgebungsvariable `PGPASSWORD` für das generierte Token. Zudem ist */sample_dir/* der vollständige Pfad zur SSL-Zertifikatsdatei, die den öffentlichen Schlüssel enthält.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

Informationen dazu, wie Sie über einen Proxy eine Verbindung mit einem DB-Cluster herstellen, finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK für .NET

Sie können sich mit dem AWS SDK für .NET wie nachfolgend beschrieben mit einer Aurora-MySQL- oder Aurora-PostgreSQL-DB-Cluster verbinden.

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Beispiele

In den folgenden Beispielcodes wird gezeigt, wie Sie ein Authentifizierungstoken erzeugen und anschließend zum Verbinden mit einem DB-Cluster verwenden.

Um dieses Codebeispiel auszuführen, benötigen Sie das AWS SDK für .NET, das sich auf der AWS-Seite befindet. Die `AWSSDK.CORE`- und die `AWSSDK.RDS`-Pakete sind erforderlich. Um eine Verbindung mit einer DB-Cluster herzustellen, verwenden Sie den .NET-Datenbankkonnektor für die DB-Engine, z. B. MySqlConnector für MariaDB oder MySQL oder Npgsql für PostgreSQL.

Dieser Code stellt eine Verbindung mit einem Aurora MySQL DB-Cluster her. Ändern Sie die Werte der folgenden Parameter nach Bedarf.

- `server` – Der Endpunkt des DB--Clusters, auf den Sie zugreifen möchten.
- `user` – Das Datenbankkonto, auf das Sie zugreifen möchten.
- `database` – Die Datenbank, auf die Sie zugreifen möchten.
- `port` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `SslMode` – Der zu verwendende SSL-Modus.

  Wenn Sie `SslMode=Required` verwenden, prüft die SSL-Verbindung den Endpunkt des DB-Clusters gegen den Endpunkt des SSL-Zertifikats.
- `SslCa` – Der vollständige Pfad zum SSL-Zertifikat für Amazon Aurora

Informationen zum Download eines Zertifikats finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new
 MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
      conn.Open();

      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

      // Read all rows and output the first column in each row
```

```
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

Dieser Code stellt eine Verbindung mit einem Aurora PostgreSQL DB-Cluster her.

Ändern Sie die Werte der folgenden Parameter nach Bedarf.

- `Server` – Der Endpunkt des DB--Clusters, auf den Sie zugreifen möchten.
- `User ID` – Das Datenbankkonto, auf das Sie zugreifen möchten.
- `Database` – Die Datenbank, auf die Sie zugreifen möchten.
- `Port` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `SSL Mode` – Der zu verwendende SSL-Modus.

  Wenn Sie `SSL Mode=Required` verwenden, prüft die SSL-Verbindung den Endpunkt des DB-Clusters gegen den Endpunkt des SSL-Zertifikats.
- `Root Certificate` – Der vollständige Pfad zum SSL-Zertifikat für Amazon Aurora

  Informationen zum Download eines Zertifikats finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password={pwd};Database=mydb;SSL
 Mode=Require;Root Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
            conn.Close();
        }
    }
}
```

Informationen dazu, wie Sie über einen Proxy eine Verbindung mit einem DB-Cluster herstellen, finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK für Go

Sie können sich mit dem AWS SDK für Go wie nachfolgend beschrieben mit einer Aurora-MySQL- oder Aurora-PostgreSQL-DB-Cluster verbinden.

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Beispiele

Um diese Codebeispiele auszuführen, benötigen Sie das AWS SDK für Go, das sich auf der AWS-Site befindet.

Ändern Sie die Werte der folgenden Parameter nach Bedarf.

- `dbName` – Die Datenbank, auf die Sie zugreifen möchten.
- `dbUser` – Das Datenbankkonto, auf das Sie zugreifen möchten.
- `dbHost` – Der Endpunkt des DB--Clusters, auf den Sie zugreifen möchten.

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

- `dbPort` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `region` – Die AWS-Region, in der der DB-Cluster ausgeführt wird.

Stellen Sie außerdem sicher, dass die importierten Bibliotheken im Beispielcode auf Ihrem System vorhanden sind.

> **Important**
>
> Die Beispiele in diesem Abschnitt verwenden den folgenden Code, um Anmeldeinformationen bereitzustellen, die von einer lokalen Umgebung aus auf eine Datenbank zugreifen:
> `creds := credentials.NewEnvCredentials()`
> Wenn Sie von einem AWS-Service wie Amazon EC2 oder Amazon ECS auf eine Datenbank zugreifen, können Sie den Code durch den folgenden Code ersetzen:

```
sess := session.Must(session.NewSession())
creds := sess.Config.Credentials
```

Wenn Sie diese Änderung vornehmen, stellen Sie sicher, dass Sie den folgenden Import hinzufügen:

```
"github.com/aws/aws-sdk-go/aws/session"
```

Themen

- Herstellen einer Verbindung mit IAM-Authentifizierung und AWS SDK für Go V2
- Herstellen einer Verbindung mit IAM-Authentifizierung und AWS SDK für Go V1.

Herstellen einer Verbindung mit IAM-Authentifizierung und AWS SDK für Go V2

Sie können mithilfe der IAM-Authentifizierung und der AWS SDK für Go V2 eine Verbindung zu einem herstellen.

In den folgenden Beispielcodes wird gezeigt, wie Sie ein Authentifizierungstoken erzeugen und anschließend zum Verbinden mit einem DB-Cluster verwenden.

Dieser Code stellt eine Verbindung mit einem Aurora MySQL DB-Cluster her.

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/go-sql-driver/mysql"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 3306
```

```
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Dieser Code stellt eine Verbindung mit einem Aurora PostgreSQL DB-Cluster her.

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)
```

```
func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Informationen dazu, wie Sie über einen Proxy eine Verbindung mit einem DB-Cluster herstellen, finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

Herstellen einer Verbindung mit IAM-Authentifizierung und AWS SDK für Go V1.

Sie können mithilfe der IAM-Authentifizierung und der AWS SDK für Go V1 eine Verbindung zu einem herstellen

In den folgenden Beispielcodes wird gezeigt, wie Sie ein Authentifizierungstoken erzeugen und anschließend zum Verbinden mit einem DB-Cluster verwenden.

Dieser Code stellt eine Verbindung mit einem Aurora MySQL DB-Cluster her.

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
```

```
}
```

Dieser Code stellt eine Verbindung mit einem Aurora PostgreSQL DB-Cluster her.

```
package main

import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Informationen dazu, wie Sie über einen Proxy eine Verbindung mit einem DB-Cluster herstellen, finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK für Java

Sie können sich mit dem AWS SDK für Java wie nachfolgend beschrieben mit einer Aurora-MySQL- oder Aurora-PostgreSQL-DB-Cluster verbinden.

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung
- Richten Sie dasAWS SDK für Java ein

Beispiele zur Verwendung des SDK für Java 2.x finden Sie unter Amazon RDS examples using SDK for Java 2.x. Sie können auch den fortschrittlichen AWS-JDBC-Wrapper verwenden. Weitere Informationen finden Sie in der Dokumentation zum fortschrittlichen AWS-JDBC-Wrapper.

Themen

- Generieren eines IAM-Authentifizierungstokens
- Manuelles Erzeugen eines IAM-Authentifizierungstokens
- Herstellen der Verbindung zu einem DB--Cluster

Generieren eines IAM-Authentifizierungstokens

Wenn Sie Programme mit AWS SDK für Java schreiben, können Sie das signierte Authentifizierungstoken unter Verwendung der `RdsIamAuthTokenGenerator`-Klasse erhalten. Zur Verwendung dieser Klasse müssen Sie AWS-Anmeldeinformationen bereitstellen. Erstellen Sie dazu eine Instance der Klasse `DefaultAWSCredentialsProviderChain`. `DefaultAWSCredentialsProviderChain` verwendet den ersten AWS-Zugriffsschlüssel und privaten Schlüssel, den es in der standardmäßigen Anbieterkette von Anmeldeinformationen findet. Weitere Informationen über AWS-Zugriffsschlüssel finden Sie unter Verwalten von Zugriffsschlüsseln für Benutzer.

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

Nachdem Sie eine Instance von `RdsIamAuthTokenGenerator` erstellt haben, können Sie das `getAuthToken`-Verfahren aufrufen, um ein signiertes Token zu erhalten. Geben Sie die AWS-Region, den Hostnamen, die Portnummer und den Benutzernamen an. Der folgende Beispielcode veranschaulicht diese Vorgehensweise.

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();

     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
      .hostname(hostName)
      .port(Integer.parseInt(port))
```

```
      .userName(username)
      .build());

     return authToken;
    }

}
```

Manuelles Erzeugen eines IAM-Authentifizierungstokens

Der einfachste Weg in Java, um ein Authentifizierungstoken zu erzeugen, ist die Verwendung von `RdsIamAuthTokenGenerator`. Diese Klasse erzeugt ein Authentifizierungstoken für Sie und signiert es anschließend mithilfe der AWS Signature Version 4. Weitere Informationen finden Sie unter Signaturprozess mit Signaturversion 4 im Allgemeine AWS-Referenz.

Sie können allerdings das Authentifizierungstoken auch manuell erstellen und signieren, wie im folgenden Beispielcode dargestellt.

```
package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
```

```
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
        DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
    String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
    String awsSecretKey = creds.getCredentials().getAWSSecretKey();
        String expiryMinutes = "900";

        System.out.println("Step 1:  Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2:  Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3:  Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();

        System.out.println("Step 4:  Add the signing info to the request");

        System.out.println(appendSignature(signature));
```

```
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
 should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String
 date, String dateTime, String region, String expiryPeriod, String hostName, String
 port) throws Exception {
        canonicalQueryParameters.put("Action", action);
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
"%2F" + region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
 canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
 currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
 + '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
 String accessKey, String date, String region) throws Exception {
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
 BinaryUtils.toHex(hash(canonicalRequest));
```

```
    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
                          SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String
serviceName) {
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(String stringData, byte[] key,
                        SigningAlgorithm algorithm) throws SdkClientException {
        try {
            byte[] data = stringData.getBytes(UTF8);
```

```
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

Herstellen der Verbindung zu einem DB--Cluster

Im folgenden Codebeispiel wird gezeigt, wie Sie ein Authentifizierungstoken erzeugen und anschließend zum Verbinden mit einem Cluster für die Ausführung mit Aurora MySQL verwenden können.

Um dieses Codebeispiel auszuführen, benötigen Sie das AWS SDK für Java, das sich auf der AWS-Seite befindet. Außerdem benötigen Sie Folgendes:

- MySQL Connector/J. Dieses Codebeispiel wurde mit `mysql-connector-java-5.1.33-bin.jar` getestet.
- Ein Zwischenzertifikat für Amazon Aurora einer bestimmten AWS-Region. (Weitere Informationen finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.) Während der Laufzeit sucht der Class Loader das Zertifikat in demselben Verzeichnis, in dem sich dieser Java-Beispielcode befindet, damit er es finden kann.

- Ändern Sie die Werte der folgenden Parameter nach Bedarf.
  - `RDS_INSTANCE_HOSTNAME` – Der Hostname des DB--Clusters, auf den Sie zugreifen möchten.
  - `RDS_INSTANCE_PORT` – Die Nummer des Ports, der für die Verbindung zum PostgreSQL-DB-Cluster verwendet wird.
  - `REGION_NAME` – Die AWS-Region, in der der DB-Cluster ausgeführt wird.
  - `DB_USER` – Das Datenbankkonto, auf das Sie zugreifen möchten.
  - `SSL_CERTIFICATE` – Ein Zwischenzertifikat für Amazon Aurora einer bestimmten AWS-Region.

    Informationen zum Herunterladen eines Zertifikats für Ihre AWS-Region finden Sie unter Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln. Speichern Sie das SSL-Zertifikat in demselben Verzeichnis wie diese Java-Programmdatei ab, damit der Class Loader das Zertifikat während der Laufzeit finden kann.

Dieser Beispielcode erhält die AWS-Anmeldeinformationen von der default credential provider chain.

> **Note**
>
> Geben Sie aus Sicherheitsgründen für `DEFAULT_KEY_STORE_PASSWORD` ein anderes Passwort als hier angegeben an.

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication
 token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
```

```
        }

        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
 Database Authentication
     * @return
     * @throws Exception
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
 Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }

    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
```

```
     * @return
     */
    private static String generateAuthToken() {
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
                .credentials(new
AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

.hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
     * This method sets the SSL properties which specify the key store file, its type
and password:
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword",
DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL
verification during the IAM Database Authentication to
     * the db instance.
     * @return
     * @throws Exception
     */
    private static String createKeyStoreFile() throws Exception {
        return createKeyStoreFile(createCertificate()).getPath();
    }

    /**
     *  This method generates the SSL certificate
     * @return
     * @throws Exception
     */
    private static X509Certificate createCertificate() throws Exception {
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
```

```
        URL url = new File(SSL_CERTIFICATE).toURI().toURL();
        if (url == null) {
            throw new Exception();
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
 Exception {
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
 KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
           System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

Informationen dazu, wie Sie über einen Proxy eine Verbindung mit einem DB-Cluster herstellen, finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

Herstellen einer Verbindung zu Ihrem DB--Cluster mithilfe der IAM-Authentifizierung und AWS SDK for Python (Boto3)

Sie können sich mit dem AWS SDK for Python (Boto3) wie nachfolgend beschrieben mit einer Aurora-MySQL- oder Aurora-PostgreSQL-DB-Cluster verbinden.

Voraussetzungen

Die folgenden Voraussetzungen gelten für die Verbindung mit Ihrem DB--Cluster mithilfe der IAM-Authentifizierung:

- Aktivieren und Deaktivieren der IAM-Datenbank-Authentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Stellen Sie außerdem sicher, dass die importierten Bibliotheken im Beispielcode auf Ihrem System vorhanden sind.

Beispiele

Die Codebeispiele verwenden Profile für freigegebene Anmeldeinformationen. Informationen zum Angeben von Anmeldeinformationen finden Sie unter Anmeldeinformationen in der AWS SDK for Python (Boto3)-Dokumentation.

In den folgenden Beispielcodes wird gezeigt, wie Sie ein Authentifizierungstoken erzeugen und anschließend zum Verbinden mit einem DB-Cluster verwenden.

Um dieses Codebeispiel auszuführen, benötigen Sie das AWS SDK for Python (Boto3), das sich auf der AWS-Seite befindet.

Ändern Sie die Werte der folgenden Parameter nach Bedarf.

- `ENDPOINT` – Der Endpunkt des DB--Clusters, auf den Sie zugreifen möchten.
- `PORT` – Die Nummer des Ports, der für die Verbindung mit dem DB--Cluster verwendet wird.
- `USER` – Das Datenbankkonto, auf das Sie zugreifen möchten.
- `REGION` – Die AWS-Region, in der der DB-Cluster ausgeführt wird.
- `DBNAME` – Die Datenbank, auf die Sie zugreifen möchten.

- `SSLCERTIFICATE` – Der vollständige Pfad zum SSL-Zertifikat für Amazon Aurora

  Geben Sie für `ssl_ca` ein SSL-Zertifikat an. Zum Download eines SSL-Zertifikats siehe Wird verwendet SSL/TLS , um eine Verbindung zu einer zu verschlüsseln.

> **Note**
>
> Sie können keinen benutzerdefinierten Route-53-DNS-Eintrag oder einen benutzerdefinierten Aurora-Endpunkt anstelle des DB-Cluster-Endpunkts verwenden, um das Authentifizierungstoken zu generieren.

Dieser Code stellt eine Verbindung mit einem Aurora MySQL DB-Cluster her.

Bevor Sie diesen Code ausführen, installieren Sie den PyMySQL-Treiber, indem Sie die Anweisungen im Python-Paketindex befolgen.

```
import pymysql
import sys
import boto3
import os

ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =
 pymysql.connect(auth_plugin_map={'mysql_clear_password':None},host=ENDPOINT,
 user=USER, password=token, port=PORT, database=DBNAME, ssl_ca='SSLCERTIFICATE',
 ssl_verify_identity=True, ssl_verify_cert=True)
```

```
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

Dieser Code stellt eine Verbindung mit einem Aurora PostgreSQL DB-Cluster her.

Bevor Sie diesen Code ausführen, installieren Sie `psycopg2`, indem Sie die Anweisungen in der Psycopg-Dokumentation befolgen.

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

Informationen dazu, wie Sie über einen Proxy eine Verbindung mit einem DB-Cluster herstellen, finden Sie unter Mithilfe der IAM-Authentifizierung eine Verbindung zu einer Datenbank herstellen.

Problembehebung bei der IAM-DB-Authentifizierung

Im Folgenden finden Sie Tipps zur Problembehebung für einige häufige IAM-DB-Authentifizierungsprobleme und Informationen zu CloudWatch-Protokollen und Metriken für die IAM-DB-Authentifizierung.

Exportieren von Fehlerprotokolle zur IAM-DB-Authentifizierung in CloudWatch Logs

Fehlerprotokolle zur IAM-DB-Authentifizierung werden auf dem Datenbank-Host gespeichert und Sie können diese Protokolle in Ihr CloudWatch-Logs-Konto exportieren. Verwenden Sie die Protokolle und Problembehebungsmethoden auf dieser Seite, um Probleme mit der IAM-DB-Authentifizierung zu beheben.

Sie können Protokollexporte nach CloudWatch Logs über die Konsole, die AWS CLI und die RDS-API aktivieren. Anleitungen zur Konsole finden Sie unter Veröffentlichen von Datenbankprotokollen in Amazon CloudWatch Logs.

Verwenden Sie den folgenden Befehl, um Ihre IAM-DB-Authentifizierungsfehlerprotokolle nach CloudWatch Logs zu exportieren, wenn Sie einen DB-Cluster über die AWS CLI erstellen:

```
aws rds create-db-cluster --db-cluster-identifier mydbinstance \
--region us-east-1 \
--engine postgres \
--engine-version 16 \
--master-username master \
--master-user-password password \
--publicly-accessible \
--enable-iam-database-authentication \
--enable-cloudwatch-logs-exports=iam-db-auth-error
```

Verwenden Sie den folgenden Befehl, um Ihre IAM-DB-Authentifizierungsfehlerprotokolle nach CloudWatch Logs zu exportieren, wenn Sie einen DB-Cluster über die AWS CLI ändern:

```
aws rds modify-db-cluster --db-cluster-identifier mydbcluster \
--region us-east-1 \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["iam-db-auth-error"]}'
```

Um zu bestätigen, ob Ihr DB-Cluster IAM-DB-Authentifizierungsprotokolle nach CloudWatch Logs exportiert, überprüfen Sie, ob der Parameter `EnabledCloudwatchLogsExports` in der Ausgabe für den Befehl `describe-db-instances` auf `iam-db-auth-error` festgelegt ist.

```
aws rds describe-db-cluster --region us-east-1 --db-cluster-identifier mydbcluster
            ...

             "EnabledCloudwatchLogsExports": [
                "iam-db-auth-error"
            ],
            ...
```

CloudWatch-Metriken zur IAM-DB-Authentifizierung

Amazon Aurora liefert nahezu in Echtzeit Metriken zur IAM-DB-Authentifizierung für Ihr Amazon-CloudWatch-Konto. Die folgende Tabelle listet die IAM-DB-Authentifizierungsmetriken auf, die in CloudWatch verfügbar sind:

Metrik	Beschreibung
`IamDbAuthConnectionRequests`	Gesamtzahl der Verbindungsanfragen, die mit der IAM-DB-Authentifizierung gestellt wurden
`IamDbAuthConnectionSuccess`	Gesamtzahl der erfolgreichen IAM-DB-Authentifizierungsanfragen
`IamDbAuthConnectionFailure`	Gesamtzahl der fehlgeschlagenen IAM-DB-Authentifizierungsanfragen
`IamDbAuthConnectionFailureInvalidToken`	Gesamtzahl der aufgrund ungültiger Token fehlgeschlagenen IAM-DB-Authentifizierungsanfragen
`IamDbAuthConnectionFailureInsufficientPermissions`	Gesamtzahl der aufgrund falscher Richtlinien oder Berechtigungen fehlgeschlagenen IAM-DB-Authentifizierungsanfragen

Metrik	Beschreibung
`IamDbAuthConnectionFailureThrottling`	Gesamtzahl der aufgrund der Drosselung der IAM-DB-Authentifizierung fehlgeschlagenen IAM-DB-Authentifizierungsanfragen
`IamDbAuthConnectionFailureServerError`	Gesamtzahl der aufgrund interner Serverfehler im IAM-DB-Authentifizierungsfeature fehlgeschlagenen IAM-DB-Authentifizierungsanfragen

Häufige Probleme und Lösungen

Bei der Verwendung der IAM-DB-Authentifizierung können die folgenden Probleme auftreten. Verwenden Sie die Schritte zur Problembehebung in der Tabelle, um die Probleme zu lösen:

Fehler	Metrik(en)	Ursache	Lösung

`[ERROR] Failed to authenticate the connection request for user` *`db_user`* `because the provided token is malformed or otherwise invalid. (Status Code: 400, Error Code: InvalidToken)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureInvalidToken`
		Das IAM-DB-Authentifizierungs-Token in der Verbindungsanfrage ist entweder kein gültiges SigV4A-Token oder es ist nicht richtig formatiert.	Überprüfen Sie Ihre Strategie zur Token-Generierung in Ihrer Anwendung. In einigen Fällen müssen Sie sicherstellen, dass Sie das Token mit einer gültigen Formatierung übergeben. Wenn Sie das Token kürzen (oder eine falsche Zeichenkettenformatierung verwendet wurde), wird das Token ungültig.

`[ERROR] Failed to authenticate the connection`	`IamDbAuthConnectionFailure`	Das IAM-DB-Authentifizierungs-Token ist abgelaufe	Überprüfen Sie Ihre Token-Zwischenspeicherung und/oder

Fehler	Metrik(en)	Ursache	Lösung
`request for user db_user because the token age is longer than 15 minutes. (Status Code: 400, Error Code:ExpiredToken)`	`IamDbAuthConnectionFailureInvalidToken`	n. Tokens sind nur 15 Minuten lang gültig.	Token-Wiederverwendungslogik in Ihrer Anwendung. Sie sollten Tokens, die älter als 15 Minuten sind, nicht wiederverwenden.

`[ERROR] Failed to authorize the connection request for user db_user because the IAM policy assumed by the caller 'arn:aws:sts::123456789012:assumed-role/ <RoleName>/ <RoleSession>' is not authorized to perform `rds-db:connect` on the DB instance. (Status Code: 403, Error Code:NotAuthorized)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureInsufficientPermissions`
		Dies kann folgende Ursachen haben:
		• Die von der Anwendung angenommene IAM-Richtlinie autorisiert die Aktion `rds-db:connect` nicht.
		• Sie nehmen die falsche Rolle/Richtlinie für *db_user* an, um eine Verbindung zur Datenbank herzustellen.
		• Sie nehmen die richtige Richtlinie für *db_user* an, stellen aber keine Verbindung zur richtigen Datenbank her.
			Stellen Sie sicher, dass Sie die IAM-Rolle und/oder -Richtlinie in Ihrer Anwendung übernehmen. Stellen Sie sicher, dass Sie für die Generierung des Tokens dieselbe Richtlinie verwenden wie für die Verbindung mit der DB.

Fehler	Metrik(en)	Ursache	Lösung

`[ERROR] Failed to authorize the connection request for user db_user due to IAM DB authentication throttling. (Status Code: 429, Error Code: ThrottlingException)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureThrottling`
		Sie stellen in kurzer Zeit zu viele Verbindungsanforderungen an Ihre DB. Das Limit für die Drosselung der IAM-DB-Authentifizierung liegt bei 200 Verbindungen pro Sekunde.	Reduzieren Sie die Geschwindigkeit, mit der neue Verbindungen hergestellt werden, mit der IAM-Authentifizierung. Erwägen Sie die Implementierung von Verbindungspooling mithilfe von RDS-Proxy, um vorhandene Verbindungen in Ihrer Anwendung wiederzuverwenden.

`[ERROR] Failed to authorize the connection request for user db_user due to an internal IAM DB authentication error. (Status Code: 500, Error Code: InternalError)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureThrottling`
		Bei der Autorisierung der DB-Verbindung mit der IAM-DB-Authentifizierung ist ein interner Fehler aufgetreten.	Wenden Sie sich an https://aws.amazon.com/premiumsupport/, um das Problem zu untersuchen.

Fehlerbehebung für Amazon Aurora-Identität und -Zugriff

Verwenden Sie die folgenden Informationen, um häufige Probleme zu diagnostizieren und zu beheben, die beim Arbeiten mit Aurora und IAM auftreten könnten.

Themen

- Ich bin nicht autorisiert, eine Aktion in Aurora auszuführen

- Ich bin nicht zum Durchführen von iam:PassRole berechtigt
- Ich möchte Personen außerhalb meines AWS-Kontos Zugriff auf meine Aurora-Ressourcen erteilen

Ich bin nicht autorisiert, eine Aktion in Aurora auszuführen

Wenn die AWS-Managementkonsole Ihnen mitteilt, dass Sie nicht zur Ausführung einer Aktion autorisiert sind, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Der folgende Beispielfehler tritt auf, wenn der Benutzer `mateojackson` versucht, die Konsole zum Anzeigen von Details zu einem *`Widget`* zu verwenden, jedoch nicht über `rds:`*`GetWidget`*-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

In diesem Fall bittet Mateo seinen Administrator um die Aktualisierung seiner Richtlinien, um unter Verwendung der Aktion *`my-example-widget`* auf die Ressource `rds:`*`GetWidget`* zugreifen zu können.

Ich bin nicht zum Durchführen von iam:PassRole berechtigt

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Ausführung der Aktion `iam:PassRole` autorisiert sind, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt. Bitten Sie diese Person um die Aktualisierung Ihrer Richtlinien, um eine Rolle an Aurora übergeben zu können.

Einige AWS-Services gewähren Ihnen die Berechtigung zur Übergabe einer vorhandenen Rolle an diesen Service, statt eine neue Service-Rolle oder serviceverknüpfte Rolle erstellen zu müssen. Hierfür benötigen Sie Berechtigungen zur Übergabe der Rolle an den Service.

Der folgende Beispielfehler tritt auf, wenn ein Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in Aurora auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Service-Rolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall bittet Mary ihren Administrator um die Aktualisierung ihrer Richtlinien, um die Aktion `iam:PassRole` ausführen zu können.

Ich möchte Personen außerhalb meines AWS-Kontos Zugriff auf meine Aurora-Ressourcen erteilen

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Im Fall von Diensten, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (Access Control Lists, ACLs) verwenden, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen finden Sie hier:

- Informationen dazu, ob Aurora diese Funktionen unterstützt, finden Sie unter Funktionsweise von Amazon Aurora mit IAM.
- Informationen zum Gewähren des Zugriffs auf Ihre Ressourcen für alle Ihre AWS-Konten finden Sie unter Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen Ihrer AWS-Konten im IAM-Benutzerhandbuch.
- Informationen dazu, wie Sie AWS-Drittkonten Zugriff auf Ihre Ressourcen bereitstellen, finden Sie unter Gewähren des Zugriffs auf AWS-Konten von externen Benutzern im IAM-Benutzerhandbuch.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien im IAM-Benutzerhandbuch.

Protokollieren und Überwachen in Amazon Aurora

Die Überwachung ist ein wichtiger Teil der Aufrechterhaltung von Zuverlässigkeit, Verfügbarkeit und Leistung von Amazon Aurora und Ihren AWS-Lösungen. Sammeln Sie Überwachungsdaten aller Bestandteile Ihrer AWS-Lösung, damit Sie Ausfälle an mehreren Punkten leichter debuggen können. AWS bietet mehrere Tools für die Überwachung Ihrer Amazon Aurora-Ressourcen und die Reaktion auf mögliche Vorfälle:

Amazon-CloudWatch-Alarme

Mit Amazon-CloudWatch-Alarmen überwachen Sie eine Metrik über einen von Ihnen definierten Zeitraum. Wenn die Metrik einen bestimmten Schwellenwert überschreitet, wird eine Benachrichtigung an ein Amazon SNS-Thema oder eine AWS Auto Scaling-Richtlinie gesendet. CloudWatch-Alarme rufen keine Aktionen auf, nur weil sie einen besonderen Status haben. Der Status muss sich stattdessen geändert haben und für eine festgelegte Anzahl an Zeiträumen aufrechterhalten worden sein.

AWS CloudTrail-Protokolle

CloudTrail bietet eine Aufzeichnung der von einem Benutzer, einer Rolle oder einem AWS-Service in Amazon Aurora durchgeführten Aktionen. CloudTrail erfasst alle API-Aufrufe für Amazon Aurora als Ereignisse, einschließlich Aufrufen von der Konsole und von Code-Aufrufen an die Amazon-RDS-API-Vorgänge. Mit den von CloudTrail gesammelten Informationen können Sie die an Amazon Aurora gestellte Anforderung, die IP-Adresse, von der die Anforderung gestellt wurde, den Initiator der Anforderung, den Zeitpunkt der Anforderung und weitere Angaben bestimmen. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora-API-Aufrufen in AWS CloudTrail.

„Enhanced Monitoring“ (Erweiterte Überwachung)

Amazon Aurora stellt in Echtzeit Metriken für das Betriebssystem (BS) bereit, auf dem Ihren DB-Cluster ausgeführt wird. Sie können die Metriken für Ihren DB-Cluster über die Konsole anzeigen oder die Enhanced Monitoring-JSON-Ausgabe aus Amazon CloudWatch Logs in einem Überwachungssystem Ihrer Wahl nutzen. Weitere Informationen finden Sie unter Überwachen von Betriebssystem-Metriken mithilfe von „Enhanced Monitoring“·(Erweiterte·Überwachung).

Erkenntnisse zur Amazon-RDS-Leistung

Performance Insights lässt sich auf vorhandene Amazon Aurora-Überwachungsfunktionen erweitern, damit Sie die Performance Ihrer Datenbank darstellen und mögliche Probleme analysieren können. Mit dem Performance-Insights-Dashboard können Sie die Datenbankauslastung visualisieren und die Auslastung nach Wartezeiten, SQL-Anweisungen, Hosts oder Benutzern filtern. Weitere Informationen finden Sie unter Überwachung mit Performance Insights auf .

Datenbankprotokolle

Sie können Datenbankprotokolle mithilfe der AWS-Managementkonsole, AWS CLI oder RDS-API anzeigen, herunterladen und ansehen. Weitere Informationen finden Sie unter Überwachen von Amazon Aurora-Protokolldateien.

Amazon Aurora-Empfehlungen

Amazon Aurora bietet automatisierte Empfehlungen für Datenbankressourcen. Diese Empfehlungen bieten Anleitungen nach bewährten Methoden, indem sie die Konfigurations-, Nutzungs- und Performance-Daten des DB-Clusters analysieren. Weitere Informationen finden Sie unter Empfehlungen von Amazon Aurora.

Amazon Aurora-Ereignisbenachrichtigung

Amazon Aurora verwendet Amazon Simple Notification Service (Amazon SNS), um Benachrichtigungen zu senden, wenn ein Amazon Aurora-Ereignis stattfindet. Diese Benachrichtigungen können jedes von Amazon SNS für eine AWS-Region unterstützte Format aufweisen, wie zum Beispiel eine E-Mail, eine SMS oder ein Anruf an einen HTTP-Endpunkt. Weitere Informationen finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen.

AWS Trusted Advisor

Trusted Advisor stützt sich auf bewährte Methoden, die sich während der gesamten Betriebsgeschichte der Betreuung vieler Hunderttausend AWS-Kunden ergeben haben. Trusted Advisor überprüft Ihre AWS-Umgebung und gibt dann Empfehlungen, sobald sich Möglichkeiten ergeben, Kosten zu senken, die Systemleistung zu verbessern oder Schwachstellen zu schließen. Alle AWS-Kunden haben Zugriff auf fünf Trusted Advisor-Prüfungen. Kunden mit dem „Business“- oder „Enterprise“-Support-Plan können alle Trusted Advisor-Überprüfungen anzeigen.

Trusted Advisor bietet die folgenden Amazon-Aurora-bezogenen Überprüfungen:

- Inaktive DB-Instances von Amazon Aurora
- Zugriffsrisiko für Amazon Aurora-Sicherheitsgruppen
- Amazon Aurora-Sicherungen
- Multi-AZ von Amazon Aurora
- Barrierefreiheit von Aurora-DB-Instances

Weitere Informationen zu diesen Prüfungen finden Sie unter Trusted Advisor – bewährte Methoden (Prüfungen).

Datenbankaktivitäts-Streams

Datenbankaktivitäts-Streams schützen Ihre Datenbanken vor internen Bedrohungen durch Kontrolle des DBA-Zugriffs auf die Datenbankaktivitäts-Streams. Die DBAs, die die Datenbank verwalten, haben folglich keine Zugriffsrechte für die Erfassung, Übertragung, Speicherung und nachfolgende Verarbeitung des Datenbankaktivitäts-Streams. Datenbankaktivitäts-

Streams können Sicherheitsmaßnahmen für Ihre Datenbank bereitstellen und die Compliance-Anforderungen und behördlichen Vorgaben erfüllen. Weitere Informationen finden Sie unter Überwachung von Amazon Aurora mithilfe von Datenbankaktivitätsstreams.

Weitere Informationen zur Überwachung von Aurora finden Sie unter Überwachung von Metriken in einem Amazon-Aurora-Cluster .

Compliance-Validierung für Amazon Aurora

Externe Auditoren bewerten im Rahmen verschiedener AWS-Compliance-Programme die Sicherheit und Compliance von Amazon Aurora. Hierzu zählen unter anderem SOC, PCI, FedRAMP und HIPAA.

Eine Liste der AWS-Services, die in den Geltungsbereich bestimmter Compliance-Programme fallen, finden Sie auf der Seite AWS-Services in Scope nach Compliance-Programm. Allgemeine Informationen finden Sie unter AWS-Compliance-Programme.

Die Auditberichte von Drittanbietern lassen sich mit herunterlade AWS Artifact. Weitere Informationen finden Sie unter Herunterladen von Berichten in AWS Artifact.

Ihre Compliance-Verantwortung bei der Verwendung von Amazon Aurora wird durch die Sensibilität Ihrer Daten, die Compliance-Ziele Ihrer Organisation und die geltenden Gesetze und Vorschriften bestimmt. AWS stellt die folgenden Ressourcen zur Unterstützung der Compliance bereit:

- Kurzanleitungen für Sicherheit und Compliance – In diesen Bereitstellungsleitfäden finden Sie wichtige Überlegungen zur Architektur sowie die einzelnen Schritte zur Bereitstellung von sicherheits- und Compliance-orientierten Basisumgebungen in AWS.
- Erstellung einer Architektur mit HIPAA-konformer Sicherheit und Compliance in Amazon Web Services – In diesem Whitepaper wird beschrieben, wie Unternehmen mithilfe von AWS HIPAA-konforme Anwendungen erstellen können.
- AWS-Compliance-Ressourcen – Diese Sammlung von Arbeitsmappen und Leitfäden könnte für Ihre Branche und Ihren Standort interessant sein.
- AWS Config – Dieser AWS-Service bewertet, zu welchem Grad die Konfiguration Ihrer Ressourcen den internen Vorgehensweisen, Branchenrichtlinien und Vorschriften entspricht.
- AWS Security Hub CSPM: Dieser AWS-Service bietet einen umfassenden Überblick über Ihren Sicherheitsstatus innerhalb von AWS. Security Hub verwendet Sicherheitskontrollen, um Ihre AWS-Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Die Liste der unterstützten Services und Kontrollen finden Sie in der Security-Hub-Steuerelementreferenz.

Ausfallsicherheit in Amazon Aurora

Im Zentrum der globalen AWS-Infrastruktur stehen die AWS-Regionen und Availability Zones (Verfügbarkeitszonen, AZs). AWS Regionen stellen mehrere physisch getrennte und isolierte Availability Zones bereit, die mit Netzwerken mit geringer Latenz, hohem Durchsatz und hochredundanten Vernetzungen verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen über AWS-Regionen und -Availability Zones finden Sie unter Globale AWS-Infrastruktur.

Neben der globalen AWS-Infrastruktur stellt Aurora Funktionen bereit, um Ihren Anforderungen an Ausfallsicherheit und Datensicherung gerecht zu werden.

Backup und Backup

Aurora sichert Ihr Cluster-Volume automatisch und bewahrt die Wiederherstellungsdaten für die Länge des Aufbewahrungszeitraums für Backups auf. Aurora-Backups sind kontinuierlich und inkrementell, sodass Sie schnell eine Backup auf einen beliebigen Punkt im Aufbewahrungszeitraum für Backups durchführen können. Es gibt keine Auswirkungen auf die Leistungsfähigkeit oder Unterbrechung von Datenbank-Services, während Backup-Daten geschrieben werden. Sie können einen Aufbewahrungszeitraum für Backups von 1 bis 35 Tagen festlegen, wenn Sie Ihr DB-Cluster erstellen oder ändern.

Wenn Sie ein Backup nach dem eingestellten Aufbewahrungszeitraum behalten möchten, können Sie auch einen Snapshot der Daten Ihres Cluster-Volumes machen. Aurora bewahrt für den gesamten Aufbewahrungszeitraum für Backups inkrementelle Wiederherstellungsdaten auf. Daher müssen Sie einen Snapshot nur für Daten erstellen, die Sie über die Aufbewahrungsfrist hinaus aufbewahren möchten. Sie können ein neues DB-Cluster aus dem Snapshot erstellen.

Sie können Ihre Daten wiederherstellen, indem Sie ein neues Aurora-DB-Cluster aus den von Aurora aufbewahrten Sicherungsdaten oder aus einem von Ihnen gespeicherten DB-Cluster-Snapshot erstellen. Sie können aus den Sicherungsdaten zu einem beliebigen Zeitpunkt während Ihrer Aufbewahrungsfrist schnell eine neue Kopie eines DB-Clusters erstellen. Das Prinzip der fortlaufenden und ansteigenden Aurora-Backups während des Aufbewahrungszeitraums für

Backups bedeutet, dass Sie keine häufigen Snapshots Ihrer Daten machen müssen, um die Wiederherstellungszeiten zu optimieren.

Weitere Informationen finden Sie unter Sichern und Wiederherstellen eines Amazon-Aurora-DB-Clusters.

Replikation

Aurora-Replicas sind unabhängige Endpunkte in einem Aurora-DB-Cluster, die die beste Methode für das Skalieren von Leseoperationen und Erhöhen der Verfügbarkeit darstellen. Es können bis zu 15 Aurora-Replikate über die Availability Zones verteilt werden, über die sich ein DB-Cluster innerhalb einer AWS-Region erstreckt. Das DB-Cluster-Volume besteht aus mehreren Kopien der Daten für den DB-Cluster. Die Daten im Cluster-Volume werden jedoch als ein einzelnes logisches Volume der primären DB-Instance und der Aurora Replicas im DB-Cluster dargestellt. Wenn die primäre DB-Instance ausfällt, wird ein Aurora-Replica zur primären DB-Instance hochgestuft.

Aurora unterstützt außerdem für Aurora MySQL und Aurora PostgreSQL spezifische Replikationsoptionen.

Weitere Informationen finden Sie unter Replikation mit Amazon Aurora.

Failover

Aurora speichert Kopien der Daten in einem DB-Cluster über mehrere Availability Zones in einer einzigen AWS-Region. Dieser Speichervorgang erfolgt unabhängig davon, ob die DB-Instances im DB-Cluster mehrere Availability Zones umfassen. Wenn Sie Aurora Replicas über mehrere Availability Zones hinweg erstellen, stellt Aurora diese automatisch bereit und verwaltet sie synchron. Die primäre DB-Instance wird synchron über mehrere Availability Zones zu einer Aurora Replica repliziert, um für Datenredundanz zu sorgen, das Einfrieren von I/O-Vorgängen zu vermeiden und Latenzspitzen im Verlauf von Systemsicherungen zu minimieren. Wenn Sie einen DB-Cluster mit hoher Verfügbarkeit ausführen, kann dies die Verfügbarkeit bei geplanten Systemwartungen verbessern. Außerdem kann dies Ihre Datenbanken bei Ausfällen und bei Nichtverfügbarkeit von Availability Zones schützen.

Weitere Informationen finden Sie unter Hohe Verfügbarkeit für Amazon Aurora.

Sicherheit der Infrastruktur in Amazon Aurora

Als verwalteter Service ist Amazon Relational Database Service durch AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter AWS Cloud-Sicherheit. Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf Amazon RDS zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Darüber hinaus bietet Aurora Funktionen zur Unterstützung der Infrastruktursicherheit.

Sicherheitsgruppen

Sicherheitsgruppen kontrollieren die Zugriffsaktivitäten von eingehendem und ausgehendem Datenverkehr in einem DB-Cluster. Standardmäßig ist der Netzwerkzugriff auf einen DB-Cluster deaktiviert. Sie können Regeln in einer Sicherheitsgruppe angeben, die den Zugriff aus einem IP-Adressbereich, über einen Port oder für eine Sicherheitsgruppe zulassen. Nach der Konfiguration von ingress-Regeln gelten diese für alle DB-Cluster, die dieser Sicherheitsgruppe zugeordnet sind.

Weitere Informationen finden Sie unter Zugriffskontrolle mit Sicherheitsgruppen.

Public accessibility (Öffentliche Zugänglichkeit)

Wenn Sie eine DB-Instance innerhalb einer Virtual Private Cloud (VPC) basierend auf dem Amazon VPC-Service starten, können Sie die öffentliche Zugriffsmöglichkeit für diese DB-Instance ein- oder ausschalten. Um festzulegen, ob die von Ihnen erstellte DB-Instance einen DNS-Namen hat, der in eine öffentliche IP-Adresse aufgelöst wird, verwenden Sie den Parameter Public Accessibility (Öffentliche Erreichbarkeit). Mit diesem Parameter können Sie festlegen, ob ein öffentlicher Zugriff auf die DB-Instance besteht. Sie können eine DB-Instance ändern und die öffentliche Zugänglichkeit im Parameter Öffentlicher Zugriff aktivieren und deaktivieren.

Weitere Informationen finden Sie unter Ausblenden einer DB-Clusters in einer VPC vor dem Internet.

> **Note**
>
> Wenn sich Ihre DB-Instance in einer VPC befindet, aber nicht öffentlich zugänglich ist, können Sie auch eine AWS Site-to-Site VPN-Verbindung oder eine Direct Connect Verbindung verwenden, um von einem privaten Netzwerk aus darauf zuzugreifen. Weitere Informationen finden Sie unter Richtlinie für den Datenverkehr zwischen Netzwerken.

Amazon-RDS-API und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)

Sie können eine private Verbindung zwischen Ihrer VPC und Amazon-RDS-API-Endpunkten herstellen, indem Sie einen Schnittstellen-VPC-Endpunkt erstellen. Schnittstellenendpunkte werden von unterstütz AWS PrivateLink.

AWS PrivateLink ermöglicht Ihnen den privaten Zugriff auf Amazon RDS-API-Operationen ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder Direct Connect Verbindung. DB-Instances in Ihrer VPC benötigen keine öffentlichen IP-Adressen für die Kommunikation mit Amazon-RDS-API-Endpunkten, um DB-Instances und DB-Cluster zu starten, zu ändern oder zu beenden. Ihre DB-Instances benötigen auch keine öffentlichen IP-Adressen, um beliebige der verfügbaren RDS-API-Operationen zu verwenden. Der Datenverkehr zwischen der VPC und Amazon RDS verlässt das Amazon-Netzwerk nicht.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere Elastic Network-Schnittstellen in Ihren Subnetzen dargestellt. Weitere Informationen zu Elastic Network Interfaces finden Sie unter Elastic Network Interfaces im EC2 Amazon-Benutzerhandbuch.

Weitere Informationen zu VPC-Endpunkten finden Sie unter Schnittstellen-VPC-Endpunkte (AWS PrivateLink) im Amazon VPC Benutzerhandbuch. Weitere Informationen zu RDS-API-Operationen finden Sie in der Amazon-RDS-API-Referenz.

Sie benötigen keinen Schnittstellen-VPC-Endpunkt, um eine Verbindung zu einer DB-Cluster herzustellen. Weitere Informationen finden Sie unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC.

Überlegungen zu VPC-Endpunkten

Bevor Sie einen Schnittstellen-VPC-Endpunkt für Amazon RDS-API-Endpunkte einrichten, stellen Sie sicher, dass Sie die Eigenschaften und Einschränkungen des Schnittstellenendpunkts im Amazon VPC Benutzerhandbuch einsehen.

Alle RDS-API-Operationen, die für die Verwaltung von Amazon-Aurora-Ressourcen relevant sind, sind mit AWS PrivateLinküber Ihre VPC verfügbar.

VPC-Endpunktrichtlinien werden für RDS-API-Endpunkte unterstützt. Standardmäßig ist der vollständige Zugriff auf RDS-API-Operationen über den Endpunkt zulässig. Weitere Informationen

finden Sie unter Steuerung des Zugriffs auf Services mit VPC-Endpunkten im Amazon VPC Benutzerhandbuch.

Verfügbarkeit

Die Amazon RDS-API unterstützt derzeit VPC-Endpunkte in den folgenden Regionen: AWS

- US East (Ohio)
- USA Ost (Nord-Virginia)
- USA West (Nordkalifornien)
- USA West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asien-Pazifik (Mumbai)
- Asien-Pazifik (Neuseeland)
- Asien-Pazifik (Osaka)
- Asien-Pazifik (Seoul)
- Asien-Pazifik (Singapur)
- Asien-Pazifik (Sydney)
- Asien-Pazifik (Taipeh)
- Asien-Pazifik (Thailand)
- Asien-Pazifik (Tokio)
- Canada (Central)
- Kanada West (Calgary)
- China (Peking)
- China (Ningxia)
- Europa (Frankfurt)
- Europa (Zürich)
- Europa (Irland)
- Europa (London)
- Europe (Paris)

- Europe (Stockholm)
- Europa (Milan)
- Israel (Tel Aviv)
- Mexiko (Zentral)
- Middle East (Bahrain)
- Südamerika (São Paulo)
- AWS GovCloud (US-Ost)
- AWS GovCloud (US-West)

Erstellen eines Schnittstellen-VPC-Endpunkts für die Amazon RDS-API

Sie können einen VPC-Endpunkt für die Amazon RDS-API entweder mit der Amazon VPC-Konsole oder mit AWS Command Line Interface ()AWS CLI erstellen. Weitere Informationen finden Sie unter Erstellung eines Schnittstellenendpunkts im Amazon VPC Benutzerhandbuch.

Erstellen Sie einen VPC-Endpunkt für die Amazon RDS-API unter Verwendung des Servicenamens `com.amazonaws.`*`region`*`.rds`.

Mit Ausnahme von AWS Regionen in China können Sie, wenn Sie privates DNS für den Endpunkt aktivieren, API-Anfragen an Amazon RDS mit dem VPC-Endpunkt stellen, indem Sie beispielsweise `rds.us-east-1.amazonaws.com` dessen Standard-DNS-Namen für die AWS Region verwenden. Für die AWS Regionen China (Peking) und China (Ningxia) können Sie API-Anfragen mit dem VPC-Endpunkt jeweils mit `rds-api.cn-north-1.amazonaws.com.cn` und `rds-api.cn-northwest-1.amazonaws.com.cn` stellen.

Weitere Informationen finden Sie unter Zugriff auf einen Service über einen Schnittstellenendpunkt im Amazon VPC Benutzerhandbuch.

Erstellen einer VPC-Endpunktrichtlinie für die Amazon RDS-API

Sie können eine Endpunktrichtlinie an Ihren VPC-Endpunkt anhängen, der den Zugriff auf die Amazon RDS-API steuert. Die Richtlinie gibt die folgenden Informationen an:

- Prinzipal, der die Aktionen ausführen kann.
- Aktionen, die ausgeführt werden können
- Die Ressourcen, für die Aktionen ausgeführt werden können.

Weitere Informationen finden Sie unter Steuerung des Zugriffs auf Services mit VPC-Endpunkten im Amazon VPC Benutzerhandbuch.

Beispiel: VPC-Endpunktrichtlinie für Amazon RDS-API-Aktionen

Im Folgenden finden Sie ein Beispiel für eine Endpunktrichtlinie für die Amazon RDS-API. Wenn diese Richtlinie an einen Endpunkt angefügt wird, gewährt sie Zugriff auf die aufgelisteten Amazon RDS-API-Aktionen für alle Prinzipale auf allen Ressourcen.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBInstance",
            "rds:ModifyDBInstance",
            "rds:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

Beispiel: VPC-Endpunktrichtlinie, die jeglichen Zugriff von einem bestimmten Konto aus verweigert AWS

Die folgende VPC-Endpunktrichtlinie verweigert dem AWS Konto `123456789012` jeglichen Zugriff auf Ressourcen, die den Endpunkt verwenden. Die Richtlinie erlaubt alle Aktionen von anderen Konten.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
```

```
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Bewährte Methoden für die Sicherheit in Amazon Aurora

Verwenden Sie AWS Identity and Access Management (IAM)-Konten zur Steuerung des Zugriffs auf API-Operationen VON Amazon RDS, insbesondere Operationen, mit denen Amazon Aurora-Ressourcen erstellt, geändert oder gelöscht werden. Zu solchen Ressourcen gehören DB-Cluster, Sicherheitsgruppen und Parametergruppen. Verwenden Sie zudem IAM zur Steuerung der Aktionen, die allgemeine administrative Aktionen wie die Sicherung und Wiederherstellung von DB-Clustern ausführen.

- Erstellen Sie einen individuellen Benutzer für jede Person, die Ressourcen von Amazon Aurora verwaltet, einschließlich Sie selbst. Verwenden Sie keine AWS-Stamm-Anmeldeinformationen zum Verwalten von Ressourcen von Amazon Aurora.
- Gewähren Sie jedem Benutzer nur den Mindestsatz an Berechtigungen, die für die Ausführung seiner Aufgaben erforderlich sind.
- Verwenden Sie IAM-Gruppen, um Berechtigungen für mehrere Benutzer effektiv zu verwalten.
- Wechseln Sie regelmäßig die IAM-Anmeldeinformationen.
- Konfigurieren Sie AWS Secrets Manager so, dass die Geheimnisse für Amazon Aurora automatisch rotiert werden. Weitere Informationen finden Sie unter Rotieren von AWS Secrets Manager-Secrets im AWS Secrets Manager-Benutzerhandbuch. Sie können die Anmeldeinformationen auch programmgesteuert aus AWS Secrets Manager abrufen. Weitere Informationen finden Sie unter Abrufen des Secret-Wertes im AWS Secrets Manager-Benutzerhandbuch.

Weitere Informationen zur Sicherheit von Amazon Aurora finden Sie unter Sicherheit in Amazon Aurora. Weitere Informationen zu IAM finden Sie unter AWS Identity and Access Management. Informationen zu den bewährten Methoden für IAM finden Sie unter Bewährte Methoden für IAM.

AWS Security Hub CSPM verwendet Sicherheitskontrollen für die Bewertung von Ressourcenkonfigurationen und Sicherheitsstandards, um Sie bei der Einhaltung verschiedener Compliance-Frameworks zu unterstützen. Weitere Informationen zur Verwendung von Security

Hub zur Bewertung von RDS-Ressourcen finden Sie unter Steuerelemente von Amazon Relational Database Service im AWS Security Hub CSPM-Benutzerhandbuch.

Sie können Ihre Nutzung von RDS in Bezug auf bewährte Sicherheitsmethoden mithilfe von Security Hub überwachen. Weitere Informationen finden Sie unter Was ist AWS Security Hub CSPM?.

Verwenden Sie die AWS-Managementkonsole, die AWS CLI oder die RDS-API zum Ändern des Passworts für den Hauptbenutzer. Falls Sie zum Ändern des Hauptbenutzerpassworts ein anderes Tool verwenden, beispielsweise einen SQL-Client, werden dem Benutzer unter Umständen ohne Absicht seine Berechtigungen entzogen.

Amazon GuardDuty ist ein stetiger Sicherheitsüberwachungsservice, der verschiedene Datenquellen, einschließlich Amazon-RDS-Anmeldeaktivitäten, analysiert und verarbeitet. Er verwendet Feeds mit Bedrohungsinformationen und Machine Learning, um unerwartetes und potenziell nicht autorisiertes, verdächtiges Anmeldeverhalten und bösartige Aktivitäten in Ihrer AWS-Umgebung zu identifizieren.

Wenn Amazon GuardDuty RDS Protection eine potenzielle Bedrohung erkennt, generiert GuardDuty eine neue Erkenntnis mit Details zur potenziell kompromittierten Datenbank. Weitere Informationen finden Sie unter Überwachen von Bedrohungen mit Amazon GuardDuty RDS Protection für Amazon Aurora.

Zugriffskontrolle mit Sicherheitsgruppen

VPC-Sicherheitsgruppen kontrollieren die Zugriffsaktivitäten von eingehendem und ausgehendem Datenverkehr in einem DB-Cluster. Standardmäßig ist der Netzwerkzugriff auf einen DB-Cluster deaktiviert. Sie können Regeln in einer Sicherheitsgruppe angeben, die den Zugriff aus einem IP-Adressbereich, über einen Port oder für eine Sicherheitsgruppe zulassen. Nach der Konfiguration von Regeln für eingehenden Datenverkehr gelten diese für alle Cluster, die dieser Sicherheitsgruppe zugeordnet sind. Sie können bis zu 20 Regeln in einer Sicherheitsgruppe angeben.

Überblick über VPC-Sicherheitsgruppen

Jede VPC-Sicherheitsgruppenregel ermöglicht einer bestimmten Quelle den Zugriff auf einen DB-Cluster in einer VPC, die dieser VPC-Sicherheitsgruppe zugeordnet ist. Die Quelle kann ein Adressenbereich (zum Beispiel: 203.0.113.0/24) oder eine andere VPC-Sicherheitsgruppe sein. Wenn Sie eine VPC-Sicherheitsgruppe als Quelle festlegen, erlauben Sie eingehenden Datenverkehr von allen Instances (typischerweise Anwendungsserver), die Quell-VPC-Sicherheitsgruppe verwenden. VPC-Sicherheitsgruppen können über Regeln verfügen, die den eingehenden und ausgehenden Datenverkehr regulieren. Die Regeln für ausgehenden Datenverkehr gelten jedoch

normalerweise nicht für DB-Cluster. Die Regeln für den ausgehenden Datenverkehr gelten nur, wenn der DB-Cluster als Client fungiert. Sie müssen die Amazon EC2-API oder die Option Security Group (Sicherheitsgruppe) in der VPC-Konsole verwenden, um VPC-Sicherheitsgruppen zu erstellen.

Wenn Sie Regeln für Ihre VPC-Sicherheitsgruppe erstellen, die den Zugriff auf Cluster in Ihrer VPC erlauben, müssen Sie einen Port für jeden Adressbereich bestimmen, für den die Regel Zugriff zulässt. Wenn Sie beispielsweise SSH-Zugriff auf Instances in der VPC aktivieren möchten, dann erstellen Sie eine Regel, die Zugriff auf TCP-Port 22 für den bestimmten Adressbereich zulässt.

Sie können mehrere VPC-Sicherheitsgruppen konfigurieren, die Zugriff auf verschiedenen Ports für verschiedenen Instances in Ihrer VPC zulassen. Beispielsweise können Sie eine VPC-Sicherheitsgruppe erstellen, die den Zugriff auf TCP-Port 80 für Webserver in Ihrer VPC ermöglicht. Sie können dann eine andere VPC-Sicherheitsgruppe erstellen, die den Zugriff auf TCP-Port 3306 für DB-Instances von Aurora MySQL in Ihrer VPC ermöglicht.

> **Note**
>
> In einem Aurora-DB-Cluster ist die VPC-Sicherheitsgruppe, die mit dem DB-Cluster verknüpft ist, auch mit allen anderen DB-Instances im DB-Cluster verknüpft. Wenn Sie die VPC-Sicherheitsgruppe für den DB-Cluster oder die DB-Instance ändern, wird die Änderung automatisch für alle DB-Instances im DB-Cluster übernommen.

Weitere Informationen zu VPC-Sicherheitsgruppen finden Sie unter Sicherheitsgruppen im Amazon Virtual Private Cloud-Benutzerhandbuch.

> **Note**
>
> Wenn sich Ihr DB-Cluster in einer VPC befindet, aber nicht öffentlich zugänglich ist, können Sie auch eine AWS-Site-to-Site-VPN-Verbindung oder eine Direct Connect-Verbindung verwenden, um von einem privaten Netzwerk aus darauf zuzugreifen. Weitere Informationen finden Sie unter Richtlinie für den Datenverkehr zwischen Netzwerken.

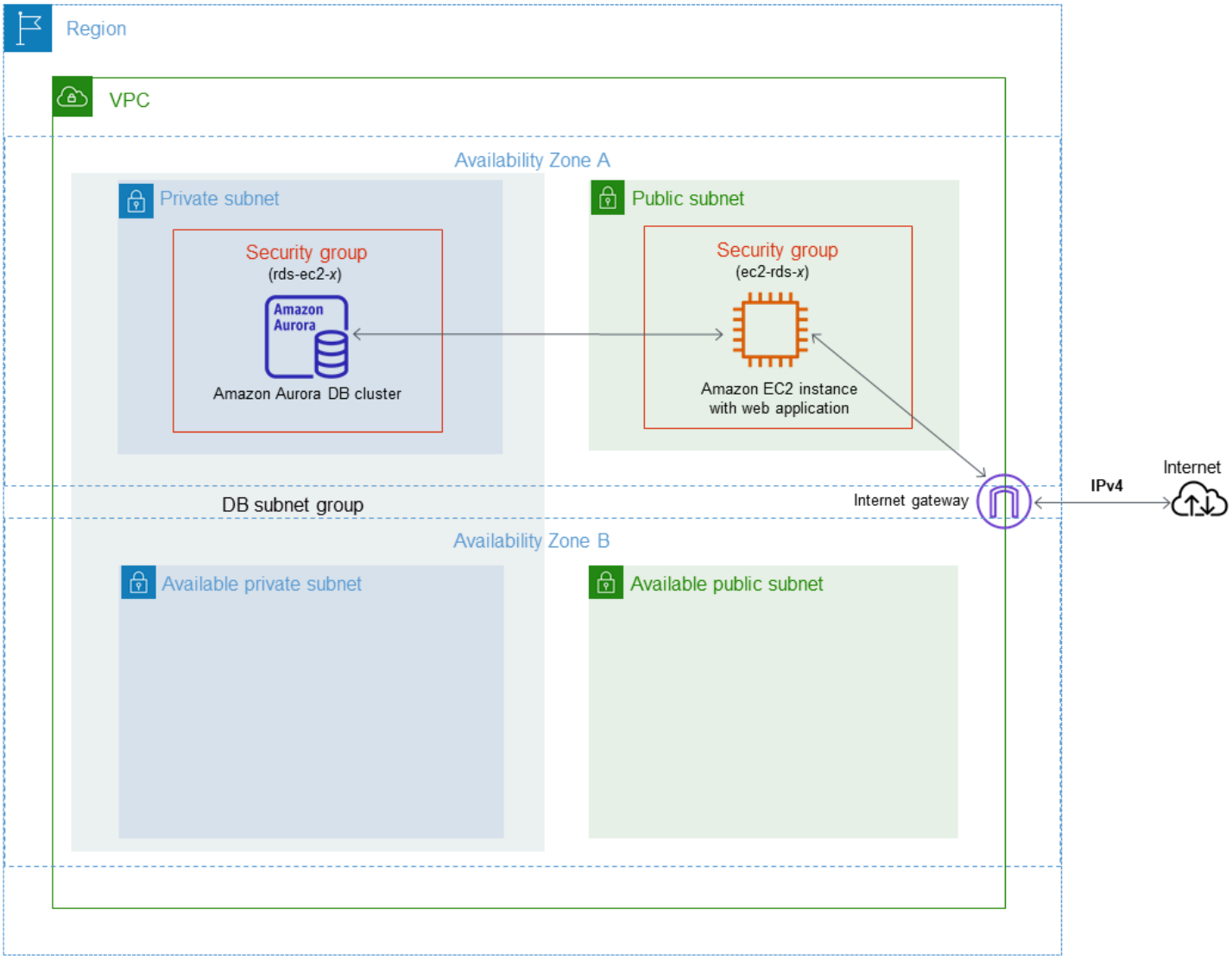
Sicherheitsgruppenszenario

Eine häufige Verwendung von DB-Clustern in einer VPC ist das Teilen von Daten mit einem Anwendungsserver. Dieser wird in einer Amazon-EC2-Instance ausgeführt, die sich in derselben

VPC befindet, auf die eine Clientanwendung von außerhalb der VPC zugreift. Für dieses Szenario verwenden Sie die RDS- und VPC-Seiten der AWS-Managementkonsole oder die RDS- und EC2-API-Operationen, um die notwendigen Instances und Sicherheitsgruppen zu erstellen:

1. Erstellen einer VPC-Sicherheitsgruppe (zum Beispiel `sg-0123ec2example`) und Definieren von eingehenden Regeln, welche die IP-Adressen der Client-Anwendung als Quelle verwenden. Diese Sicherheitsgruppe erlaubt Ihrer Client-Anwendung, sich mit EC2-Instances in einer VPC zu verbinden, die diese Sicherheitsgruppe verwendet.
2. Erstellen Sie eine EC2-Instance für eine Anwendung und fügen Sie dieser EC2-Instance eine VPC-Sicherheitsgruppe (`sg-0123ec2example`) hinzu, die Sie im vorherigen Schritt erstellt haben.
3. Erstellen Sie eine zweite VPC-Sicherheitsgruppe (zum Beispiel `sg-6789rdsexample`) und erstellen Sie eine neue Regel durch Festlegen der VPC-Sicherheitsregel, die Sie in Schritt 1 (`sg-0123ec2example`) als Quelle erstellt haben.
4. Erstellen Sie einen neuen DB-Cluster und fügen Sie die den DB-Cluster der VPC-Sicherheitsgruppe (`sg-6789rdsexample`) hinzu, die Sie im vorherigen Schritt erstellt haben. Wenn Sie einen DB-Cluster erstellen, verwenden Sie dieselbe Portnummer, die für die VPC-Sicherheitsgruppenregel (`sg-6789rdsexample`) festgelegt ist, die Sie in Schritt 3 erstellt haben.

Im folgenden Diagramm wird dieses Szenario veranschaulicht.



Ausführliche Anleitungen zum Konfigurieren einer VPC für dieses Szenario finden Sie unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4). Weitere Informationen zur Verwendung einer VPC finden Sie unter Amazon VPC und Amazon Aurora.

Erstellen einer VPC-Sicherheitsgruppe

Sie können unter Verwendung der VPC-Konsole eine VPC-Sicherheitsgruppe für eine DB-Instance erstellen. Weitere Informationen zum Erstellen einer Sicherheitsgruppe finden Sie unter Ermöglichen des Zugriffs auf den DB-Cluster in der VPC durch Erstellen einer Sicherheitsgruppe und Sicherheitsgruppen im Amazon Virtual Private Cloud-Benutzerhandbuch.

Verknüpfen einer Sicherheitsgruppe mit einem DB-Cluster

Sie können eine Sicherheitsgruppe mit einem DB-Cluster auf der RDS-Konsole über Cluster ändern des Amazon-RDS-API-Aufrufs `ModifyDBCluster` oder des `modify-db-cluster`-Befehls AWS CLI verknüpfen.

Das folgende CLI-Beispiel ordnet eine bestimmte VPC-Gruppe zu und entfernt DB-Sicherheitsgruppen aus der DB-Cluster.

```
aws rds modify-db-cluster --db-cluster-identifier dbName --vpc-security-group-ids sg-ID
```

Weitere Informationen zum Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Berechtigungen von Hauptbenutzerkonten

Wenn Sie neue DB-Cluster erstellen, erhält der Standardhauptbenutzer, den Sie verwenden, bestimmte Berechtigungen für diese DB-Cluster. Sie können den Hauptbenutzernamen nicht ändern, nachdem DB-Cluster erstellt wurden.

> **Important**
>
> Wir empfehlen Ihnen, den Hauptbenutzer nicht direkt in Ihren Anwendungen zu verwenden. Bleiben Sie stattdessen bei der bewährten Methode, einen Datenbankbenutzer zu verwenden, der mit den Mindestberechtigungen erstellt wurde, die für Ihre Anwendung erforderlich sind.

> **Note**
>
> Wenn Sie die Berechtigungen für den Hauptbenutzer aus Versehen gelöscht haben, können Sie diese wiederherstellen, indem Sie die DB-Cluster ändern und ein neues Passwort für den Hauptbenutzer festlegen. Weitere Informationen über das Ändern von DB-Clustern finden Sie unter Ändern eines Amazon Aurora-DB-Clusters .

Die folgende Tabelle zeigt die Sonderrechte und Datenbankrollen, die der Hauptbenutzer in jeder der Datenbank-Engines erhält.

Datenban-Engine	Systemberechtigung	Datenbankrolle
Aurora MySQL	Version 2: `ALTER,ALTER ROUTINE, CREATE,CREATE ROUTINE,CREATE TEMPORARY TABLES,CREATE USER,CREATE VIEW,DELETE,DROP, EVENT,EXECUTE,GRANT OPTION,INDEX,INSERT, LOAD FROM S3,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATION CLIENT , REPLICATION SLAVE ,SELECT, SELECT INTO S3,SHOW DATABASES , SHOW VIEW,TRIGGER, UPDATE`	—
	Version 3: `ALTER,APPLICATION_PASSWORD_ADMIN , ALTER ROUTINE,CONNECTION_ADMIN , CREATE,CREATE ROLE,CREATE ROUTINE,CREATE TEMPORARY TABLES, CREATE USER,CREATE VIEW, DELETE,DROP,DROP ROLE, EVENT,EXECUTE,INDEX, INSERT,LOCK TABLES, PROCESS,REFERENCES , RELOAD,REPLICATION CLIENT , REPLICATION SLAVE ,ROLE_ADMIN , SET_USER_ID ,SELECT,SHOW DATABASES ,SHOW VIEW,TRIGGER, UPDATE,XA_RECOVER_ADMIN`  Ab Aurora MySQL 3.04.0 erhält der Hauptbenutzer auch die Berechtigung `SHOW_ROUTINE` .  Ab Aurora MySQL 3.09.0 erhält der Hauptbenutzer auch die Berechtigungen `FLUSH_OPTIMIZER_COSTS` , `FLUSH_STATUS` , `FLUSH_TABLES` und `FLUSH_USER_RESOURCES` .	`rds_superuser_role` Mehr über rds_superuser_role erfahren Sie unter Rollenbasiertes Berechtigungsmodell.

Datenban -Engine	Systemberechtigung	Datenbankrolle
Aurora PostgreSQ L	`LOGIN,NOSUPERUSER , INHERIT,CREATEDB, CREATEROLE ,NOREPLICATION ,VALID UNTIL 'infinity'`	`RDS_SUPERUSER` Mehr über RDS_SUPERUSER erfahren Sie unter Grundlegendes zu PostgreSQL-Rollen und -Berechtigungen.

Verwenden von serviceverknüpften Rollen für Amazon Aurora

Amazon Aurora nutzt von AWS Identity and Access Management (IAM) serviceverknüpfte Rollen. Eine serviceverknüpfte Rolle ist ein spezieller Typ einer IAM-Rolle, die direkt mit Amazon Aurora verknüpft ist. Serviceverknüpfte Rollen werden von Amazon Aurora vordefiniert und schließen alle Berechtigungen ein, die der Service zum Aufrufen anderer AWS-Services in Ihrem Namen erfordert.

Eine serviceverknüpfte Rolle macht die Nutzung von Amazon Aurora einfacher, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. Amazon Aurora definiert die Berechtigungen seiner serviceverknüpften Rollen, und sofern nicht anders definiert, kann nur Amazon Aurora seine Rollen übernehmen. Die definierten Berechtigungen umfassen die Vertrauens- und Berechtigungsrichtlinie. Diese Berechtigungsrichtlinie kann keinen anderen IAM-Entitäten zugewiesen werden.

Sie können die Rollen nur nach dem Löschen der zugehörigen Ressourcen löschen. Dies schützt Ihre Amazon Aurora-Ressourcen, da Sie nicht versehentlich die Berechtigung für den Zugriff auf die Ressourcen entfernen können.

Informationen zu anderen Services, die serviceverknüpfte Rollen unterstützen, finden Sie unter AWS-Services, die mit IAM funktionieren. Suchen Sie nach den Services, für die Yes (Ja) in der Spalte Service-Linked Role (Serviceverknüpfte Rolle) angegeben ist. Wählen Sie über einen Link Ja aus, um die Dokumentation zu einer serviceverknüpften Rolle für diesen Service anzuzeigen.

Berechtigungen von serviceverknüpften Rollen für Amazon Aurora

Amazon Aurora verwendet die serviceverknüpfte Rolle namens AWSServiceRoleForRDS– damit Amazon RDS -AWSServices im Namen Ihrer DB-clusters aufrufen kann.

Die servicegebundene Rolle AWSServiceRoleForRDS vertraut den folgenden Services, die diese Rolle übernehmen:

- `rds.amazonaws.com`

Dieser dienstgebundenen Rolle ist eine Berechtigungsrichtlinie namens `AmazonRDSServiceRolePolicy` zugeordnet, die ihr Berechtigungen für den Betrieb in Ihrem Konto erteilt.

Weitere Informationen zu dieser Richtlinie, einschließlich des JSON-Richtliniendokuments, finden Sie unter AmazonRDSServiceRolePolicy im Referenzleitfaden zur AWS-verwalteten Richtlinie.

> **Note**
>
> Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Wenn Sie die folgende Fehlermeldung erhalten:
> Unable to create the resource. Überprüfen Sie, ob Sie die Berechtigung haben, eine serviceverknüpfte Rolle zu erstellen. Andernfalls warten Sie und versuchen Sie es später noch einmal.
> Stellen Sie sicher, dass Sie die folgenden Berechtigungen für Sie aktiviert sind:
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> Weitere Informationen finden Sie unter Serviceverknüpfte Rollenberechtigung im IAM-Benutzerhandbuch.

Erstellen einer serviceverknüpften Rolle für Amazon Aurora

Sie müssen eine serviceverknüpfte Rolle nicht manuell erstellen. Wenn Sie einen DB-Cluster erstellen, erstellt Amazon Aurora die serviceverknüpfte Rolle für Sie.

> **Important**
>
> Wenn Sie Amazon Aurora bereits vor dem 1. Dezember 2017 verwendet haben, bevor der Service serviceverknüpfte Rollen unterstützt hat, hat Amazon Aurora die Rolle AWSServiceRoleForRDS in Ihrem Konto erstellt. Weitere Informationen finden Sie unter In meinem AWS-Konto wird eine neue Rolle angezeigt.

Wenn Sie diese serviceverknüpfte Rolle löschen und sie dann erneut erstellen müssen, können Sie dasselbe Verfahren anwenden, um die Rolle in Ihrem Konto neu anzulegen. Wenn Sie einen DB-Cluster erstellen, erstellt Amazon Aurora wieder die serviceverknüpfte Rolle für Sie.

Bearbeiten einer serviceverknüpften Rolle für Amazon Aurora

Amazon Aurora erlaubt es Ihnen nicht, die serviceverknüpfte Rolle AWSServiceRoleForRDS zu bearbeiten. Da möglicherweise verschiedene Entitäten auf die Rolle verweisen, kann der Rollenname nach dem Erstellen einer serviceverknüpften Rolle nicht mehr geändert werden. Sie können jedoch die Beschreibung der Rolle mit IAM bearbeiten. Weitere Informationen finden Sie unter Bearbeiten einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Löschen einer serviceverknüpften Rolle für Amazon Aurora

Wenn Sie eine Funktion oder einen Service, die bzw. der eine serviceverknüpfte Rolle erfordert, nicht mehr benötigen, sollten Sie diese Rolle löschen. Auf diese Weise haben Sie keine ungenutzte Entität, die nicht aktiv überwacht oder verwaltet wird. Sie müssen jedoch all Ihre DB-Cluster löschen, bevor Sie die serviceverknüpfte Rolle löschen können.

Bereinigen einer serviceverknüpften Rolle

Bevor Sie mit IAM eine serviceverknüpfte Rolle löschen können, müssen Sie sich zunächst vergewissern, dass die Rolle über keine aktiven Sitzungen verfügt, und alle Ressourcen entfernen, die von der Rolle verwendet werden.

So überprüfen Sie in der IAM-Konsole, ob die serviceverknüpfte Rolle über eine aktive Sitzung verfügt

1. Melden Sie sich bei der AWS-Managementkonsole an, und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich der IAM Console Roles aus. Wählen Sie dann den Namen (nicht das Kontrollkästchen) der Rolle AWSServiceRoleForRDS aus.
3. Wählen Sie auf der Seite Zusammenfassung für die ausgewählte Rolle die Registerkarte Zuletzt aufgerufen aus.
4. Überprüfen Sie auf der Registerkarte Zuletzt aufgerufen die jüngsten Aktivitäten für die serviceverknüpfte Rolle.

> **Note**
>
> Wenn Sie sich nicht sicher sind, ob Amazon Aurora die Rolle AWSServiceRoleForRDS verwendet, können Sie versuchen, die Rolle zu löschen. Wenn der Service die Rolle verwendet, schlägt die Löschung fehl und Sie können die AWS-Regionen anzeigen, in denen die Rolle verwendet wird. Wenn die Rolle verwendet wird, müssen Sie warten, bis die Sitzung beendet wird, bevor Sie die Rolle löschen können. Die Sitzung für eine serviceverknüpfte Rolle können Sie nicht widerrufen.

Wenn Sie die Rolle AWSServiceRoleForRDS entfernen wollen, müssen Sie zunächst alle DB-Cluster löschen.

Löschen aller Ihrer Cluster

Verwenden Sie eines der folgenden Verfahren, um einen einzelnen Cluster zu löschen. Wiederholen Sie dieses Verfahren für jeden der Cluster.

So löschen Sie einen Cluster (Konsole)

1. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der Liste Databases (Datenbanken) den Cluster aus, den Sie löschen möchten.
3. Wählen Sie bei Cluster Actions (Cluster-Aktionen) die Option Delete (Löschen).
4. Wählen Sie Delete (Löschen).

So löschen Sie einen Cluster (CLI)

Siehe `delete-db-cluster` in der AWS CLI-Befehlsreferenz.

So löschen Sie einen Cluster (API)

Weitere Informationen finden Sie im Amazon RDS API Reference unter `DeleteDBCluster`.

Sie können die IAM-Konsole, die IAM-CLI oder die IAM-API verwenden, um die serviceverknüpfte Rolle AWSServiceRoleForRDS zu löschen. Weitere Informationen finden Sie unter Löschen einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Serviceverknüpfte Rollenberechtigungen für Amazon RDS Beta

Amazon Aurora verwendet die serviceverknüpfte Rolle namens `AWSServiceRoleForRDSBeta`, damit Amazon Aurora AWS-Services im Namen Ihrer RDS-DB-Ressourcen aufrufen kann.

Die servicegebundene Rolle AWSServiceRoleForRDSBeta vertraut darauf, dass folgende Services diese Rolle übernehmen können:

- `rds.amazonaws.com`

Dieser dienstgebundenen Rolle ist eine Berechtigungsrichtlinie namens `AmazonRDSBetaServiceRolePolicy` zugeordnet, die ihr Berechtigungen für den Betrieb in Ihrem Konto erteilt. Weitere Informationen finden Sie unter AWS-verwaltete Richtlinie: AmazonRDSBetaServiceRolePolicy.

> **Note**
>
> Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Wenn Sie die folgende Fehlermeldung erhalten:
> Unable to create the resource. Überprüfen Sie, ob Sie die Berechtigung haben, eine serviceverknüpfte Rolle zu erstellen. Andernfalls warten Sie und versuchen Sie es später noch einmal.
> Stellen Sie sicher, dass Sie die folgenden Berechtigungen für Sie aktiviert sind:

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
AmazonRDSBetaServiceRolePolicy",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"custom.rds.amazonaws.com"
        }
    }
}
```

Weitere Informationen finden Sie unter Serviceverknüpfte Rollenberechtigung im IAM-Benutzerhandbuch.

Serviceverknüpfte Rolle für Amazon RDS Preview

Amazon Aurora verwendet die serviceverknüpfte Rolle namens `AWSServiceRoleForRDSPreview`, damit Amazon Aurora AWS-Services im Namen Ihrer RDS-DB-Ressourcen aufrufen kann.

Die serviceverknüpfte Rolle AWSServiceRoleForRDSPreview vertraut darauf, dass die folgenden Services diese Rolle übernehmen können:

- `rds.amazonaws.com`

Dieser dienstgebundenen Rolle ist eine Berechtigungsrichtlinie namens `AmazonRDSPreviewServiceRolePolicy` zugeordnet, die ihr Berechtigungen für den Betrieb in Ihrem Konto erteilt. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinie: AmazonRDSPreviewServiceRolePolicy.

> **Note**
>
> Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Wenn Sie die folgende Fehlermeldung erhalten:
> Unable to create the resource. Überprüfen Sie, ob Sie die Berechtigung haben, eine serviceverknüpfte Rolle zu erstellen. Andernfalls warten Sie und versuchen Sie es später noch einmal.
> Stellen Sie sicher, dass Sie die folgenden Berechtigungen für Sie aktiviert sind:

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
AmazonRDSPreviewServiceRolePolicy",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"custom.rds.amazonaws.com"
        }
    }
```

```
}
```

Weitere Informationen finden Sie unter Serviceverknüpfte Rollenberechtigung im IAM-Benutzerhandbuch.

Amazon VPC und Amazon Aurora

Mit Amazon Virtual Private Cloud (Amazon VPC) können Sie AWS-Ressourcen, wie z. B. Aurora-DB-Cluster, in einer Virtual Private Cloud (VPC) starten.

Wenn Sie eine VPC verwenden, haben Sie die Kontrolle über Ihre virtuelle Netzwerkumgebung. Sie können Ihren eigenen IP-Adressbereich auswählen, Subnetze erstellen sowie Routing-Tabellen und Zugriffskontrolllisten konfigurieren. Es fallen keine zusätzlichen Kosten für das Ausführen einer DB-Cluster in der Amazon VPC an.

Konten haben eine Standard-VPC. Alle neuen DB-Cluster werden in der Standard-VPC erstellt, außer Sie ändern die Einstellungen.

Themen

- Arbeiten mit einer DB-Cluster in einer VPC
- Szenarien für den Zugriff auf eine DB-Cluster in einer VPC
- Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4)
- Tutorial: Erstellen einer VPC zur Verwendung mit einer DB–einem DB-Cluster (Dual-Stack-Modus)

Im Folgenden finden Sie eine Diskussion über VPC-Funktionalität, die relevant ist für Amazon Aurora-DB-Cluster. Weitere Informationen zu Amazon VPC finden Sie unter Amazon-VPC-Handbuch „Erste Schritte“ und Amazon-VPC-Benutzerhandbuch.

Arbeiten mit einer DB-Cluster in einer VPC

Ihr DB-Cluster befindet sich in einer Virtual Private Cloud (VPC). Eine VPC ist ein virtuelles Netzwerk, das logisch von anderen virtuellen Netzwerken in der AWS Cloud isoliert ist. Amazon VPC ermöglicht es Ihnen, AWS Ressourcen wie einen Aurora in einer VPC zu starten. EC2 Bei der VPC kann es sich um die mit Ihrem Konto verknüpfte Standard-VPC oder eine von Ihnen erstellte VPC handeln. Alle VPCs sind mit Ihrem AWS Konto verknüpft.

Die Standard-VPC besitzt drei Subnetze, mit denen Sie Ressourcen innerhalb der VPC separieren können. Zudem verfügt die Standard-VPC über ein Internet-Gateway, mit dem Sie den externen Zugriff auf Ressourcen in der VPC gewähren können.

Eine Liste von Szenarien mit Amazon Aurora DB-Clusters in einer VPC finden Sie unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC.

Themen

- Arbeiten mit einer DB-Cluster in einer VPC
- VPC-Verschlüsselungssteuerung
- Arbeiten mit DB-Subnetzgruppen
- Gemeinsam genutzte Subnetze
- Amazon-Aurora-IP-Adressierung
- Ausblenden einer DB-Clusters in einer VPC vor dem Internet
- Erstellen einer DB-Cluster in einer VPC

In den folgenden Tutorials lernen Sie, eine VPC zu erstellen, die Sie für ein gängiges Amazon-Aurora-Szenario verwenden können:

- Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4)
- Tutorial: Erstellen einer VPC zur Verwendung mit einer DB–einem DB-Cluster (Dual-Stack-Modus)

Arbeiten mit einer DB-Cluster in einer VPC

Hier sind einige Tipps für das Arbeiten mit einer DB-Cluster in einer VPC:

- Ihre VPC muss mindestens zwei Subnetze besitzen. Diese Subnetze müssen sich in zwei verschiedenen Availability Zones in dem Bereich befinden AWS-Region , in dem Sie Ihren bereitstellen möchten. Ein Subnetz ist ein Segment des IP-Adressbereichs einer VPC, das Sie angeben können und das Sie zur Gruppierung von DB-Clustern auf der Grundlage Ihrer Sicherheits- und Betriebsanforderungen verwenden können.
- Wenn Sie möchten, dass Ihr DB-Cluster in der VPC öffentlich zugänglich ist, stellen Sie sicher, dass Sie die VPC-Attribute DNS-Hostnamen und DNS-Auflösung aktivieren.
- Sie müssen für Ihre VPC eine DB-Sicherheitsgruppe erstellen. Sie erstellen eine DB-Subnetzgruppe, indem Sie die Subnetze angeben, die Sie erstellt haben. Amazon Aurora wählt ein Subnetz und eine IP-Adresse innerhalb dieses Subnetzes, die mit der primären DB-Instance in Ihrem DB-Cluster verknüpft wird. Die primäre DB-Instance verwendet die Availability Zone, die das Subnetz enthält.
- Ihre VPC muss über eine VPC-Sicherheitsgruppe verfügen, die den Zugriff auf die DB-Cluster zulässt.

  Weitere Informationen finden Sie unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC.

- Die CIDR-Blöcke in jedem Subnetz müssen groß genug sein, um freie IP-Adressen für Amazon Aurora unterzubringen, die während der Wartungsarbeiten genutzt werden können, einschließlich Failover und Skalierung. Beispielsweise ist ein Bereich wie 10.0.0.0/24 und 10.0.1.0/24 normalerweise groß genug.
- Eine VPC kann über das Attribut instance tenancy mit dem Wert default oder dedicated verfügen. Bei allen Standardeinstellungen VPCs ist das Instance-Tenancy-Attribut auf Standard gesetzt, und eine Standard-VPC kann jede DB-Instance-Klasse unterstützen.

  Wenn Sie sich dafür entscheiden, Ihren in einer dedizierten VPC zu haben, in der das Instance-Tenancy-Attribut auf Dedicated gesetzt ist, muss die DB-Instance-Klasse Ihres einer der zugelassenen Amazon EC2 Dedicated Instance-Typen sein. Die EC2 dedizierte r5.large-Instance entspricht beispielsweise der DB-Instance-Klasse db.r5.large. Informationen über die Instance-Tenancy in einer VPC finden Sie unter Dedicated Instances im Amazon Elastic Compute Cloud-Benutzerhandbuch.

  Weitere Informationen zu den Instance-Typen, die sich in einer Dedicated Instance befinden können, finden Sie auf der EC2 Amazon-Preisseite unter Amazon EC2 Dedicated Instances.

  > **Note**
  >
  > Wenn Sie das Attribut instance tenancy für einen DB-Cluster auf „dedicated“ setzen, garantiert dies nicht, dass der DB-Cluster auf einem dedizierten Host läuft.

VPC-Verschlüsselungssteuerung

Mit VPC-Verschlüsselungskontrollen können Sie den gesamten Netzwerkverkehr in Ihrem VPCs System durchsetzen encryption-in-transit. Verwenden Sie die Verschlüsselungskontrolle, um die Einhaltung gesetzlicher Vorschriften zu erfüllen, indem Sie sicherstellen, dass nur verschlüsselungsfähige NITRO-basierte Hardware an den dafür vorgesehenen Stellen bereitgestellt werden kann. VPCs Durch die Verschlüsselungskontrolle werden Kompatibilitätsprobleme auch bei der API-Anfrage und nicht bei der Bereitstellung erkannt. Ihre vorhandenen Workloads werden weiterhin ausgeführt und nur neue inkompatible Anfragen werden blockiert.

Richten Sie Ihre VPC-Verschlüsselungskontrollen ein, indem Sie den VPC-Steuerungsmodus auf Folgendes einstellen:

- deaktiviert (Standard)

- überwachen
- erzwungen

Um den aktuellen Steuermodus für Ihre VPC zu überprüfen, verwenden Sie den AWS-Managementkonsole oder DescribeVpcsCLI- oder API-Befehl.

Wenn Ihre VPC die Verschlüsselung erzwingt, können Sie in dieser VPC nur Nitro-basierte bereitstellen, die Verschlüsselung bei der Übertragung unterstützen. Weitere Informationen finden Sie unter DB-Instance-Klassenarten. Informationen zu Nitro-Instances finden Sie im EC2 Amazon-Benutzerhandbuch unter Instances, die auf dem AWS Nitro-System basieren.

> **Note**
>
> Wenn Sie versuchen, inkompatible in einer durch Verschlüsselung erzwungenen VPC bereitzustellen, gibt Aurora eine Ausnahme zurück. `VpcEncryptionControlViolationException`

Aurora ServerlessFür MySQL und PostreSQL erfordert die Verschlüsselungskontrolle Plattformversion 3 oder höher.

Arbeiten mit DB-Subnetzgruppen

Subnetze sind Segmente eines IP-Adressbereichs der VPC, die Sie festlegen und mit denen Sie basierend auf Ihren Sicherheits- und Betriebsanforderungen Ressourcen gruppieren können. Eine DB-Subnetzgruppe ist eine Sammlung von Subnetzen (in der Regel private Subnetze), die Sie in einer VPC erstellen und anschließend Ihrer DB-Clusters zuweisen. Mithilfe einer DB-Subnetzgruppe können Sie eine bestimmte VPC angeben, wenn Sie mithilfe der AWS CLI oder RDS-API erstellen. Wenn Sie die Konsole verwenden, können Sie die VPC und Subnetze auswählen, die Sie verwenden möchten.

Jede DB-Subnetzgruppe sollte über Subnetze in mindestens zwei Availability Zones in einer bestimmten AWS-Region verfügen. Beim Erstellen einer DB-Cluster in einer VPC müssen Sie eine DB-Subnetzgruppe dafür auswählen. Aus der DB-Subnetzgruppe wählt Amazon Aurora ein Subnetz und eine IP-Adresse innerhalb dieses Subnetzes aus, um es mit der primären DB-Instance in Ihrem DB-Cluster zu verbinden. Die DB verwendet die Availability Zone, die das Subnetz enthält. Aurora weist immer eine IP-Adresse aus einem Subnetz zu, das über freien IP-Adressraum verfügt.

Die Subnetze in einer DB-Subnetzgruppe sind entweder öffentlich oder privat. Die Subnetze sind öffentlich oder privat, abhängig von der Konfiguration, die Sie für ihre Netzwerk-Zugriffskontrolllisten (Netzwerk ACLs) und Routing-Tabellen festgelegt haben. Damit öffentlich auf eine DB-Cluster zugegriffen werden kann, müssen alle Subnetze in der entsprechenden DB-Subnetzgruppe öffentlich sein. Wenn ein Subnetz, das mit einer öffentlich zugänglichen DBCluster verknüpft ist, von öffentlich in privat geändert wird, kann dies die Verfügbarkeit der DB-Cluster beeinträchtigen.

Um eine DB-Subnetzgruppe zu erstellen, die den Dual-Stack-Modus unterstützt, stellen Sie sicher, dass jedem Subnetz, das Sie der DB-Subnetzgruppe hinzufügen, ein CIDR-Block der Internetprotokollversion 6 (IPv6) zugeordnet ist. Weitere Informationen finden Sie unter Amazon-Aurora-IP-Adressierung und Migration zu IPv6 im Amazon VPC-Benutzerhandbuch.

Wenn Amazon Aurora eine DB-Cluster in einer VPC erstellt, wird Ihrer DB-Cluster mithilfe einer IP-Adresse aus Ihrer DB-Subnetzgruppe eine Netzwerkschnittstelle zugewiesen. Wir empfehlen jedoch dringend, den DNS-Namen (Domain Name System) für die Verbindung zu Ihrem DB-Cluster zu verwenden. Wir empfehlen dies, da sich die zugrunde liegende IP-Adresse während des Failovers ändert.

> **Note**
>
> Für jede DB-Cluster, die Sie in einer VPC ausführen, stellen Sie sicher, mindestens eine Adresse in jedem Subnetz der DB-Subnetzgruppe für Wiederherstellungsmaßnahmen von Amazon Aurora zu reservieren.

Gemeinsam genutzte Subnetze

Sie können eine(n) DB-Cluster in einer gemeinsam genutzten VPC erstellen.

Einige Überlegungen, die Sie bei der Verwendung von Shared beachten sollten: VPCs

- Sie können eine(n) DB-Cluster von einem gemeinsam genutzten VPC-Subnetz in ein nicht gemeinsam genutztes VPC-Subnetz verschieben und umgekehrt.
- Teilnehmer in einer gemeinsam genutzten VPC müssen eine Sicherheitsgruppe in der VPC erstellen, damit sie eine(n) DB-Cluster erstellen können.
- Besitzer und Teilnehmer in einer gemeinsam genutzten VPC können mithilfe von SQL-Abfragen auf die Datenbank zugreifen. Allerdings kann nur der Ersteller einer Ressource beliebige API-Aufrufe für die Ressource tätigen.

Amazon-Aurora-IP-Adressierung

IP-Adressen ermöglichen es Ressourcen in Ihrer VPC untereinander und mit Ressourcen im Internet zu kommunizieren. Amazon Aurora unterstützt beide IPv4 und IPv6 Adressierungsprotokolle. Standardmäßig verwenden , Amazon Aurora und Amazon VPC das IPv4 Adressierungsprotokoll. Sie können dieses Standardverhalten nicht deaktivieren. Achten Sie beim Erstellen einer VPC darauf, einen IPv4 CIDR-Block (einen Bereich von privaten IPv4 Adressen) anzugeben. Sie können Ihrer VPC und Ihren Subnetzen optional einen IPv6 CIDR-Block zuweisen und IPv6 Adressen aus diesem Block in Ihrem Subnetz zuweisen.

Die Support des IPv6 Protokolls erweitert die Anzahl der unterstützten IP-Adressen. Durch die Verwendung des IPv6 Protokolls stellen Sie sicher, dass Sie über ausreichend verfügbare Adressen für das future Wachstum des Internets verfügen. Neue und bestehende RDS-Ressourcen können verwendet IPv4 und innerhalb Ihrer VPC IPv6 adressiert werden. Das Konfigurieren, Sichern und Übersetzen des Netzwerkverkehrs zwischen den beiden Protokollen, die in verschiedenen Teilen einer Anwendung verwendet werden, können den Betriebsaufwand erhöhen. Sie können das IPv6 Protokoll für Amazon RDS-Ressourcen standardisieren, um Ihre Netzwerkkonfiguration zu vereinfachen.

Themen

- IPv4 Adressen
- IPv6 Adressen
- Dual-Stack-Modus

IPv4 Adressen

Wenn Sie eine VPC erstellen, müssen Sie einen IPv4 Adressbereich für die VPC in Form eines CIDR-Blocks angeben, z. B. `10.0.0.0/16` Eine DB-Subnetzgruppe definiert den Bereich der IP-Adressen in diesem CIDR-Block, den eine DB-Cluster verwenden kann. Diese IP-Adressen können privat oder öffentlich sein.

Eine private IPv4 Adresse ist eine IP-Adresse, die über das Internet nicht erreichbar ist. Sie können private IPv4 Adressen für die Kommunikation zwischen Ihrem und anderen Ressourcen wie EC2 Amazon-Instances in derselben VPC verwenden. Jeder DB-Cluster hat eine private IP-Adresse für die Kommunikation in der VPC.

Eine öffentliche IP-Adresse ist eine IPv4 Adresse, die über das Internet erreichbar ist. Sie können öffentliche Adressen zur Kommunikation zwischen Ihrem DB-Cluster und Ressourcen im Internet, wie ein SQL-Client, verwenden. Anhand der folgenden Schritte können Sie kontrollieren, ob Ihr DB-Cluster eine öffentliche IP-Adresse erhält:

Ein Tutorial, das Ihnen zeigt, wie Sie eine VPC mit nur privaten IPv4 Adressen erstellen, die Sie für ein gängiges Aurora Aurora-Szenario verwenden können, finden Sie unterTutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).

IPv6 Adressen

Sie können Ihrer VPC und Ihren Subnetzen optional einen IPv6 CIDR-Block zuordnen und den Ressourcen in Ihrer VPC IPv6 Adressen aus diesem Block zuweisen. Jede IPv6 Adresse ist weltweit einzigartig.

Der IPv6 CIDR-Block für Ihre VPC wird automatisch aus dem Adresspool von IPv6 Amazon zugewiesen. Sie können den Bereich nicht selbst auswählen.

Wenn Sie eine Verbindung zu einer IPv6 Adresse herstellen, stellen Sie sicher, dass die folgenden Bedingungen erfüllt sind:

- Der Client ist so konfiguriert, dass der Datenverkehr zwischen Client und Datenbank zulässig IPv6 ist.
- Die von der DB-Instance verwendeten RDS-Sicherheitsgruppen sind korrekt konfiguriert, sodass der Datenverkehr vom Client zur Datenbank zulässig IPv6 ist.
- Der Client-Betriebssystemstapel ermöglicht den Datenverkehr über die IPv6 Adresse, und die Betriebssystemtreiber und Bibliotheken sind so konfiguriert, dass sie den richtigen Standard-DB-Instance-Endpunkt (entweder IPv4 oder IPv6) auswählen.

Weitere Informationen finden Sie IPv6 unter IP-Adressierung im Amazon VPC-Benutzerhandbuch.

Dual-Stack-Modus

Wenn ein sowohl über das als auch über das IPv4 IPv6 Adressierungsprotokoll kommunizieren kann, wird er im Dual-Stack-Modus ausgeführt. Ressourcen können also über, oder beides IPv4, IPv6 mit dem kommunizieren. RDS deaktiviert den Internet-Gateway-Zugriff für IPv6 Endpunkte von privaten DB-Instances im Dual-Stack-Modus. RDS tut dies, um sicherzustellen, dass Ihre IPv6 Endpunkte privat sind und nur von Ihrer VPC aus darauf zugegriffen werden kann.

Themen

- Dual-Stack-Modus und DB-Subnetzgruppen
- Arbeiten mit DB-Instances im Dual-Stack-Modus
- Ändern von IPv4 reinen zur Verwendung des Dual-Stack-Modus
- Verfügbarkeit von Dual-Stack-Netzwerk-DB-Cluster
- Einschränkungen für Dual-Stack-Netzwerk-DB-Cluster

Ein Tutorial, das Ihnen zeigt, wie Sie eine VPC mit beiden IPv4 IPv6 Adressen erstellen, die Sie für ein gängiges Aurora Aurora-Szenario verwenden können, finden Sie unterTutorial: Erstellen einer VPC zur Verwendung mit einer DB–einem DB-Cluster (Dual-Stack-Modus).

Dual-Stack-Modus und DB-Subnetzgruppen

Um den Dual-Stack-Modus zu verwenden, stellen Sie sicher, dass jedem Subnetz in der DB-Subnetzgruppe, das Sie dem zuordnen, ein IPv6 CIDR-Block zugeordnet ist. Sie können eine neue DB-Subnetzgruppe erstellen oder eine vorhandene DB-Subnetzgruppe ändern, um diese Anforderung zu erfüllen. Nachdem ein DB-Cluster in den Dual-Stack-Modus gewechselt ist, können sich Clients normal damit verbinden. Stellen Sie sicher, dass die Client-Sicherheits-Firewalls und die Sicherheitsgruppen der RDS-DB-Instance korrekt konfiguriert sind, um den Datenverkehr zu ermöglichen. IPv6 Zum Herstellen einer Verbindung verwenden Clients den Endpunkt der primären Instance des DB-Clusters. Clientanwendungen können angeben, welches Protokoll bevorzugt wird, wenn eine Verbindung mit einer Datenbank hergestellt wird. Im Dual-Stack-Modus erkennt der das vom Client bevorzugte Netzwerkprotokoll (entweder IPv4 oder IPv6) und verwendet dieses Protokoll für die Verbindung.

Wenn eine DB-Subnetzgruppe den Dual-Stack-Modus aufgrund der Löschung des Subnetzes oder der CIDR-Trennung nicht mehr unterstützt, besteht die Gefahr eines inkompatiblen Netzwerkstatus für DB-Instances, die mit der DB-Subnetzgruppe verknüpft sind. Sie können die DB-Subnetzgruppe auch nicht verwenden, wenn Sie einen neuen DB-Cluster im Dual-Stack-Modus erstellen.

Um mithilfe von festzustellen, ob eine DB-Subnetzgruppe den Dual-Stack-Modus unterstützt AWS-Managementkonsole, sehen Sie sich den Netzwerktyp auf der Detailseite der DB-Subnetzgruppe an. Um mithilfe von zu ermitteln, ob eine DB-Subnetzgruppe den Dual-Stack-Modus unterstützt AWS CLI, führen Sie den describe-db-subnet-groupsBefehl aus und sehen Sie sich die Ausgabe an. `SupportedNetworkTypes`

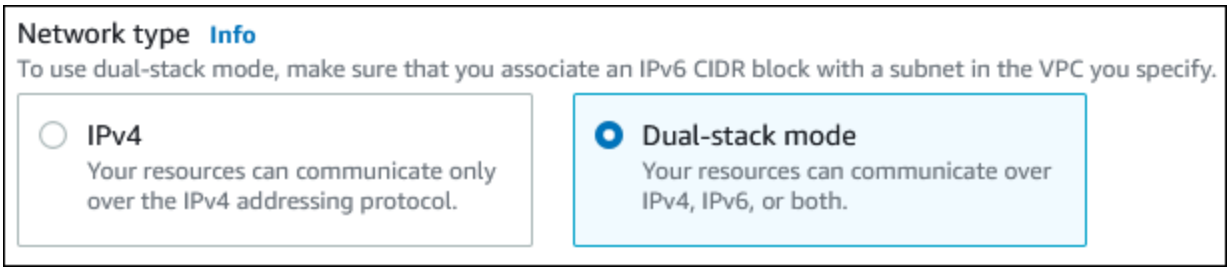
Lesereplikate werden als unabhängige DB-Instances behandelt und können einen anderen Netzwerktyp haben als die primäre DB-Instance. Wenn Sie den Netzwerktyp der primären DB-

Instance eines Lesereplikats ändern, ist das Lesereplikat nicht betroffen. Wenn Sie eine DB-Instance wiederherstellen, können Sie sie auf jedem unterstützten Netzwerktyp wiederherstellen.

Arbeiten mit DB-Instances im Dual-Stack-Modus

Wenn Sie einen erstellen oder ändern, können Sie den Dual-Stack-Modus angeben, damit Ihre Ressourcen mit Ihrem über IPv4 oder beides kommunizieren können. IPv6

Wenn Sie den verwenden AWS-Managementkonsole , um eine DB-Instance zu erstellen oder zu ändern, können Sie den Dual-Stack-Modus im Abschnitt Netzwerktyp angeben. Die folgende Abbildung zeigt den Abschnitt Network type (Netzwerktyp) in der Konsole.



Wenn Sie den verwenden AWS CLI , um einen zu erstellen oder zu ändern, stellen Sie die `--network-type` Option `DUAL` auf Dual-Stack-Modus ein. Wenn Sie die RDS API zum Erstellen oder Ändern eines DB-Clusters verwenden, legen Sie den Parameter `NetworkType` auf `DUAL` fest, um den Dual-Stack-Modus zu verwenden. Wenn Sie den Netzwerktyp einer DB-Instance ändern, sind Ausfallzeiten möglich. Wenn der Dual-Stack-Modus von der angegebenen DB-Engine-Version oder der DB-Subnetzgruppe nicht unterstützt wird, wird der Fehler `NetworkTypeNotSupported` zurückgegeben.

Weitere Informationen zum Erstellen eines DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters. Weitere Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

Um festzustellen, ob sich ein DB-Cluster im Dual-Stack-Modus befindet, sehen Sie sich den Netzwerktyp auf der Registerkarte Connectivity & security (Konnektivität & Sicherheit) für den DB-Cluster an.

Ändern von IPv4 reinen zur Verwendung des Dual-Stack-Modus

Sie können einen verwendet. Dazu ändern Sie den Netzwerktyp der DB-Cluster. Die Änderung kann zu Ausfallzeiten führen.

Es wird empfohlen, den Netzwerktyp Ihrer Amazon-Aurora-DB-Cluster während eines Wartungsfensters zu ändern. Das Festlegen des Netzwerktyps neuer Instances auf den Dual-Stack-

Modus wird derzeit nicht unterstützt. Sie können den Netzwerktyp manuell festlegen, indem Sie den Befehl `modify-db-cluster` verwenden.

Bevor Sie eine DB-Cluster zur Verwendung des Dual-Stack-Modus ändern, stellen Sie sicher, dass ihre DB-Subnetzgruppe den Dual-Stack-Modus unterstützt. Wenn die mit der DB-Cluster verknüpfte DB-Subnetzgruppe den Dual-Stack-Modus nicht unterstützt, geben Sie eine andere DB-Subnetzgruppe an, die DB-Cluster unterstützt, wenn Sie sie ändern. Das Ändern der DB-Subnetzgruppe eines DB-Clusters kann zu Ausfallzeiten führen.

Wenn Sie die DB-Subnetzgruppe eines DB-Clusters ändern, bevor Sie den DB-Cluster zur Verwendung des Dual-Stack-Modus ändern, stellen Sie sicher, dass die DB-Subnetzgruppe vor und nach der Änderung für den DB-Cluster gültig ist.

Wir empfehlen, dass Sie die modify-db-clusterAPI nur mit dem `--network-type` Parameter with value ausführen, `DUAL` um das Netzwerk eines Amazon Aurora Aurora-Clusters in den Dual-Stack-Modus zu ändern. Das Hinzufügen anderer Parameter zusammen mit dem Parameter `--network-type` in demselben API-Aufruf kann zu Ausfallzeiten führen.

Wenn Sie nach der Änderung keine Verbindung zum herstellen können, stellen Sie sicher, dass die Client- und Datenbanksicherheitsfirewalls und Routing-Tabellen korrekt konfiguriert sind, sodass der Datenverkehr zur Datenbank im ausgewählten Netzwerk zugelassen wird (entweder IPv4 oder IPv6). Möglicherweise müssen Sie auch Betriebssystemparameter, Bibliotheken oder Treiber ändern, um eine Verbindung über eine IPv6 Adresse herzustellen.

So ändern Sie einen , der IPv4 nur den Dual-Stack-Modus verwendet

1. Ändern Sie eine DB-Subnetzgruppe, um den Dual-Stack-Modus zu unterstützen, oder erstellen Sie eine DB-Subnetzgruppe, die den Dual-Stack-Modus unterstützt:

   a. Ordnen Sie Ihrer VPC einen IPv6 CIDR-Block zu.

      Anweisungen finden Sie unter Hinzufügen eines IPv6 CIDR-Blocks zu Ihrer VPC im Amazon VPC-Benutzerhandbuch.

   b. Hängen Sie den IPv6 CIDR-Block an alle Subnetze in Ihrer DB-Subnetzgruppe an.

      Anweisungen finden Sie unter Hinzufügen eines IPv6 CIDR-Blocks zu Ihrem Subnetz im Amazon VPC-Benutzerhandbuch.

   c. Vergewissern Sie sich, dass die DB-Subnetzgruppe den Dual-Stack-Modus unterstützt.

Wenn Sie den verwenden AWS-Managementkonsole, wählen Sie die DB-Subnetzgruppe aus und stellen Sie sicher, dass der Wert Unterstützte Netzwerktypen auf Dual, gesetzt ist. IPv4

Wenn Sie den verwenden AWS CLI, führen Sie den describe-db-subnet-groupsBefehl aus und stellen Sie sicher, dass der `SupportedNetworkType` Wert für die DB-Instance lautet`Dual, IPv4`.

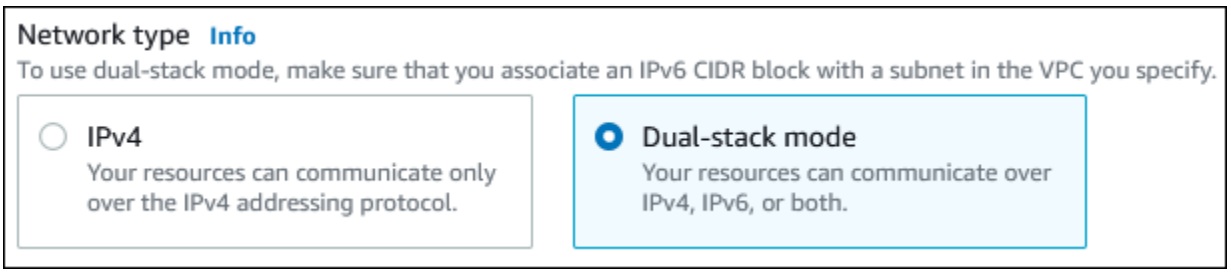
2. Ändern Sie die dem zugeordnete Sicherheitsgruppe, um IPv6 Verbindungen zur Datenbank zuzulassen, oder erstellen Sie eine neue Sicherheitsgruppe, die IPv6 Verbindungen zulässt.

   Eine Anleitung dazu finden Sie unter Sicherheitsgruppenregeln im Amazon-VPC-Benutzerhandbuch.

3. Ändern der DB-Cluster zur Unterstützung des Dual-Stack-Modus. Setzen Sie hierzu den Netzwerk-Typ auf Dual-Stack-Modus.

   Wenn Sie die Konsole verwenden, stellen Sie sicher, dass die folgenden Einstellungen korrekt sind:

   - Network type (Netzwerktyp) – Dual-stack mode (Dual-Stack-Modus)

   - DB-Subnet group (Subnetzgruppe) – Die DB-Subnetzgruppe, die Sie in einem vorherigen Schritt konfiguriert haben
   - Security group (Sicherheitsgruppe) – Die Sicherheitsgruppe, die Sie in einem vorherigen Schritt konfiguriert haben

   Wenn Sie das verwenden AWS CLI, stellen Sie sicher, dass die folgenden Einstellungen korrekt sind:

   - `--network-type` – `dual`
   - `--db-subnet-group-name` – Die DB-Subnetzgruppe, die Sie in einem vorherigen Schritt konfiguriert haben

- `--vpc-security-group-ids` – Die VPC-Sicherheitsgruppe, die Sie in einem vorherigen Schritt konfiguriert haben

Zum Beispiel:

```
aws rds modify-db-cluster --db-cluster-identifier my-cluster --network-type "DUAL"
```

4. Vergewissern Sie sich, dass die DB-Cluster den Dual-Stack-Modus unterstützt.

   Wenn Sie die Konsole verwenden, wählen Sie die Registerkarte Konfiguration für die DB-Cluster. Stellen Sie auf dieser Registerkarte sicher, dass der Wert des Netzwerk-TypsDual-Stack-Modus ist.

   Wenn Sie den verwenden AWS CLI, führen Sie den describe-db-clustersBefehl aus und stellen Sie sicher, dass der `NetworkType` Wert für den DB-Cluster lautet`dual`.

   Führen Sie den `dig` Befehl auf dem Endpunkt der Writer-DB-Instance aus, um die zugehörige IPv6 Adresse zu ermitteln.

   ```
   dig db-instance-endpoint AAAA
   ```

   Verwenden Sie den Endpunkt der Writer-DB-Instance, nicht die IPv6 Adresse, um eine Verbindung zum herzustellen.

Verfügbarkeit von Dual-Stack-Netzwerk-DB-Cluster

Dual-Stack-Netzwerk-DB-Cluster sind in allen Bereichen verfügbar, AWS-Regionen mit Ausnahme der folgenden:

- Asien-Pazifik (Hyderabad)
- Asien-Pazifik (Malaysia)
- Asien-Pazifik (Melbourne)
- Asien-Pazifik (Thailand)
- Kanada West (Calgary)
- Europa (Spain)
- Europa (Zürich)

- Israel (Tel Aviv)
- Mexiko (Zentral)
- Naher Osten (VAE)

Die folgenden DB-Engine-Versionen unterstützen DB-Instances im Dual-Stack-Netzwerk:

- Aurora-MySQL-Versionen:
  - 3.02 und höhere 3-Versionen
  - 2.09.1 und höhere 2-Versionen

  Weitere Informationen über Aurora MySQL-Versionen finden Sie in den Release Notes für Aurora MySQL.
- Aurora-PostgreSQL-Versionen:
  - 15.2 und alle höheren Versionen
  - 14.3 und höhere 14-Versionen
  - 13.7 und höhere 13-Versionen

  Weitere Informationen zu Aurora-PostgreSQL-Versionen finden Sie unter Versionshinweise für Aurora PostgreSQL.

Einschränkungen für Dual-Stack-Netzwerk-DB-Cluster

Die folgenden Einschränkungen gelten für Dual-Stack-Netzwerk-DB-Cluster:

- können das IPv6 Protokoll nicht ausschließlich verwenden. Sie können IPv4 ausschließlich oder das IPv6 AND-Protokoll verwenden (Dual-Stack-Modus). IPv4
- Amazon RDS unterstützt keine systemeigenen IPv6 Subnetze.
- DB-Cluster, die den Dual-Stack-Modus verwenden, müssen privat sein. Sie dürfen nicht öffentlich zugänglich sein.
- Sie können RDS-Proxy nicht mit DB-Cluster im Dual-Stack-Modus verwenden.

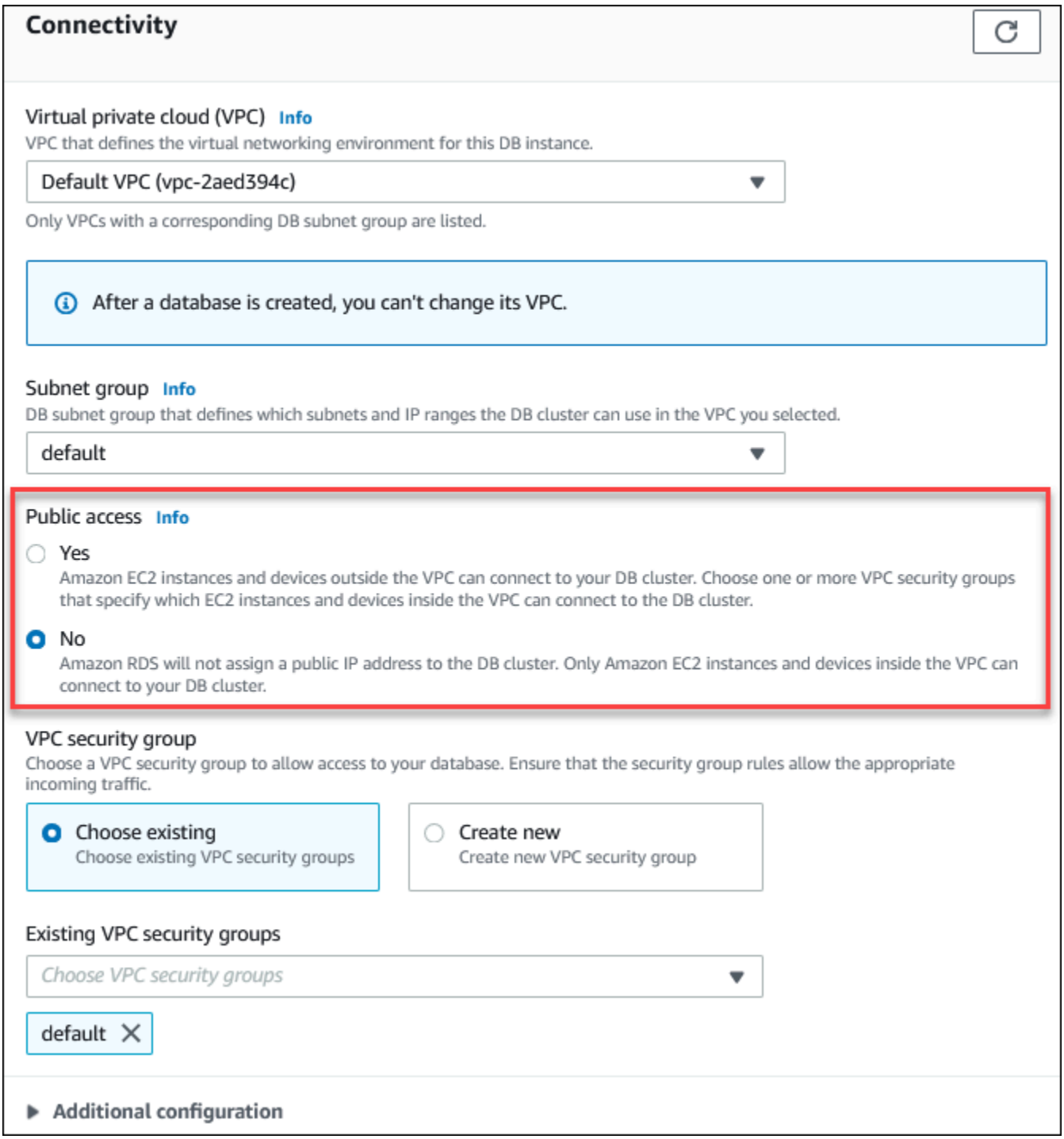
Ausblenden einer DB-Clusters in einer VPC vor dem Internet

Ein ist eine VPC, in der Sie über eine EC2 Amazon-Instance mit einer öffentlich zugänglichen Webanwendung und einen mit einer Datenbank verfügen, auf die nicht öffentlich zugegriffen werden

kann. Sie können beispielsweise eine VPC mit einem öffentlichen Subnetz und einem privaten Subnetz erstellen. EC2 Instanzen, die als Webserver fungieren, können im öffentlichen Subnetz bereitgestellt werden. Die DB-Cluster werden im privaten Subnetz bereitgestellt. Bei einer solchen Bereitstellung haben nur die Webserver Zugang zu den DB-Cluster. Eine bildliche Darstellung dieses Szenarios finden Sie unter DB-Cluster in einer VPC, auf die eine Amazon-EC2-Instance in derselben VPC zugreift.

Wenn Sie eine DB-Cluster innerhalb einer VPC starten, verfügt die DB-Cluster über eine private IP-Adresse für den Datenverkehr innerhalb der VPC. Diese private IP-Adresse ist nicht öffentlich zugänglich. Mit der Option Public access (Öffentlicher Zugriff) können Sie festlegen, ob die DB-Cluster neben der privaten IP-Adresse auch eine öffentliche IP-Adresse hat. Wenn die DB-Cluster als öffentlich zugänglich bezeichnet wird, wird ihr DNS-Endpunkt innerhalb der VPC in die private IP-Adresse aufgelöst. Es wird in die öffentliche IP-Adresse von außerhalb der VPC aufgelöst. Zugriff auf die DB-Cluster wird letztendlich von der Sicherheitsgruppe kontrolliert, die sie verwendet. Dieser öffentliche Zugang ist nicht erlaubt, wenn die dem DB-Cluster zugewiesene Sicherheitsgruppe keine eingehenden Regeln enthält, die ihn erlauben. Damit ein DB-Cluster öffentlich zugänglich ist, müssen die Subnetze in seiner DB-Subnetzgruppe außerdem über ein Internet-Gateway verfügen. Weitere Informationen finden Sie unter Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden

Sie können eine DB-Cluster ändern und die öffentliche Zugänglichkeit mit der Option Public access (Öffentlicher Zugriff) aktivieren und deaktivieren. Die folgende Abbildung zeigt die Option Public access (Öffentlicher Zugriff) im Abschnitt Additional connectivity configuration (Zusätzliche Konnektivitätskonfiguration) . Um die Option festzulegen, öffnen Sie den Abschnitt Additional connectivity configuration (Zusätzliche Konnektivitätskonfiguration) im Abschnitt Connectivity (Konnektivität) .



Hinweise zum Ändern einer DB-Instance zum Festlegen der Option Public access (Öffentlicher Zugriff) finden Sie unter Ändern einer DB-Instance in einem DB-Cluster.

Erstellen einer DB-Cluster in einer VPC

In den folgenden Verfahren wird das Erstellen einer DB-Cluster in einer VPC veranschaulicht. Um die Standard-VPC zu verwenden, können Sie mit Schritt 2 beginnen und die VPC- und DB-Subnetzgruppe verwenden, die bereits für Sie erstellt wurden. Wenn Sie eine zusätzliche VPC erstellen möchten, können Sie eine neue VPC erstellen.

> **Note**
>
> Wenn Sie möchten, dass Ihr DB-Cluster in der VPC öffentlich zugänglich ist, müssen Sie die DNS-Informationen für die VPC aktualisieren, indem Sie die VPC-Attribute DNS-Hostnamen und DNS-Auflösung aktivieren. Weitere Informationen zum Aktualisieren der DNS-Informationen für eine VPC-Instance finden Sie unter Aktualisieren des DNS-Supports für Ihre VPC.

Gehen Sie wie folgt vor, um eine DB-Instance in einer VPC zu erstellen:

- Schritt 1: Erstellen einer VPC
- Schritt 2: Erstellen einer DB-Subnetzgruppe
- Schritt 3: Erstellen einer VPC-Sicherheitsgruppe
- Schritt 4: Erstellen einer DB-Instance in der VPC

Schritt 1: Erstellen einer VPC

Erstellen Sie eine VPC mit Subnetzen in mindestens zwei Availability Zones. Diese Subnetze verwenden Sie, wenn Sie eine DB-Subnetzgruppe erstellen. Wenn Sie eine Standard-VPC verwenden, wird automatisch ein Subnetz in jeder Availability Zone der AWS-Region für Sie erstellt.

Weitere Informationen finden Sie unter Erstellen einer VPC mit privaten und öffentlichen Subnetzen oder unter Create a VPC (VPC erstellen) im Amazon VPC Benutzerhandbuch.

Schritt 2: Erstellen einer DB-Subnetzgruppe

Eine DB-Subnetzgruppe ist eine Sammlung von Subnetzen (in der Regel private Subnetze), die Sie für eine VPC erstellen und anschließend Ihren DB-Cluster zuweisen. Mit einer DB-Subnetzgruppe können Sie eine bestimmte VPC angeben, wenn Sie mithilfe der AWS CLI oder RDS-API erstellen.

Wenn Sie die Konsole verwenden, können Sie einfach die VPC und Subnetze auswählen, die Sie verwenden möchten. Jede DB-Subnetzgruppe muss über mindestens ein Subnetz in mindestens zwei Availability Zones der AWS-Region verfügen. Jede DB-Subnetzgruppe sollte über mindestens ein Subnetz für jede Availability Zone in einer bestimmten AWS-Region verfügen.

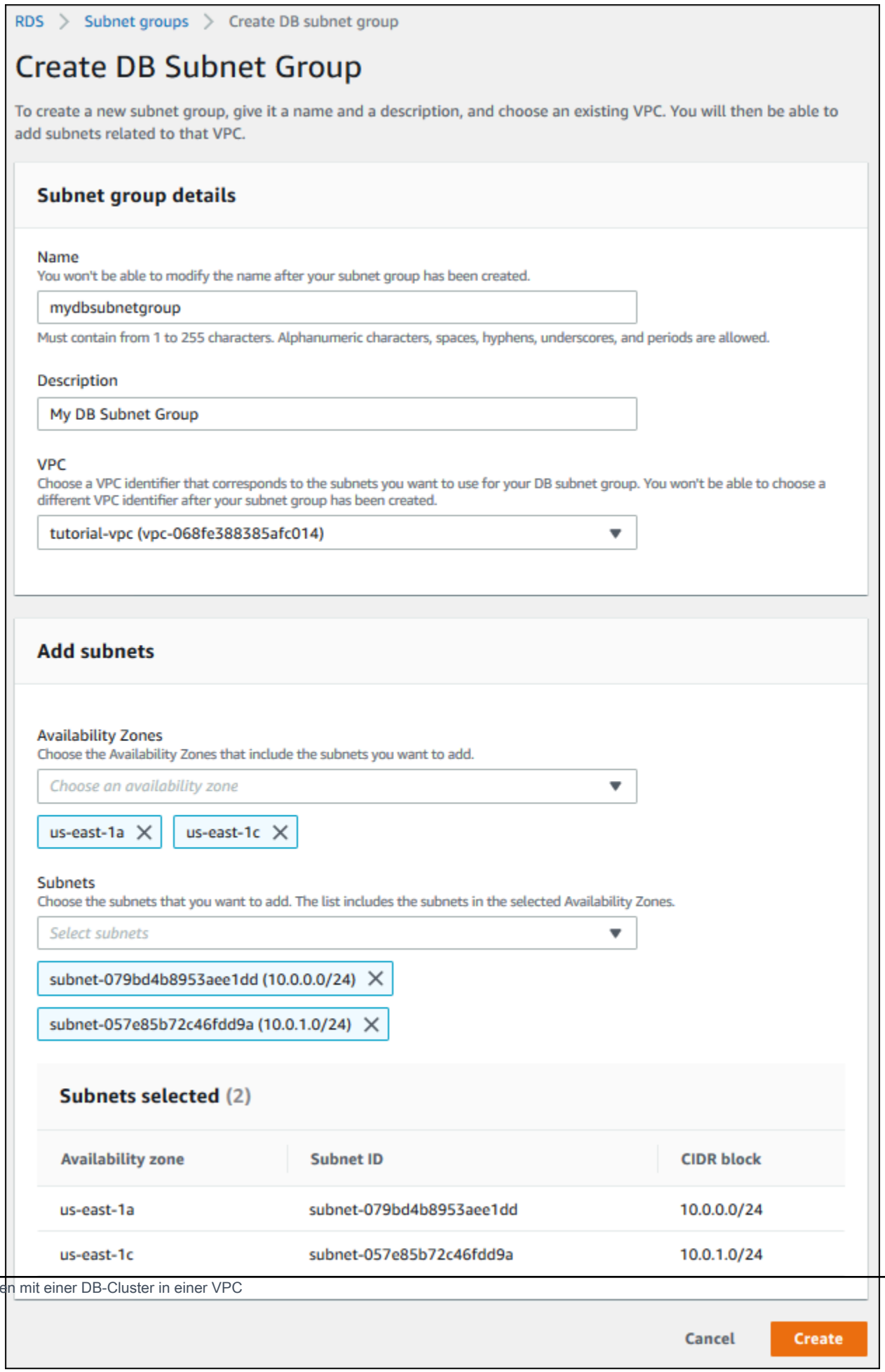
Damit öffentlich auf eine DB-Cluster zugegriffen werden kann, müssen die Subnetze in der DB-Subnetzgruppe über ein Internet-Gateway verfügen. Weitere Informationen zu Internet-Gateways für Subnetze finden Sie unter Verbinden mit dem Internet durch einen Internet-Gateway im Amazon VPC Benutzerhandbuch.

Beim Erstellen einer DB-Cluster in einer VPC müssen Sie eine DB-Subnetzgruppe auswählen. Amazon Aurora wählt ein Subnetz und eine IP-Adresse innerhalb dieses Subnetzes, um es mit Ihrem DB-Cluster zu verbinden. Wenn keine DB-Subnetzgruppen existieren, erstellt Amazon Aurora eine Standard-Subnetzgruppe, wenn Sie einen DB-Cluster erstellen. Amazon Aurora erstellt und ordnet Ihrer DB-Cluster eine Elastic-Network-Schnittstelle mit dieser IP-Adresse zu. Die DB-Cluster verwendet die Availability Zone, die das Subnetz enthält.

In diesem Schritt erstellen Sie eine DB-Subnetzgruppe und fügen die Subnetze hinzu, die Sie für Ihre VPC erstellt haben.

Erstellen einer DB-Sicherheitsgruppe

1. Öffnen Sie die Amazon RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Subnetzgruppe aus.
3. Wählen Sie DB-Subnetzgruppe erstellen aus.
4. Geben Sie im Feld Name den Namen Ihrer DB-Subnetzgruppe ein.
5. Geben Sie unter Beschreibung eine Beschreibung für Ihre DB-Subnetzgruppe ein.
6. Für VPC wählen Sie die Standard-VPC oder die zuvor erstellte VPC aus.
7. Wählen Sie im Abschnitt Subnetze hinzufügen die Availability Zones aus, die die Subnetze aus Availability Zones enthalten, und wählen Sie dann die Subnetze aus Subnetze aus.


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

8. Wählen Sie Create (Erstellen) aus.

   Ihre neue DB-Subnetzgruppe wird in der Liste der DB-Subnetzgruppen in der RDS-Konsole angezeigt. Sie können die DB-Subnetzgruppe auswählen und unten im Detailbereich ausführliche Informationen einschließlich aller Subnetze für diese Gruppe anzeigen.

Schritt 3: Erstellen einer VPC-Sicherheitsgruppe

Vor der DB-Cluster müssen Sie eine VPC-Sicherheitsgruppe erstellen, die Ihrer DB-Cluster zugeordnet wird. Wenn Sie keine VPC-Sicherheitsgruppe erstellen, können Sie die Standardsicherheitsgruppe beim Erstellen einer DB-Cluster verwenden. Eine Anleitung zum Erstellen einer Sicherheitsgruppe für Ihren DB-Cluster finden Sie unter Erstellen einer VPC-Sicherheitsgruppe für einen privaten DB-Cluster, oder unter Control traffic to resources using security groups (Kontrolle des Datenverkehrs zu Ressourcen mithilfe von Sicherheitsgruppen) im Amazon VPC Benutzerhandbuch.

Schritt 4: Erstellen einer DB-Instance in der VPC

In diesem Schritt erstellen Sie eine DB-Cluster und verwenden den VPC-Namen, die DB-Subnetzgruppe und die VPC-Sicherheitsgruppe, die Sie in den vorherigen Schritten erstellt haben.

> **Note**
>
> Wenn Sie möchten, dass Ihr DBCluster in der VPC öffentlich zugänglich ist, müssen Sie die VPC-Attribute DNS-Hostnamen und DNS-Auflösung aktivieren. Weitere Infomationen finden Sie unter DNS-Attribute für Ihre VPC im Amazon VPC-Benutzerhandbuch.

Weitere Informationen zum Erstellen eines DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters.

Wenn Sie im Abschnitt Konnektivität dazu aufgefordert werden, geben Sie den VPC-Namen, die DB-Subnetzgruppe und die VPC-Sicherheitsgruppe ein.

> **Note**
>
> VPCs Die Aktualisierung wird derzeit für Aurora-DB-Cluster nicht unterstützt.

Szenarien für den Zugriff auf eine DB-Cluster in einer VPC

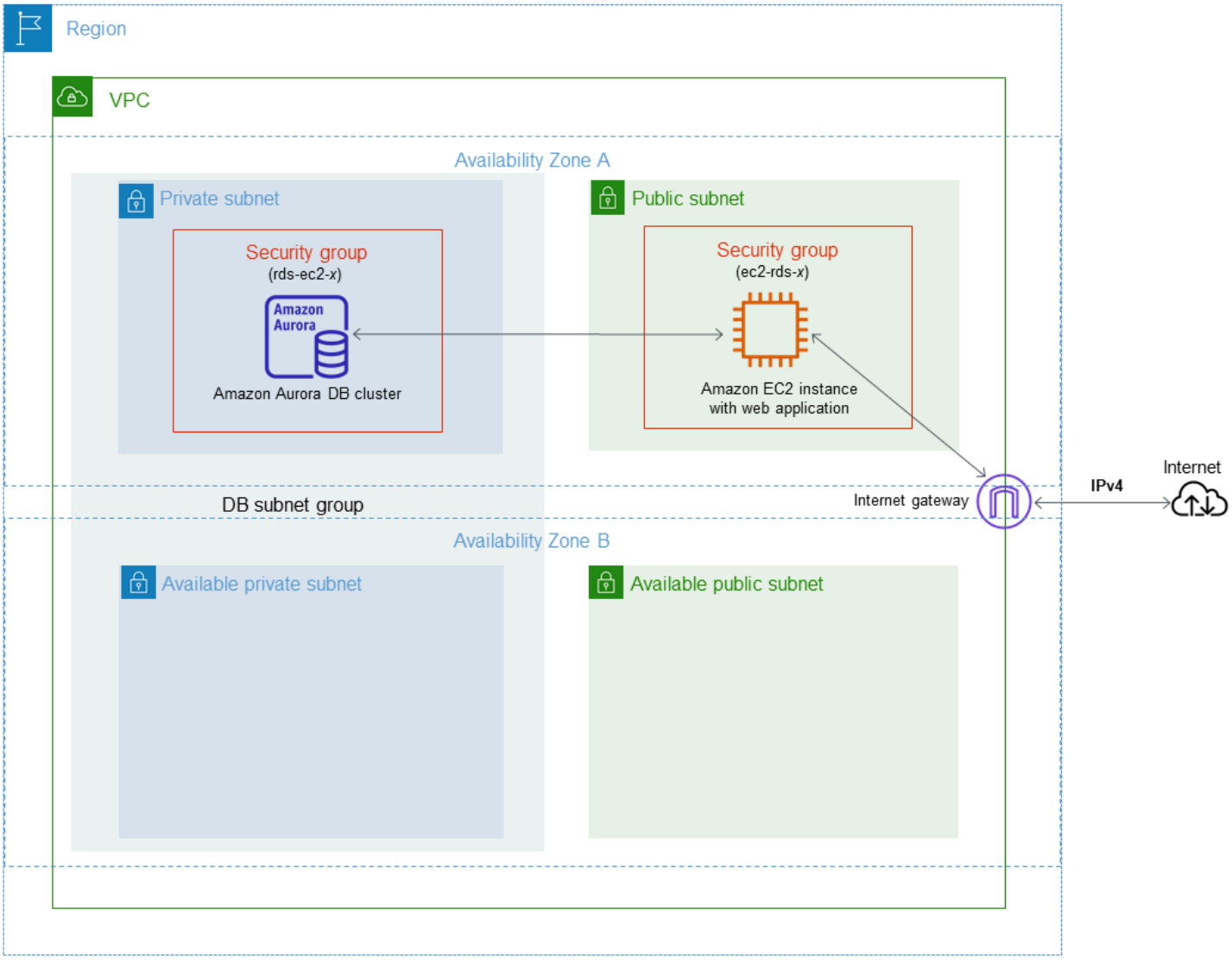
Amazon Aurora unterstützt die folgenden Szenarien für den Zugriff auf eine DB-einen Cluster in einer VPC:

- Eine Amazon-EC2-Instance in derselben VPC
- Eine EC2-Instance in einer anderen VPC
- Eine Client-Anwendung über das Internet
- Ein Privates Netzwerk

DB-Cluster in einer VPC, auf die eine Amazon-EC2-Instance in derselben VPC zugreift

Häufig werden DB-Cluster in einer VPC genutzt, um Daten mit einem Anwendungsserver zu teilen, der auf einer Amazon-EC2-Instance in derselben VPC ausgeführt wird.

Im folgenden Diagramm wird dieses Szenario veranschaulicht.



Nachfolgend finden Sie den einfachsten Weg für die Verwaltung des Zugriffs zwischen EC2-Instances und DB- Cluster in derselben VPC:

- Erstellen Sie eine VPC-Sicherheitsgruppe, in der sich Ihre DB- Cluster befinden sollen. Diese Sicherheitsgruppe kann verwendet werden, um den Zugriff auf DB- Cluster zu beschränken. Sie können beispielsweise eine benutzerdefinierte Regel für diese Sicherheitsgruppe erstellen. Dies kann den TCP-Zugriff über den Port ermöglichen, den Sie dem DB- Cluster bei der Erstellung zugewiesen haben, sowie eine IP-Adresse, die Sie für den Zugriff auf den DB- Cluster für Entwicklungs- oder andere Zwecke verwenden.
- Erstellen Sie eine VPC-Sicherheitsgruppe, der sich Ihre EC2-Instances (Webserver und Clients) befinden. Mithilfe dieser Sicherheitsgruppe können Sie bei Bedarf den Internetzugriff auf die EC2-

Instance über die Routing-Tabelle der VPC zulassen. Sie können beispielsweise Regeln für diese Sicherheitsgruppe festlegen, damit der TCP-Zugriff auf die EC2-Instance über Port 22 möglich ist.

- Erstellen Sie in der Sicherheitsgruppe benutzerdefinierte Regeln für Ihre DB- Cluster, um Verbindungen von der (für die EC2-Instances) erstellten Sicherheitsgruppe zuzulassen. Diese Regeln können jedem Mitglied der Sicherheitsgruppe den Zugang zu den DB- Clustern ermöglichen.

Es gibt ein zusätzliches öffentliches und privates Subnetz in einer separaten Availability Zone. Eine RDS-DB-Subnetzgruppe erfordert ein Subnetz in mindestens zwei Availability Zones. Das zusätzliche Subnetz erleichtert den Wechsel zu einer Multi-AZ-DB-Instance-Bereitstellung in der future.

Ein Tutorial mit einer Anleitung zum Erstellen einer VPC mit öffentlichen und privaten Subnetzen für dieses Szenario finden Sie unter Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4).

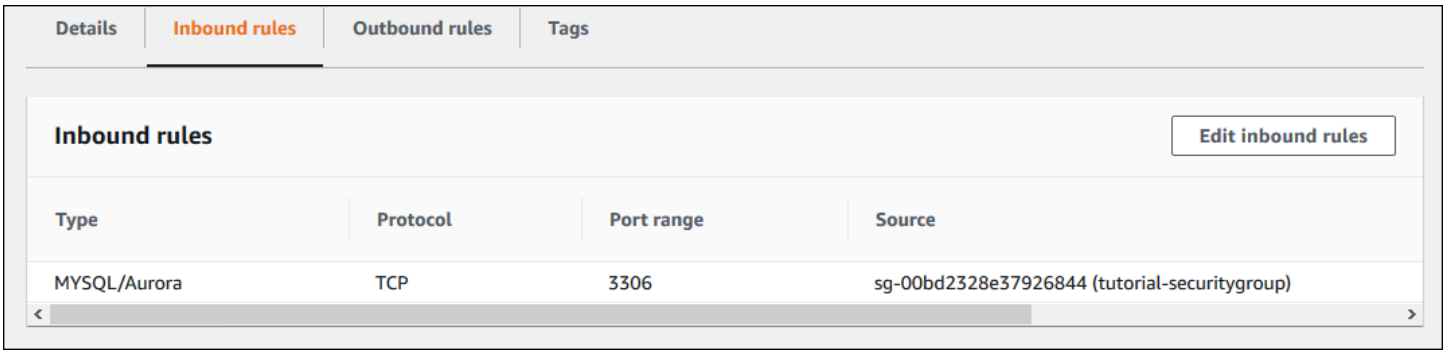
> **Tip**
>
> Sie können die Netzwerkkonnektivität zwischen einer Amazon-EC2-Instance und einem DB-Cluster automatisch beim Erstellen des DB-Clusters einrichten. Weitere Informationen finden Sie unter Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren .

Führen Sie folgende Schritte aus, um eine Regel in einer VPC-Sicherheitsgruppe zu erstellen, die Verbindungen von einer anderen Sicherheitsgruppe zulässt:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc.
2. Wählen Sie im Navigationsbereich Sicherheitsgruppen aus.
3. Erstellen oder wählen Sie eine Sicherheitsgruppe aus, der Sie den Zugriff zu Teilen einer anderen Sicherheitsgruppe erlauben möchten. Im vorhergehenden Szenario ist dies die Sicherheitsgruppe, die Sie für Ihre DB- Cluster verwenden. Wählen Sie die Registerkarte Inbound Rules (Eingehende Regeln) und anschließend Edit Inbound Rules (Eingehende Regeln bearbeiten) aus.
4. Wählen Sie auf der Seite Edit inbound rules (Regeln für eingehenden Datenverkehr bearbeiten) die Option Add Rule (Regel hinzufügen).
5. Wählen Sie für Typ den Eintrag, der dem Port entspricht, den Sie bei der Erstellung Ihres DB-Clusters verwendet haben, z. B. MYSQL/Aurora.

6. Geben Sie im Feld Source (Quelle) die ID der Sicherheitsgruppe ein. Anschließend werden die übereinstimmenden Sicherheitsgruppen auflistet. Wählen Sie die Sicherheitsgruppe mit den Mitgliedern aus, die Zugriff auf die durch diese Sicherheitsgruppe geschützten Ressourcen erhalten sollen. Im vorhergehenden Szenario ist dies die Sicherheitsgruppe, die Sie für Ihre EC2-Instance verwenden.
7. Wiederholen Sie die Schritte für das TCP-Protokoll, indem Sie eine Regel mit All TCP (Alle TCP) als Type (Typ) und Ihrer Sicherheitsgruppe im Feld Source (Quelle) erstellen. Wenn Sie das UDP-Protokoll verwenden möchten, erstellen Sie eine Regel mit Alle UDP als Typ und Ihrer Sicherheitsgruppe im Quelle.
8. Wählen Sie Save rules (Regeln speichern) aus.

Der folgende Bildschirm zeigt eine eingehende Regel mit einer Sicherheitsgruppe für ihre Quelle.

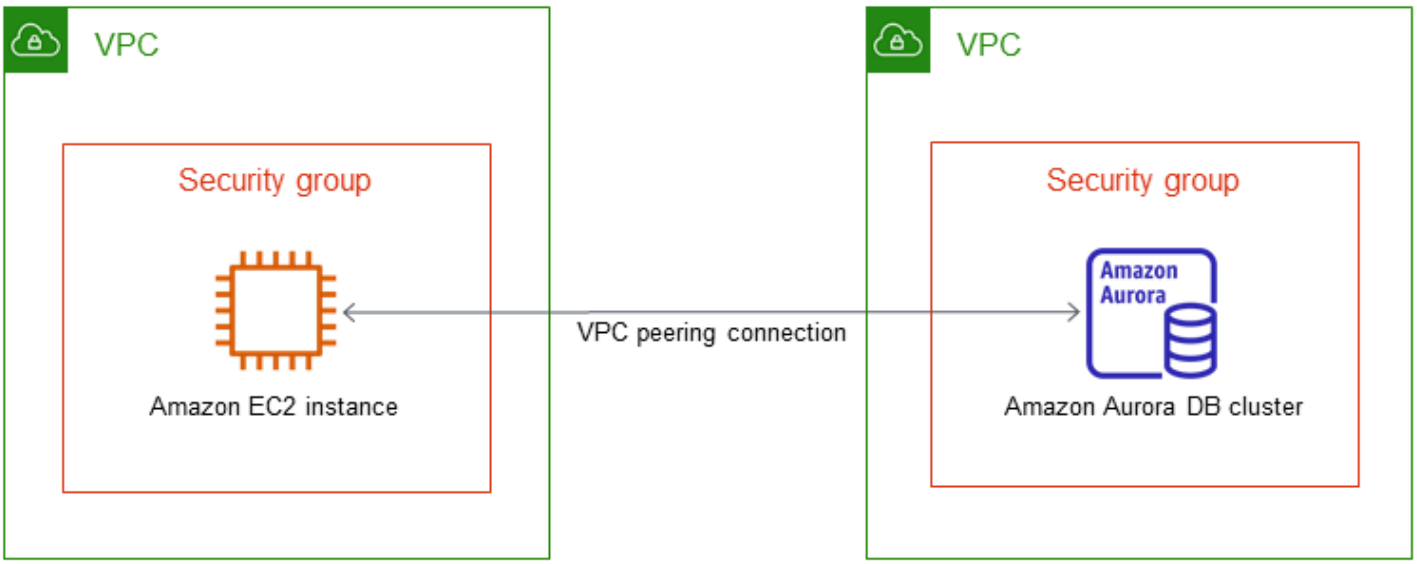


Weitere Informationen zum Herstellen einer Verbindung mit dem DB-Cluster von Ihrer EC2-Instance aus finden Sie unter Herstellen einer Verbindung mit einem Amazon Aurora-DB-Cluster.

Ein DB-Cluster in einer VPC, auf den eine EC2-Instance in einer anderen VPC zugreift

Wenn sich Ihre DB- Cluster in einer anderen VPC als die für den Zugriff darauf verwendete EC2-Instance befindet, können Sie per VPC Peering auf die DB- Cluster zugreifen.

Im folgenden Diagramm wird dieses Szenario veranschaulicht.

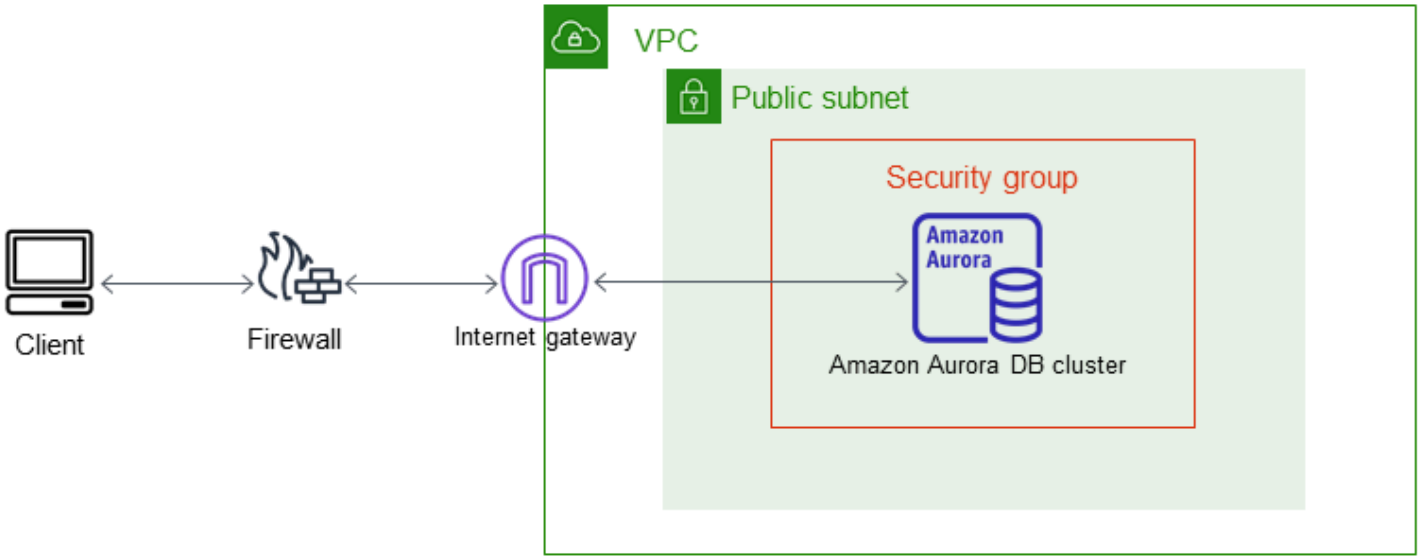


Peering: Eine VPC-Peering-Verbindung ist eine Netzwerkverbindung zwischen zwei VPCs. Diese ermöglicht die Weiterleitung des Datenverkehrs zwischen den VPCs mithilfe von privaten IP-Adressen. Ressourcen in jeder der VPCs können so miteinander kommunizieren, als befänden sie sich im selben Netzwerk. Sie können eine VPC-Peering-Verbindung zwischen Ihren eigenen VPCs, mit einer VPC in einem anderen AWS-Konto oder mit einer VPC in einer anderen AWS-Region herstellen. Weitere Informationen über VPC-Peering finden Sie unter VPC-Peering im Amazon Virtual Private Cloud-Benutzerhandbuch.

Zugriff auf eine DB-Cluster in einer VPC durch eine Client-Anwendung über das Internet

Damit eine Client-Anwendung über das Internet auf eine DB- Cluster in einer VPC zugreifen kann, konfigurieren Sie eine VPC mit einem einzelnen öffentlichen Subnetz und ein Internet-Gateway für die Kommunikation über das Internet.

Im folgenden Diagramm wird dieses Szenario veranschaulicht.



Wir empfehlen die folgende Konfiguration:

- Eine VPC der Größe /16 (z. B. CIDR: 10.0.0.0/16). Diese Größe bietet 65.536 private IP-Adressen.
- Ein Subnetz der Größe /24 (z. B. CIDR: 10.0.0.0/24). Diese Größe bietet 256 private IP-Adressen.
- Ein DB-Cluster von Amazon Aurora mit Zuordnung zur VPC und zum Subnetz. Amazon RDS weist Ihrer DB- Cluster eine IP-Adresse im Subnetz zu.
- Ein Internet-Gateway, das die VPC mit dem Internet und mit anderen AWS-Produkten verbindet.
- Eine Sicherheitsgruppe, die der DB- Cluster zugeordnet ist. Die Regeln für eingehenden Datenverkehr der Sicherheitsgruppe erlauben der Clientanwendung den Zugriff auf die DB- Cluster.

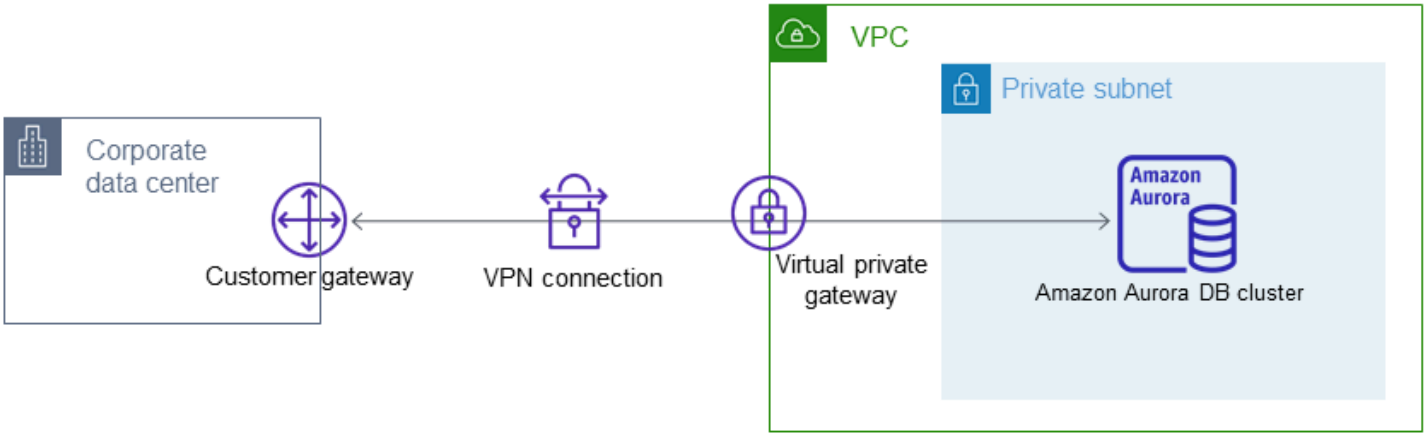
Informationen zum Erstellen einer DB- Cluster in einer VPC finden Sie unter Erstellen einer DB-Cluster in einer VPC.

Eine DB-Cluster in einer VPC, auf die von einem privaten Netzwerk zugegriffen wird

Wenn Ihre DB- Cluster nicht öffentlich zugänglich ist, haben Sie die folgenden Optionen, um von einem privaten Netzwerk aus darauf zuzugreifen:

- AWS Site-to-Site VPN-Verbindung Weitere Informationen finden Sie unter Was ist AWS Site-to-Site VPN?
- Direct Connect-Verbindung Weitere Informationen finden Sie unter Was ist Direct Connect?
- AWS Client VPN-Verbindung Weitere Informationen finden Sie unter Was ist AWS Client VPN?

Das folgende Diagramm stellt ein Szenario mit einer AWS-Site-to-Site-VPN-Verbindung dar.

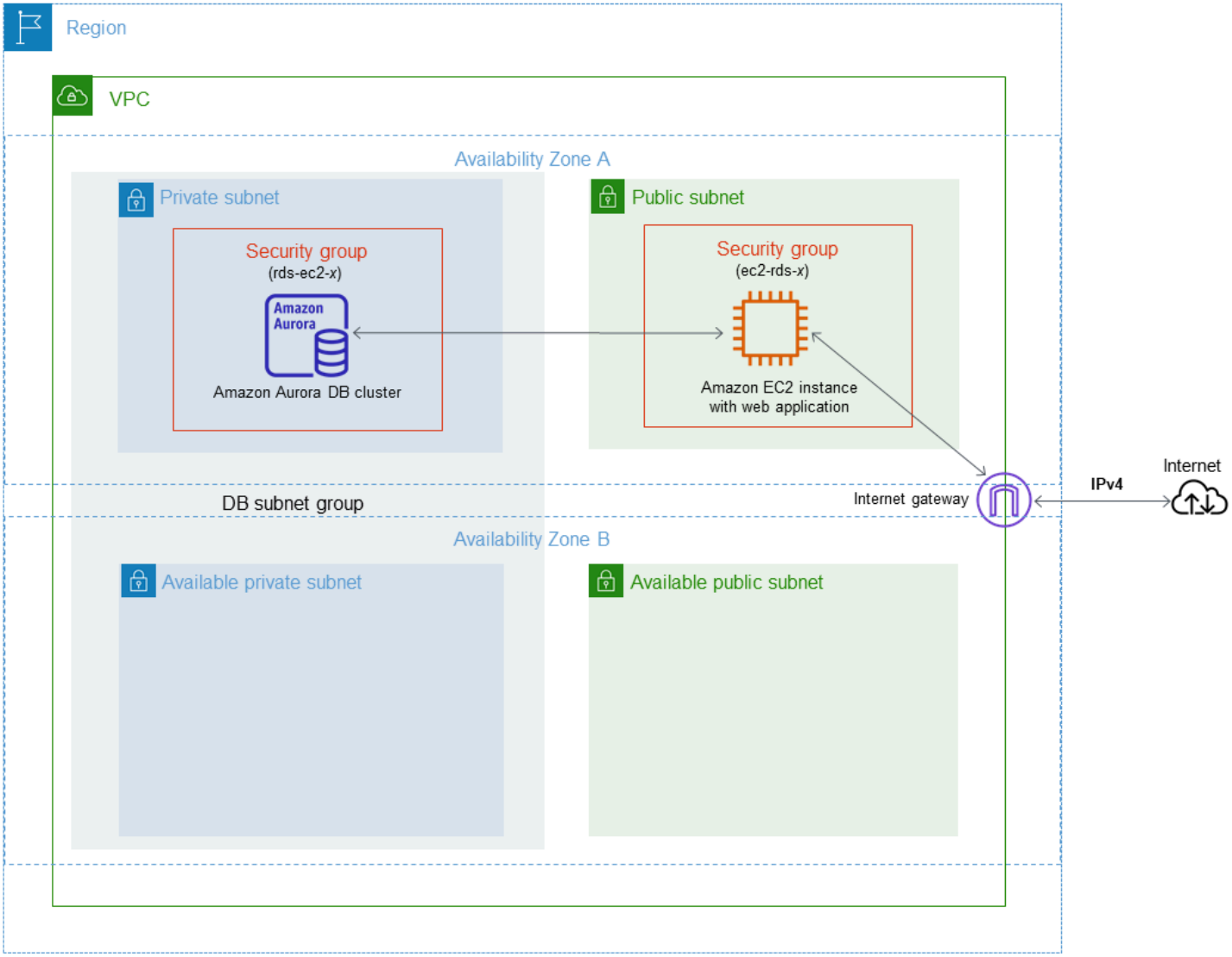


Weitere Informationen finden Sie unter Richtlinie für den Datenverkehr zwischen Netzwerken.

Tutorial: Erstellen einer VPC zur Verwendung mit einem DB-Cluster (nur IPv4)

Ein häufiges Szenario umfasst einen DB-Cluster in einer Virtual Private Cloud (VPC), die auf dem Amazon-VPC-Service basiert. Diese VPC teilt Daten mit einem Webserver, der in derselben VPC ausgeführt wird. In diesem Tutorial erstellen Sie die VPC für dieses Szenario.

Im folgenden Diagramm wird dieses Szenario veranschaulicht. Weitere Informationen zu anderen Szenarien finden Sie unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC.



Ihr DB-Cluster braucht nur für Ihren Webserver verfügbar zu sein und muss nicht vom öffentlichen Internet aus erreichbar sein. Daher erstellen Sie eine VPC sowohl mit öffentlichen als auch mit privaten Subnetzen. Der Webserver wird im öffentlichen Subnetz gehostet, damit er im öffentlichen

Internet erreicht werden kann. Der DB-Cluster wird in einem privaten Subnetz gehostet. Der Webserver kann eine Verbindung mit dem DB-Cluster herstellen, da das Hosten innerhalb derselben VPC erfolgt. Der DB-Cluster ist jedoch nicht für das öffentliche Internet verfügbar und bietet so mehr Sicherheit.

In diesem Tutorial wird ein zusätzliches öffentliches und ein privates Subnetz in einer separaten Availability Zone konfiguriert. Diese Subnetze werden im Tutorial nicht verwendet. Eine RDS-DB-Subnetzgruppe erfordert ein Subnetz in mindestens zwei Availability Zones. Das zusätzliche Subnetz erleichtert die Konfiguration von mehr als einer Aurora-DB-Instance.

In diesem Tutorial wird das Konfigurieren einer VPC für Amazon Aurora-DB-Cluster beschrieben. Ein Tutorial, das Ihnen veranschaulicht, wie Sie einen Webserver für dieses VPC-Szenario erstellen, finden Sie unter Tutorial: Erstellen eines Webservers und einer eines Amazon Aurora-DB-Clusters. Weitere Informationen zu Amazon VPC finden Sie unter Amazon-VPC-Handbuch „Erste Schritte“ und Amazon-VPC-Benutzerhandbuch.

> **Tip**
>
> Sie können die Netzwerkkonnektivität zwischen einer Amazon-EC2-Instance und einem DB-Cluster automatisch einrichten, wenn Sie den DB-Cluster erstellen. Die Netzwerkkonfiguration ähnelt der in diesem Tutorial beschriebenen Konfiguration. Weitere Informationen finden Sie unter Automatische Netzwerkkonnektivität mit einer EC2-Instance konfigurieren.

Erstellen einer VPC mit privaten und öffentlichen Subnetzen

Verwenden sie die folgenden Vorgänge, um eine VPC sowohl mit öffentlichen als auch mit privaten Subnetzen zu erstellen.

So erstellen Sie eine VPC und Subnetze

1. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie oben rechts in der AWS-Managementkonsole die Region aus, in der Sie Ihre VPC erstellen möchten. In diesem Beispiel wird die Region USA West (Oregon) verwendet.
3. Wählen Sie links oben die Option VPC-Dashboard aus. Wählen Sie Create VPC (VPC erstellen) aus, um mit dem Erstellen einer VPC zu beginnen.
4. Wählen Sie unter VPC Settings (VPC-Einstellungen) für Resources to create (Zu erstellende Ressourcen) VPC and more (VPC und mehr) aus.

5. Legen Sie für VPC settings (VPC-Einstellungen) folgende Werte fest:

   - Name tag auto-generation (Namens-Tag automatisch erstellen) – **`tutorial`**
   - IPv4 CIDR block (IPv4-CIDR-Block) – **`10.0.0.0/16`**
   - IPv6 CIDR block (IPv6-CIDR-Block) – No IPv6 CIDR block (Kein IPv6-CIDR-Block)
   - Tenancy – Default (Standard)
   - Number of Availability Zones (AZs) (Anzahl der Availability Zones (AZs) – 2
   - Customize AZs (AZs anpassen) – Übernehmen Sie die Standardwerte.
   - Number of public subnet (Anzahl der öffentlichen Subnetze) – 2
   - Number of private subnets (Anzahl der privaten Subnetze) – 2
   - Customize subnets CIDR blocks (CIDR-Blöcke für Subnetze anpassen) – Übernehmen Sie die Standardwerte.
   - NAT gateways ($) (NAT-Gateways ($)) – None (Keine)
   - VPC endpoints (VPC-Endpunkte) – None (Keine)
   - DNS options (DNS-Optionen) – Übernehmen Sie die Standardwerte.

6. Wählen Sie VPC erstellen aus.

Erstellen einer VPC-Sicherheitsgruppe für einen öffentlichen Webserver

Als Nächstes erstellen Sie eine Sicherheitsgruppe für öffentlichen Zugriff. Wenn Sie eine Verbindung mit öffentlichen EC2-Instances in Ihrer VPC herstellen möchten, fügen Sie Ihrer VPC-Sicherheitsgruppe Regeln für eingehenden Datenverkehr hinzu. Diese ermöglichen die Verbindung des Datenverkehrs aus dem Internet.

So erstellen Sie eine VPC-Sicherheitsgruppe

1. Öffnen Sie die Amazon VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie VPC Dashboard (VPC-Dashboard), Security Groups (Sicherheitsgruppen) und anschließend Create security group (Sicherheitsgruppe erstellen).
3. Legen Sie auf der Seite Create security group (Sicherheitsgruppe erstellen) die folgenden Werte fest:

   - Security group name (Name der Sicherheitsgruppe: **`tutorial-securitygroup`**
   - Description (Beschreibung: **`Tutorial Security Group`**

- VPC: Wählen Sie die VPC aus, die Sie zuvor erstellt haben, zum Beispiel: vpc-***identifier*** (tutorial-vpc)

4. Fügen Sie der Sicherheitsgruppe Regeln für den eingehenden Datenverkehr hinzu.

   a. Legen Sie die IP-Adresse fest, die für die Verbindung mit EC2-Instances in Ihrer VPC mit Secure Shell (SSH) verwendet werden soll. Um Ihre öffentliche IP-Adresse zu ermitteln, können Sie in einem anderen Browserfenster oder einer anderen Registerkarte den Service unter https://checkip.amazonaws.com verwenden. Ein Beispiel für eine IP-Adresse ist `203.0.113.25/32`.

      In vielen Fällen können Sie eine Verbindung über einen Internetdienstanbieter (ISP) oder hinter Ihrer Firewall ohne statische IP-Adresse herstellen. Suchen Sie in diesem Fall den Bereich der IP-Adressen, die von Client-Computern verwendet werden.

      > **Warning**
      >
      > Wenn Sie `0.0.0.0/0` für SSH-Zugriff verwenden, ermöglichen Sie für alle IP-Adressen den Zugriff auf Ihre öffentlichen Instances. Dieser Ansatz ist zwar für kurze Zeit in einer Testumgebung zulässig, aber für Produktionsumgebungen sehr unsicher. Für die Produktion wird nur eine bestimmte IP-Adresse bzw. ein bestimmter Adressbereich für den Zugriff auf Ihre Instances autorisiert.

   b. Wählen Sie im Abschnitt Eingehende Regeln die Option Regel hinzufügen aus.

   c. Legen Sie die folgenden Werte für Ihre neue eingehende Regel fest, um SSH den Zugriff auf Ihre Amazon-EC2-Instance zu erlauben. Dann können Sie eine Verbindung mit Ihrer Amazon-EC2-Instance herstellen, um den Webserver und andere Hilfsprogramme zu installieren. Außerdem stellen Sie eine Verbindung mit Ihrer EC2-Instance her, um Inhalte für Ihren Webserver hochzuladen.

      - Typ: **`SSH`**
      - Quelle: Die IP-Adresse bzw. der IP-Bereich aus Schritt a, zum Beispiel: **`203.0.113.25/32`**.

   d. Wählen Sie Add rule.

   e. Stellen Sie die folgenden Werte für Ihre neue eingehende Regel ein, um HTTP-Zugriff auf Ihren Webserver zuzulassen:

      - Typ: **`HTTP`**

- Quelle: **`0.0.0.0/0`**

5. Wählen Sie Create security group (Sicherheitsgruppe erstellen) aus, um die Sicherheitsgruppe zu erstellen.

   Notieren Sie sich die Sicherheitsgruppen-ID, da Sie sie später in diesem Tutorial benötigen.

Erstellen einer VPC-Sicherheitsgruppe für einen privaten DB-Cluster

Erstellen Sie eine zweite Sicherheitsgruppe für privaten Zugriff, um Ihren DB-Cluster privat zu halten. Um eine Verbindung mit privaten DB-Clustern in Ihrer VPC herzustellen, fügen Sie eingehende Regeln zu Ihrer VPC-Sicherheitsgruppe hinzu, die ausschließlich Verbindungen von Ihrem Webserver zulassen.

So erstellen Sie eine VPC-Sicherheitsgruppe

1. Öffnen Sie die Amazon VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie VPC Dashboard (VPC-Dashboard), Security Groups (Sicherheitsgruppen) und anschließend Create security group (Sicherheitsgruppe erstellen).
3. Legen Sie auf der Seite Create security group (Sicherheitsgruppe erstellen) die folgenden Werte fest:

   - Security group name (Name der Sicherheitsgruppe: **`tutorial-db-securitygroup`**
   - Description (Beschreibung: **`Tutorial DB Instance Security Group`**
   - VPC: Wählen Sie die VPC aus, die Sie zuvor erstellt haben, zum Beispiel: vpc-***`identifier`*** (tutorial-vpc)

4. Fügen Sie der Sicherheitsgruppe Regeln für den eingehenden Datenverkehr hinzu.

   a. Wählen Sie im Abschnitt Eingehende Regeln die Option Regel hinzufügen aus.

   b. Legen Sie die folgenden Werte für Ihre neue eingehende Regel fest, um MySQL-Datenverkehr von Ihrer Amazon-EC2-Instance an Port 3306 zuzulassen. In diesem Fall können Sie von Ihrem Webserver eine Verbindung mit Ihrem DB-Cluster herstellen. Auf diese Weise können Sie Daten aus Ihrer Webanwendung in Ihre Datenbank abrufen und dort speichern.

      - Typ: **`MySQL/Aurora`**
      - Source (Quelle): Die Kennung der Sicherheitsgruppe tutorial-securitygroup, die Sie zuvor in diesem Tutorial erstellt haben, z. B. sg-9edd5cfb.

5. Wählen Sie Create security group (Sicherheitsgruppe erstellen) aus, um die Sicherheitsgruppe zu erstellen.

Erstellen einer DB-Subnetzgruppe

Eine DB-Subnetzgruppe ist eine Sammlung von Subnetzen, die Sie in einer VPC erstellen und anschließend den DB-Clustern zuweisen. Mithilfe einer DB-Subnetzgruppe können Sie beim Erstellen von DB-Clustern eine bestimmte VPC festlegen.

Erstellen einer DB-Sicherheitsgruppe

1. Identifizieren Sie die privaten Subnetze für Ihre Datenbank in der VPC.

   a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
   b. Wählen Sie VPC Dashboard (VPC-Dashboard) und dann Subnets (Subnetze) aus.
   c. Beachten Sie die Subnetz-IDs der Subnetze mit den Namen tutorial-subnet-private1-us-west-2a und tutorial-subnet-private2-us-west-2b.

      Sie benötigen die Subnetz-IDs, wenn Sie Ihre DB-Subnetzgruppe erstellen.
2. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

   Stellen Sie sicher, dass Sie eine Verbindung mit der Amazon-RDS-Konsole herstellen, nicht mit der Amazon-VPC-Konsole.
3. Wählen Sie im Navigationsbereich Subnetzgruppe aus.
4. Wählen Sie Create DB subnet group (DB-Subnetzgruppe erstellen) aus.
5. Legen Sie auf der Seite DB-Subnetzgruppe erstellen unter Subnetzgruppendetails die folgenden Werte fest:

   - Name (Name: **`tutorial-db-subnet-group`**
   - Description (Beschreibung: **`Tutorial DB Subnet Group`**
   - VPC: tutorial-vpc (vpc-***`identifier`***)
6. Wählen Sie im Abschnitt Subnetze hinzufügen die Availability Zones und Subnetze aus.

   Wählen Sie für dieses Tutorial us-west-2a und us-west-2b als Availability Zones aus. Wählen Sie für Subnets (Subnetze) die privaten Subnetze aus, die Sie im vorherigen Schritt identifiziert haben.
7. Wählen Sie Erstellen aus.

Ihre neue DB-Subnetzgruppe wird in der Liste der DB-Subnetzgruppen in der RDS-Konsole angezeigt. Sie können die DB-Subnetzgruppe auswählen und unten im Detailbereich ausführliche Informationen anzeigen. Diese Informationen umfassen alle Subnetze, die der Gruppe zugeordnet sind.

> **Note**
>
> Wenn Sie diese VPC zum Vervollständigen von Tutorial: Erstellen eines Webservers und einer eines Amazon Aurora-DB-Clusters erstellt haben, erstellen Sie den DB-Cluster, indem Sie die Anweisungen unter Erstellen eines Amazon Aurora-DB-Clusters befolgen.

Löschen der VPC

Nachdem Sie die VPC und andere Ressourcen für dieses Tutorial erstellt haben, können Sie sie löschen, wenn sie nicht mehr benötigt werden.

> **Note**
>
> Wenn Sie in der VPC, die Sie für dieses Tutorial erstellt haben, Ressourcen hinzugefügt haben, müssen Sie diese möglicherweise löschen, bevor Sie die VPC löschen können. Zu diesen Ressourcen können beispielsweise Amazon-EC2-Instances oder DB-Cluster von Amazon RDS. Weitere Informationen finden Sie unter Löschen Ihrer VPC im Amazon VPC User Guide.

So löschen Sie eine VPC und zugehörige Ressourcen

1. Löschen Sie die DB-Subnetzgruppe.
   a. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
   b. Wählen Sie im Navigationsbereich Subnetzgruppe aus.
   c. Wählen Sie die DB-Subnetzgruppe aus, die Sie löschen möchten.tutorial-db-subnet-groupaus.
   d. Wählen Sie Löschen, und wählen Sie dann im Bestätigungsfenster Löschen.
2. Notieren Sie die VPC ID.

a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
b. Wählen Sie VPC Dashboard und dann VPCs.
c. Suchen Sie in der Liste die von Ihnen erstellte VPC, z. B.tutorial-vpc.
d. Notieren Sie die VPC ID (VPC-ID) der VPC, die Sie erstellt haben. Sie benötigen die VPC-ID in späteren Schritten.

3. Löschen der Sicherheitsgruppe.

a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
b. Wählen Sie VPC Dashboard und dann Sicherheitsgruppen.
c. Wählen Sie die Sicherheitsgruppe für die Amazon-RDS-DB-Instance aus, z. B.tutorial-db-securitygroupaus.
d. Wählen Sie unter Actions (Aktionen) Delete security groups (Sicherheitsgruppen löschen) und dann Delete (Löschen) auf der Bestätigungsseite aus.
e. Klicken Sie auf derSicherheitsgruppendie Sicherheitsgruppe für die Amazon EC2 Instance aus, z. B.tutorial-securitygroupaus.
f. Wählen Sie unter Actions (Aktionen) Delete security groups (Sicherheitsgruppen löschen) und dann Delete (Löschen) auf der Bestätigungsseite aus.

4. Löschen der VPC.

a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
b. Wählen Sie VPC Dashboard und dann VPCs.
c. Wählen Sie die VPC aus, die Sie löschen möchten, z. B.tutorial-vpcaus.
d. Wählen Sie unter Actions (Aktionen) die Option Delete VPC (VPC löschen) aus.

   Auf der Bestätigungsseite werden weitere Ressourcen angezeigt, die der VPC zugeordnet sind, die ebenfalls gelöscht werden, einschließlich der damit verknüpften Subnetze.
e. Geben Sie auf der Bestätigungsseite **`delete`** ein, und wählen Sie die Option Delete (Löschen) aus.

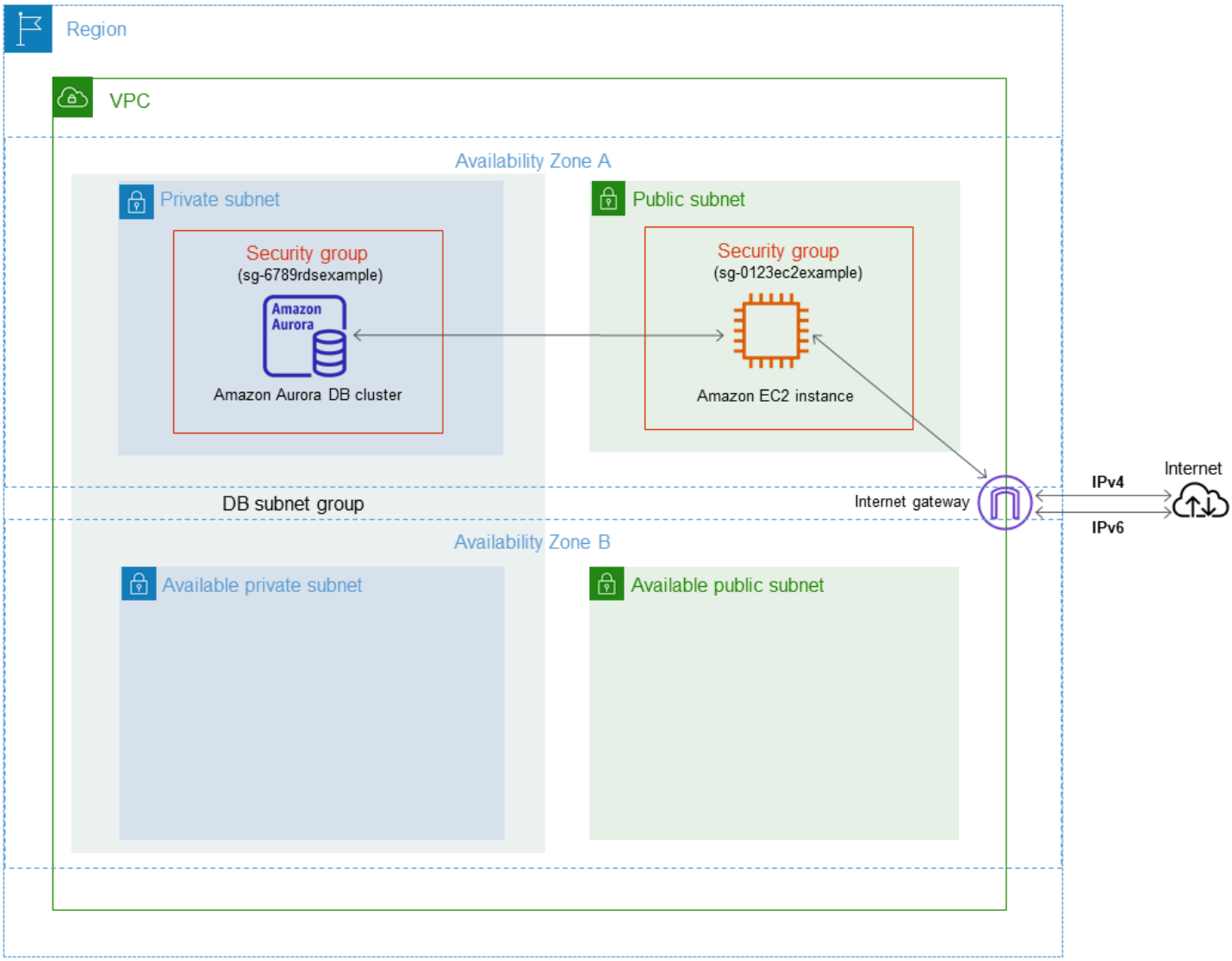
Tutorial: Erstellen einer VPC zur Verwendung mit einer DB–einem DB-Cluster (Dual-Stack-Modus)

Ein häufiges Szenario umfasst einen DB-Cluster in einer Virtual Private Cloud (VPC), die auf dem Amazon-VPC-Service basiert. Diese VPC teilt Daten mit einer öffentlichen Amazon-EC2-Instance, die in derselben VPC ausgeführt wird.

In diesem Tutorial erstellen Sie die VPC für dieses Szenario, die mit einer Datenbank arbeitet, die im Dual-Stack-Modus ausgeführt wird. Dual-Stack-Modus, um eine Verbindung über das IPv6-Adressierungsprotokoll zu ermöglichen. Weitere Informationen über die IP-Adressierung finden Sie unter Amazon-Aurora-IP-Adressierung.

Dual-Stack-Netzwerk-Cluster werden in den meisten Regionen unterstützt. Weitere Informationen finden Sie unter Verfügbarkeit von Dual-Stack-Netzwerk-DB-Cluster. Informationen zu den Einschränkungen des Dual-Stack-Modus finden Sie unter Einschränkungen für Dual-Stack-Netzwerk-DB-Cluster.

Im folgenden Diagramm wird dieses Szenario veranschaulicht.



Weitere Informationen zu anderen Szenarien finden Sie unter Szenarien für den Zugriff auf eine DB-Cluster in einer VPC.

Ihr DB-Cluster muss nur für Ihre Amazon-EC2-Instance verfügbar sein und nicht im öffentlichen Internet. Daher erstellen Sie eine VPC sowohl mit öffentlichen als auch mit privaten Subnetzen. Die Amazon-EC2-Instance wird im öffentlichen Subnetz gehostet, damit sie im öffentlichen Internet erreicht werden kann. Der DB-Cluster wird in einem privaten Subnetz gehostet. Die Amazon-EC2-Instance kann eine Verbindung mit dem DB-Cluster herstellen, da sie innerhalb derselben VPC gehostet wird. Der DB-Cluster ist jedoch nicht für das öffentliche Internet verfügbar und bietet so mehr Sicherheit.

In diesem Tutorial wird ein zusätzliches öffentliches und ein privates Subnetz in einer separaten Availability Zone konfiguriert. Diese Subnetze werden im Tutorial nicht verwendet. Eine RDS DB-

Subnetzgruppe erfordert ein Subnetz in mindestens zwei Availability Zones. Das zusätzliche Subnetz erleichtert die Konfiguration von mehr als einer Aurora-DB-Instance.

Zum Erstellen eines DB-Clusters, der den Dual-Stack-Modus verwendet, geben Sie Dual-stack mode (Dual-Stack-Modus) für die Einstellung Network type (Netzwerktyp) ein. Sie können einen DB-Cluster mit der gleichen Einstellung auch ändern. Weitere Informationen zum Erstellen eines DB-Clusters finden Sie unter Erstellen eines Amazon Aurora-DB Clusters. Weitere Informationen über das Ändern eines DB-Clusters finden Sie unter Ändern eines Amazon Aurora-DB-Clusters.

In diesem Tutorial wird das Konfigurieren einer VPC für Amazon Aurora-DB-Cluster beschrieben. Weitere Informationen zur Amazon VPC-Sicherheit finden Sie im Amazon VPC-Benutzerhandbuch.

Erstellen einer VPC mit privaten und öffentlichen Subnetzen

Verwenden sie die folgenden Vorgänge, um eine VPC sowohl mit öffentlichen als auch mit privaten Subnetzen zu erstellen.

So erstellen Sie eine VPC und Subnetze

1. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie oben rechts in der AWS-Managementkonsole die Region aus, in der Sie Ihre VPC erstellen möchten. In diesem Beispiel wird die Region USA Ost (Ohio) verwendet.
3. Wählen Sie links oben die Option VPC-Dashboard aus. Wählen Sie Create VPC (VPC erstellen) aus, um mit dem Erstellen einer VPC zu beginnen.
4. Wählen Sie unter VPC Settings (VPC-Einstellungen) für Resources to create (Zu erstellende Ressourcen) VPC and more (VPC und mehr) aus.
5. Legen Sie für die übrigen VPC settings (VPC-Einstellungen) folgende Werte fest:

    - Name tag auto-generation (Namens-Tag automatisch erstellen) – **`tutorial-dual-stack`**
    - IPv4 CIDR block (IPv4-CIDR-Block) – **`10.0.0.0/16`**
    - IPv6 CIDR block (IPv6-CIDR-Block) – Amazon-provided IPv6 CIDR block (Von Amazon bereitgestellter IPv6-CIDR-Block)
    - Tenancy – Default (Standard)
    - Number of Availability Zones (AZs) (Anzahl der Availability Zones (AZs) – 2
    - Customize AZs (AZs anpassen) – Übernehmen Sie die Standardwerte.
    - Number of public subnet (Anzahl der öffentlichen Subnetze) – 2
    - Number of private subnets (Anzahl der privaten Subnetze) – 2

- Customize subnets CIDR blocks (CIDR-Blöcke für Subnetze anpassen) – Übernehmen Sie die Standardwerte.
- NAT gateways ($) (NAT-Gateways ($)) – None (Keine)
- Egress only internet gateway (Internet-Gateway nur für ausgehenden Datenverkehr) – No (Nein)
- VPC endpoints (VPC-Endpunkte) – None (Keine)
- DNS options (DNS-Optionen) – Übernehmen Sie die Standardwerte.

> **Note**
>
> Amazon RDS erfordert mindestens zwei Subnetze in zwei verschiedenen Availability Zones, um Multi-AZ-Bereitstellungen von DB-Instances zu unterstützen. In diesem Tutorial wird eine Single-AZ-Bereitstellung erstellt, aber die Anforderung erleichtert die spätere Konvertierung in eine Multi-AZ-Bereitstellung von DB-Instances.

6. Wählen Sie VPC erstellen aus.

Erstellen einer VPC-Sicherheitsgruppe für eine öffentliche Amazon-EC2-Instance

Als Nächstes erstellen Sie eine Sicherheitsgruppe für öffentlichen Zugriff. Wenn Sie eine Verbindung mit öffentlichen EC2-Instances in Ihrer VPC herstellen möchten, fügen Sie Ihrer VPC-Sicherheitsgruppe eingehende Regeln hinzu, die Verbindungen aus dem Internet zulassen.

So erstellen Sie eine VPC-Sicherheitsgruppe

1. Öffnen Sie die Amazon VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie VPC Dashboard (VPC-Dashboard), Security Groups (Sicherheitsgruppen) und anschließend Create security group (Sicherheitsgruppe erstellen).
3. Legen Sie auf der Seite Create security group (Sicherheitsgruppe erstellen) die folgenden Werte fest:

   - Security group name (Name der Sicherheitsgruppe: **`tutorial-dual-stack-securitygroup`**
   - Description (Beschreibung: **`Tutorial Dual-Stack Security Group`**

- VPC: Wählen Sie die VPC aus, die Sie zuvor erstellt haben, zum Beispiel: vpc-***identifier*** (tutorial-dual-stack-vpc)

4. Fügen Sie der Sicherheitsgruppe Regeln für den eingehenden Datenverkehr hinzu.

   a. Legen Sie die IP-Adresse fest, die für die Verbindung mit EC2-Instances in Ihrer VPC mit Secure Shell (SSH) verwendet werden soll.

      Ein Beispiel für eine Internet Protocol Version 4 (IPv4)-Adresse ist `203.0.113.25/32`. Ein Beispiel für einen Internet Protocol Version 6 (IPv6)-Adressbereich ist `2001:db8:1234:1a00::/64`.

      In vielen Fällen können Sie eine Verbindung über einen Internetdienstanbieter (ISP) oder hinter Ihrer Firewall ohne statische IP-Adresse herstellen. Suchen Sie in diesem Fall den Bereich der IP-Adressen, die von Client-Computern verwendet werden.

      > **Warning**
      >
      > Wenn Sie `0.0.0.0/0` für IPv4 oder `::0` für IPv6 verwenden, lassen Sie für alle IP-Adressen den Zugriff auf Ihre öffentlichen Instances mit SSH zu. Dieser Ansatz ist zwar für kurze Zeit in einer Testumgebung zulässig, aber für Produktionsumgebungen sehr unsicher. Autorisieren Sie in Produktionsumgebungen nur eine bestimmte IP-Adresse bzw. einen bestimmten Adressbereich für den Zugriff auf Ihre Instances.

   b. Wählen Sie im Abschnitt Eingehende Regeln die Option Regel hinzufügen aus.

   c. Legen Sie die folgenden Werte für Ihre neue eingehende Regel fest, um Secure Shell (SSH) den Zugriff auf Ihre Amazon-EC2-Instance zu erlauben. In diesem Fall können Sie eine Verbindung mit Ihrer EC2-Instance herstellen, um SQL-Clients und andere Anwendungen zu installieren. Geben Sie eine IP-Adresse an, damit Sie auf Ihre EC2-Instance zugreifen können:

      - Typ: **SSH**
      - Source (Quelle): Die IP-Adresse bzw. der IP-Bereich aus Schritt a. Ein Beispiel für eine IPv4-IP-Adresse ist **203.0.113.25/32**. Ein Beispiel für eine IPv6-IP-Adresse ist **2001:DB8::/32**.

5. Wählen Sie Create security group (Sicherheitsgruppe erstellen) aus, um die Sicherheitsgruppe zu erstellen.

Notieren Sie sich die Sicherheitsgruppen-ID, da Sie sie später in diesem Tutorial benötigen.

Erstellen einer VPC-Sicherheitsgruppe für einen privaten DB-Cluster

Erstellen Sie eine zweite Sicherheitsgruppe für privaten Zugriff, um Ihren DB-Cluster privat zu halten. Um eine Verbindung mit privaten DB-Clustern in Ihrer VPC herzustellen, fügen Sie Regeln für eingehenden Datenverkehr zu Ihrer VPC-Sicherheitsgruppe hinzu. Diese lassen ausschließlich Datenverkehr von Ihrer Amazon-EC2-Instance zu.

So erstellen Sie eine VPC-Sicherheitsgruppe

1. Öffnen Sie die Amazon VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie VPC Dashboard (VPC-Dashboard), Security Groups (Sicherheitsgruppen) und anschließend Create security group (Sicherheitsgruppe erstellen).
3. Legen Sie auf der Seite Create security group (Sicherheitsgruppe erstellen) die folgenden Werte fest:

   - Security group name (Name der Sicherheitsgruppe: **`tutorial-dual-stack-db-securitygroup`**
   - Description (Beschreibung: **`Tutorial Dual-Stack DB Instance Security Group`**
   - VPC: Wählen Sie die VPC aus, die Sie zuvor erstellt haben, zum Beispiel: vpc-***`identifier`*** (tutorial-dual-stack-vpc)
4. Fügen Sie der Sicherheitsgruppe Regeln für den eingehenden Datenverkehr hinzu:

   a. Wählen Sie im Abschnitt Eingehende Regeln die Option Regel hinzufügen aus.

   b. Legen Sie die folgenden Werte für Ihre neue eingehende Regel fest, um MySQL-Datenverkehr von Ihrer Amazon-EC2-Instance an Port 3306 zuzulassen. In diesem Fall können Sie von Ihrer EC2-Instance eine Verbindung mit Ihrem DB-Cluster herstellen. Dies bedeutet, dass Sie Daten aus Ihrer EC2-Instance an Ihre Datenbank senden können.

      - Type (Typ) MySQL/Aurora
      - Source (Quelle): Die Kennung der Sicherheitsgruppe tutorial-dual-stack-securitygroup, die Sie zuvor in diesem Tutorial erstellt haben, z. B. sg-9edd5cfb.
5. Um die Sicherheitsgruppe zu erstellen, wählen Sie Sicherheitsgruppe erstellen aus.

Erstellen einer DB-Subnetzgruppe

Eine DB-Subnetzgruppe ist eine Sammlung von Subnetzen, die Sie in einer VPC erstellen und anschließend den DB-Clustern zuweisen. Mit einer DB-Subnetzgruppe können Sie beim Erstellen von DB-Clustern eine bestimmte VPC angeben. Wenn Sie eine DB-Sicherheitsgruppe erstellen möchten, die mit DUAL kompatibel ist, müssen alle Subnetze mit DUAL kompatibel sein. Um mit DUAL kompatibel zu sein, muss einem Subnetz ein IPv6 CIDR zugeordnet sein.

Erstellen einer DB-Sicherheitsgruppe

1. Identifizieren Sie die privaten Subnetze für Ihre Datenbank in der VPC.

   a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.

   b. Wählen Sie VPC Dashboard (VPC-Dashboard) und dann Subnets (Subnetze) aus.

   c. Beachten Sie die Subnetz-IDs der Subnetze mit den Namen tutorial-dual-stack-subnet-private1-us-west-2a und tutorial-dual-stack-subnet-private2-us-west-2b.

      Sie benötigen die Subnetz-IDs, wenn Sie Ihre DB-Subnetzgruppe erstellen.

2. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

   Stellen Sie sicher, dass Sie eine Verbindung mit der Amazon-RDS-Konsole herstellen, nicht mit der Amazon-VPC-Konsole.

3. Wählen Sie im Navigationsbereich Subnetzgruppe aus.

4. Wählen Sie Create DB subnet group (DB-Subnetzgruppe erstellen) aus.

5. Legen Sie auf der Seite DB-Subnetzgruppe erstellen unter Subnetzgruppendetails die folgenden Werte fest:

   - Name (Name: **`tutorial-dual-stack-db-subnet-group`**
   - Description (Beschreibung: **`Tutorial Dual-Stack DB Subnet Group`**
   - VPC: tutorial-dual-stack-vpc (vpc-***`identifier`***)

6. Wählen Sie im Abschnitt Add subnets (Subnetze hinzufügen) Werte für die Availability Zones und Subnets (Subnetze) aus.

   Wählen Sie für dieses Tutorial us-east-2a und us-east-2b als Availability Zones aus. Wählen Sie für Subnets (Subnetze) die privaten Subnetze aus, die Sie im vorherigen Schritt identifiziert haben.

7. Wählen Sie Erstellen aus.

Ihre neue DB-Subnetzgruppe wird in der Liste der DB-Subnetzgruppen in der RDS-Konsole angezeigt. Sie können die DB-Subnetzgruppe auswählen, um ihre Details anzuzeigen. Dazu gehören die unterstützten Adressierungsprotokolle und alle Subnetze, die mit der Gruppe verknüpft sind, sowie der von der DB-Subnetzgruppe unterstützte Netzwerktyp.

Erstellen einer Amazon-EC2-Instance im Dual-Stack-Modus

Befolgen Sie die Anleitungen unter Starten einer Instance mit dem neuen Launch Instance Wizard im Amazon-EC2-Benutzerhandbuch, um eine Amazon-EC2-Instance zu erstellen.

Legen Sie auf der Seite Configure Instance Details (Konfigurieren von Instance-Detail)s wie im Folgenden gezeigt die folgenden Werte fest und behalten Sie die Standardwerte für die anderen Werte bei:

- Network (Netzwerk) – Wählen Sie eine vorhandene VPC mit öffentlichen und privaten Subnetzen aus, z. B. tutorial-dual-stack-vpc (vpc-*identifier*), die in Erstellen einer VPC mit privaten und öffentlichen Subnetzen erstellt wurde.
- Subnet (Subnetz) – Wählen Sie ein vorhandenes öffentliches Subnetz aus, z. B.subnet-***identifier*** | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a, das in Erstellen einer VPC-Sicherheitsgruppe für eine öffentliche Amazon-EC2-Instance erstellt wurde.
- Auto-assign Public IP (Öffentliche IP-Adresse automatisch zuweisen) – Wählen Sie Enable (Aktivieren) aus.
- Auto-assign IPv6 IP (IPv6-IP automatisch zuweisen) – Wählen Sie Enable (Aktivieren) aus.
- Firewall (security groups) (Firewall (Sicherheitsgruppen)) – Wählen Sie Select an existing security group (Eine vorhandene Sicherheitsgruppe auswählen) aus.
- Common security groups (Allgemeine Sicherheitsgruppen) – Wählen Sie eine vorhandene Sicherheitsgruppe aus, z. B. die Gruppe `tutorial-securitygroup`, die Sie in Erstellen einer VPC-Sicherheitsgruppe für eine öffentliche Amazon-EC2-Instance erstellt haben. Stellen Sie sicher, dass die von Ihnen gewählte Sicherheitsgruppe Regeln für eingehenden Datenverkehr für Secure Shell (SSH) und HTTP-Zugriff enthält.

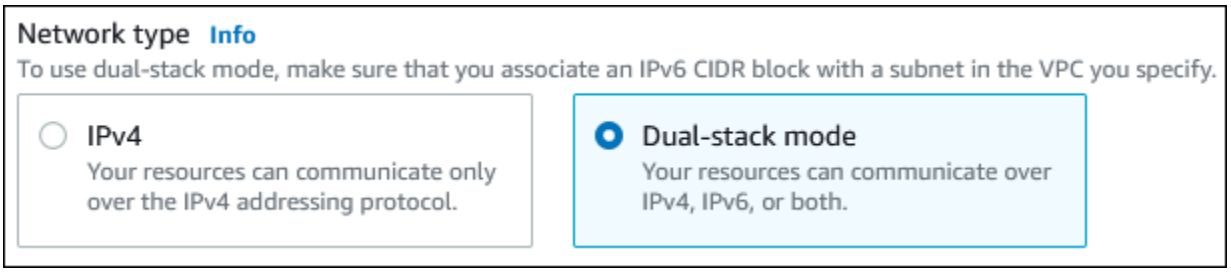
Erstellen eines DB-Clusters im Dual-Stack-Modus

In diesem Schritt erstellen Sie einen DB-Cluster zum Ausführen im Dual-Stack-Modus.

So erstellen Sie eine DB-Instance

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie in der oberen rechten Ecke der Konsole die AWS-Region aus, in der Sie die DB-Cluster erstellen möchten. In diesem Beispiel wird die Region USA Ost (Ohio) verwendet.
3. Wählen Sie im Navigationsbereich Databases (Datenbanken) aus.
4. Wählen Sie Create database (Datenbank erstellen) aus.
5. Stellen Sie auf der Seite Create database (Datenbank erstellen), die nachfolgend dargestellt ist, sicher, dass die Option Standard create (Standarderstellung) ausgewählt ist, und wählen Sie dann den DB-Engine-Typ Aurora MySQL aus.
6. Legen Sie im Abschnitt Connectivity folgende Werte fest:

   - Network type (Netzwerktyp) – Wählen Sie Dual-stack mode (Dual-Stack-Modus) aus.

   - Virtual Private Cloud (VPC) Wählen Sie eine vorhandene VPC mit öffentlichen und privaten Subnetzen aus, z. B. tutorial-dual-stack-vpc (vpc-*identifier*), die in Erstellen einer VPC mit privaten und öffentlichen Subnetzen erstellt wurde.

     Die VPC muss Subnetze in verschiedenen Availability Zones haben.

   - DB Subnet group (DB-Subnetzgruppe) – Wählen Sie eine DB-Subnetzgruppe für die VPC aus, z. B. tutorial-dual-stack-db-subnet-group, die in Erstellen einer DB-Subnetzgruppe erstellt wurde.
   - Public access (Öffentlicher Zugriff) – Wählen Sie No (Nein) aus.
   - VPC security group (firewall) (VPC-Sicherheitsgruppe (Firewall)) – Wählen Sie Choose existing (Vorhandene auswählen) aus.
   - Existing VPC security groups (Vorhandene VPC-Sicherheitsgruppen) – Wählen Sie eine vorhandene VPC-Sicherheitsgruppe aus, die für den privaten Zugriff konfiguriert ist, z. B. tutorial-dual-stack-db-securitygroup, die in Erstellen einer VPC-Sicherheitsgruppe für einen privaten DB-Cluster erstellt wurde.

Entfernen Sie andere Sicherheitsgruppen wie die Standardsicherheitsgruppe, indem Sie das jeweilige X wählen.

- Availability Zone – Wählen Sie us-west-2a aus.

  Um AZ-übergreifenden Datenverkehr zu vermeiden, stellen Sie sicher, dass sich die DB-Instance und die EC2-Instance in derselben Availability Zone befinden.

7. In den übrigen Abschnitten geben Sie die Einstellungen für Ihren DB-Cluster an. Weitere Informationen zu den einzelnen Einstellungen finden Sie unter Einstellungen für Aurora-DB-Cluster.

Herstellen einer Verbindung mit Ihrer Amazon-EC2-Instance und dem DB-Cluster

Nachdem Ihre Amazon-EC2-Instance und den DB-Cluster im Dual-Stack-Modus erstellt wurden, können Sie sich über das IPv6-Protokoll mit jeder einzelnen Instance verbinden. Befolgen Sie die Anleitungen unter Herstellen einer Verbindung zur Linux-Instance im Amazon-EC2-Benutzerhandbuch, um eine Verbindung mit einer Amazon-EC2-Instance unter Verwendung des IPv6-Protokolls herzustellen.

Um von der Amazon-EC2-Instance aus eine Verbindung zu Ihrem Aurora-MySQL-DB-Cluster herzustellen, folgen Sie den Anweisungen unter Verbindung mit einem DB-Cluster von Aurora MySQL herstellen.

Löschen der VPC

Nachdem Sie die VPC und andere Ressourcen für dieses Tutorial erstellt haben, können Sie sie löschen, wenn sie nicht mehr benötigt werden.

Wenn Sie in der VPC, die Sie für dieses Tutorial erstellt haben, Ressourcen hinzugefügt haben, müssen Sie diese möglicherweise löschen, bevor Sie die VPC löschen können. Beispiele für Ressourcen sind Amazon-EC2-Instances oder DB-Cluster. Weitere Informationen finden Sie unter Löschen Ihrer VPC im Amazon VPC User Guide.

So löschen Sie eine VPC und zugehörige Ressourcen

1. Löschen Sie die DB-Subnetzgruppe:

   a. Öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.

   b. Wählen Sie im Navigationsbereich Subnetzgruppe aus.

c. Wählen Sie die DB-Subnetzgruppe aus, die Sie löschen möchten, z. B. tutorial-db-subnet-group.

d. Wählen Sie Löschen, und wählen Sie dann im Bestätigungsfenster Löschen.

2. Notieren Sie sich die VPC-ID.

a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.

b. Wählen Sie VPC Dashboard und dann VPCs.

c. Suchen Sie die von Ihnen erstellte VPC in der Liste, z. B. tutorial-dual-stack-vpc.

d. Notieren Sie den Wert VPC ID (VPC-ID) der VPC, die Sie erstellt haben. Sie benötigen diese VPC-ID in den nachfolgenden Schritten.

3. Löschen der Sicherheitsgruppen:

a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.

b. Wählen Sie VPC Dashboard und dann Sicherheitsgruppen.

c. Wählen Sie die Sicherheitsgruppe für die Amazon-RDS-DB-Instance aus, z. B. tutorial-dual-stack-db-securitygroup.

d. Wählen Sie unter Actions (Aktionen) Delete security groups (Sicherheitsgruppen löschen) und dann Delete (Löschen) auf der Bestätigungsseite aus.

e. Wählen Sie auf der Seite Security Groups (Sicherheitsgruppen) die Sicherheitsgruppe für die Amazon-EC2-Instance aus, z. B. tutorial-dual-stack-securitygroup.

f. Wählen Sie unter Actions (Aktionen) Delete security groups (Sicherheitsgruppen löschen) und dann Delete (Löschen) auf der Bestätigungsseite aus.

4. Löschen des NAT-Gateways:

a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.

b. Wählen Sie VPC Dashboard und dann NAT Gateways.

c. Wählen Sie das NAT-Gateway der VPC aus, die Sie erstellt haben. Verwenden Sie die VPC-ID, um das richtige NAT-Gateway zu identifizieren.

d. Wählen Sie unter Actions (Aktionen) die Option Delete NAT gateway (NAT-Gateway löschen) aus.

e. Geben Sie auf der Bestätigungsseite **`delete`** ein, und wählen Sie die Option Delete (Löschen) aus.

5. Löschen der VPC:

a. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
b. Wählen Sie VPC Dashboard und dann VPCs.
c. Wählen Sie die VPC aus, die Sie löschen möchten, z. B.tutorial-dual-stack-vpc.
d. Wählen Sie unter Actions (Aktionen) die Option Delete VPC (VPC löschen) aus.

   Auf der Bestätigungsseite werden weitere Ressourcen angezeigt, die der VPC zugeordnet sind, die ebenfalls gelöscht werden, einschließlich der damit verknüpften Subnetze.
e. Geben Sie auf der Bestätigungsseite **`delete`** ein, und wählen Sie die Option Delete (Löschen) aus.

6. Freigeben der Elastic-IP-Adressen:

a. Öffnen Sie die Amazon EC2-Konsole unter https://console.aws.amazon.com/ec2/.
b. Klicken Sie aufEC2-DashboardKlicken Sie auf und danach aufElastische IP-Adressenaus.
c. Wählen Sie die Elastic-IP-Adresse aus, die Sie freigeben möchten.
d. Wählen Sie unter Actions (Aktionen) die Option Release Elastic IP addresses (Elastic-IP-Adressenfreigeben) aus.
e. Wählen Sie auf der Bestätigungsseite Freigeben.

Kontingente und Beschränkungen für Amazon Aurora

Im Folgenden finden Sie eine Beschreibung der Ressourcenkontingente und Benennungseinschränkungen für Amazon Aurora.

Themen

- Kontingente in Amazon Aurora
- Benennungseinschränkungen in Amazon Aurora
- Amazon Aurora-Größenbeschränkungen

Kontingente in Amazon Aurora

Jedes AWS Konto hat für jede AWS Region Kontingente für die Anzahl der Amazon Aurora Aurora-Ressourcen, die erstellt werden können. Nachdem das Kontingent für eine Ressource erreicht wurde, schlagen zusätzliche Aufrufe zum Erstellen dieser Ressource mit einer Ausnahme fehl.

In der folgenden Tabelle sind die Ressourcen und ihre Kontingente pro AWS Region aufgeführt.

Name	Standard	Anpas	Description
Autorisierungen pro DB-Sicherheitsgruppe	Jede unterstützte Region: 20	Nein	Anzahl der Sicherheitsgruppenberechtigungen pro DB-Sicherheitsgruppe
Benutzerdefinierte Endpunkte pro DB-Cluster	Jede unterstützte Region: 5	Yes (Ja)	Die maximale Anzahl von benutzerdefinierten Endpunkten, die Sie pro Aurora-DB-Cluster in diesem Konto in der aktuellen Region erstellen können. Dieser Wert spiegelt die höchste Anzahl von benutzerdefinierten Endpunkten in einem DB-Cluster im Konto wider. Andere

Name	Standard	Anpas	Description
			DB-Cluster im Konto haben möglicherweise eine geringere Anzahl von benutzerdefinierten Endpunkten.
Kundenspezifische Motorversionen	Jede unterstützte Region: 40	Ja	Die maximale Anzahl von benutzerdefinierte n Engine-Versionen, die in diesem Konto in der aktuellen Region zulässig sind
DB-Cluster-Parametergruppen	Jede unterstützte Region: 50	Ja	Die maximale Anzahl von DB-Cluster-Paramet ergruppen
DB-Cluster	Jede unterstützte Region: 40	Ja	Die maximale Anzahl von Aurora-Clustern in diesem Konto in der aktuellen Region
DB-Instances	ap-south-1: 20 Jede der anderen unterstützten Regionen: 40	Ja	Die maximale Anzahl von DB-Instances, die in diesem Konto in der aktuellen Region zulässig ist
DB-Shard-Gruppen	Jede unterstützte Region: 5	Yes (Ja)	Die maximale Anzahl von DB-Shard-Gruppen für Aurora Limitless Database in diesem Konto in der aktuellen Region

Name	Standard	Anpas	Description
DB-Subnetzgruppen	ap-south-1: 20 Jede der anderen unterstützten Regionen: 50	Ja	Die maximale Anzahl von DB-Subnetzgruppen
Größe des HTTP-Anforderungstexts der Daten-API	Jede unterstützte Region: 4 Megabyte	Nein	Die maximal zulässige Größe für den HTTP-Anforderungstext.
Maximale Anzahl gleichzeitiger Cluster-Geheimnis-Paare der Daten-API	Jede unterstützte Region: 30	Nein	Die maximale Anzahl eindeutiger Paare von Aurora Serverless v1-DB-Clustern und Geheimnissen in gleichzeitigen Daten-API-Anfragen für dieses Konto in der aktuellen AWS Region.
Maximale Anzahl gleichzeitiger Daten-API-Anforderungen	Jede unterstützte Region: 500	Nein	Die maximale Anzahl von Daten-API-Anforderungen an einen DB-Cluster von Aurora Serverless v1, die dasselbe Secret verwenden und gleichzeitig verarbeitet werden können. Zusätzliche Anforderungen werden in die Warteschlange gestellt und verarbeitet, sobald in Bearbeitung befindliche Anforderungen abgeschlossen sind.

Name	Standard	Anpas	Description
Maximale Ergebnissatzgröße der Daten-API	Jede unterstützte Region: 1 Megabyte	Nein	Die maximale Größe der Datenbank-Ergebnismenge, die von der Daten-API zurückgegeben werden kann.
Maximale Daten-API-Größe der JSON-Antwortzeichenfolge	Jede unterstützte Region: 10 Megabyte	Nein	Die maximale Größe der vereinfachten JSON-Antwortzeichenfolge, die von der RDS-Daten-API zurückgegeben wird.
Daten-API-Anforderungen pro Sekunde	Jede unterstützte Region: 1000 pro Sekunde	Nein	Die maximale Anzahl von Anfragen an die Daten-API pro Sekunde, die für dieses Konto in der aktuellen AWS Region zulässig ist. Dieses Kontingent gilt nur für Cluster von Amazon Aurora Serverless v1.
Ereignisabonnements	Jede unterstützte Region: 20	Ja	Die maximale Anzahl von Ereignisabonnements
IAM-Rollen pro DB-Cluster	Jede unterstützte Region: 5	Yes (Ja)	Die maximale Anzahl von IAM-Rollen, die mit einem DB-Cluster verknüpft werden können
IAM-Rollen pro DB-Instance	Jede unterstützte Region: 5	Yes (Ja)	Die maximale Anzahl von IAM-Rollen, die mit einer DB-Instance verknüpft werden können

Name	Standard	Anpas	Description
Integrationen	Jede unterstützte Region: 100	Nein	Die maximale Anzahl von Integrationen, die in diesem Konto in der aktuellen Region zulässig sind AWS
Manuelle DB-Cluster-Snapshots	Jede unterstützte Region: 100	Yes (Ja)	Die maximale Anzahl manueller DB-Cluster-Snapshots
Manuelle DB-Instance-Snapshots	Jede unterstützte Region: 100	Yes (Ja)	Die maximale Anzahl manueller DB-Instance-Snapshots
Optionsgruppen	Jede unterstützte Region: 20	Ja	Die maximale Anzahl von Optionsgruppen
Parametergruppen	ap-south-1: 20 Jede der anderen unterstützten Regionen: 50	Ja	Die maximale Anzahl von Parametergruppen
Proxys	Jede unterstützte Region: 20	Ja	Die maximale Anzahl von Proxys, die für dieses Konto in der aktuellen Region zulässig sind AWS
Read Replicas pro Primary	Jede unterstützte Region: 15	Ja	Die maximale Anzahl von Lesereplikaten pro primäre DB-Instance. Dieses Kontingent kann für Amazon Aurora nicht angepasst werden.

Name	Standard	Anpas	Description
Reservierte DB-Instances	ap-south-1: 20 Jede der anderen unterstützten Regionen: 40	Ja	Die maximale Anzahl reservierter DB-Instances, die für dieses Konto in der aktuellen AWS Region zulässig sind
Sicherheitsgruppen	ap-south-1: 20 Jede der anderen unterstützten Regionen: 25	Ja	Die maximale Anzahl von DB-Sicherheitsgruppen
Subnetze pro DB-Subnetzgruppe	Jede unterstützte Region: 20	Nein	Die maximale Anzahl von Subnetzen pro DB-Subnetzgruppe
Gesamtspeicher für alle DB-Instances	Jede unterstützte Region: 100 000	Ja	Der maximale Gesamtspeicher (in GB) von EBS-Volumes für alle Amazon-RDS-DB-Instances zusammen. Dieses Kontingent gilt nicht für Amazon Aurora, das ein maximales Cluster-Volumen von 128 TiB für jeden DB-Cluster hat.

Note

Standardmäßig ist das Verwenden von bis zu insgesamt 40 DB-Instances möglich. RDS-DB-Instances, Aurora DB-Instances, Amazon Neptune-Instances und Amazon DocumentDB-Instances werden zu diesem Kontingent hinzugerechnet.
Wenn Ihre Anwendung mehr DB-Instances benötigt, können Sie zusätzliche DB-Instances anfordern, indem Sie die Konsole für Service Quotas öffnen. Wählen Sie im Navigationsbereich AWS -Services aus. Wählen Sie Amazon Relational Database Service

(Amazon RDS) sowie ein Kontingent aus und folgen Sie den Anweisungen, um eine Kontingenterhöhung anzufordern. Weitere Informationen finden Sie unter Beantragen einer Kontingenterhöhung im Service Quotas-Benutzerhandbuch.
Für RDS für Oracle können Sie bis zu 15 Lesereplikate pro Quell-DB-Instance in jeder Region erstellen. Wir empfehlen jedoch, die Zahl der Replikate auf 5 zu beschränken, um die Replikationsverzögerung zu minimieren.
Von verwaltete Backups AWS Backup gelten als manuelle DB-Cluster-Snapshots, werden aber nicht auf das Kontingent für manuelle Cluster-Snapshots angerechnet. Informationen dazu AWS Backup finden Sie im AWS Backup Entwicklerhandbuch.

Wenn Sie RDS-API-Operationen verwenden und das Standardkontingent für die Anzahl von Aufrufen pro Sekunde überschreiten, gibt die Amazon-RDS-API eine Fehlermeldung ähnlich der folgenden aus.

ClientError: Beim Aufrufen des *API_name* Vorgangs ist ein Fehler aufgetreten (ThrottlingException): Rate überschritten.

Reduzieren Sie in diesem Fall die Anzahl der Aufrufe pro Sekunde. Das Kontingent soll die meisten Anwendungsfälle abdecken. Werden höhere Kontingente benötigt, können Sie mit einer der folgenden Optionen eine Kontingenterhöhung anfordern:

- Über die Konsole: Öffnen Sie die Konsole „Service Quotas“.
- Verwenden Sie von der AWS CLI aus den request-service-quota-increase AWS CLI Befehl.

Weitere Informationen zu diesem Service finden Sie im Benutzerhandbuch für Service Quotas.

Benennungseinschränkungen in Amazon Aurora

Die Benennungseinschränkungen in Amazon Aurora gelten wie folgt:

- DB-Cluster-ID:
  - Sie müssen 1–63 alphanumerische Zeichen oder Bindestriche enthalten.
  - Muss mit einem Buchstaben beginnen.
  - Darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.
  - Muss für alle DB-Instances pro AWS Konto und AWS Region eindeutig sein.

- Anfänglicher Datenbankname: In Aurora MySQL und Aurora PostgreSQL gelten unterschiedliche Beschränkungen für Datenbanknamen. Weitere Informationen finden Sie in den verfügbaren Einstellungen beim Erstellen jedes DB-Clusters.
- Hauptbenutzername: Für jede Datenbank-Engine gelten unterschiedliche Einschränkungen in Bezug auf den Hauptbenutzernamen. Weitere Informationen finden Sie in den verfügbaren Einstellungen beim Erstellen des DB-Clusters.
- Hauptpasswort:
  - Das Passwort für den Master-Benutzer der Datenbank kann jedes druckbare ASCII-Zeichen außer `/`, `'`, `"`, `@` oder Leerzeichen enthalten.
  - Das Passwort kann die folgende Anzahl druckbarer ASCII-Zeichen enthalten, abhängig von der DB-Engine:
    - Aurora MySQL 8–41
    - Aurora PostgreSQL 8–99
- DB-Parametergruppe:
  - Sie müssen 1–255 alphanumerische Zeichen enthalten.
  - Muss mit einem Buchstaben beginnen.
  - Bindestriche sind erlaubt, aber der Name darf nicht mit einem Bindestrich enden oder zwei aufeinanderfolgende Bindestriche enthalten.
- DB-Subnetzgruppe:
  - Müssen 1–255 Zeichen enthalten.
  - Alphanumerische Zeichen, Leerzeichen, Bindestriche, Unterstriche und Punkte sind erlaubt.

Amazon Aurora-Größenbeschränkungen

Beschränkungen der Speichergröße

Die maximale Größe des Aurora-Clusters variiert je nach Engine-Version:

Maximal 256 TiB:

- Aurora-PostgreSQL-Versionen:
  - 17.5 und alle höheren Versionen
  - 16.9 und höher
  - 15.13 und höher
- Aurora MySQL Version 3.10 (kompatibel mit MySQL 8.0.42) und höher

Maximal 128 TiB:

- Alle früheren Aurora-PostgreSQL-Versionen
- Alle verfügbaren Aurora-MySQL-3-Versionen; Aurora-MySQL-Version 2, Versionen 2.09 und höher

Weitere Informationen zur automatischen Speicherskalierung finden Sie unter Wie sich die Größe des Aurora-Speichers automatisch ändert.

Sie können die `AuroraVolumeBytesLeftTotal`-Metrik verwenden, um den verbleibenden Speicherplatz zu überwachen. Weitere Informationen finden Sie unter Metriken auf Cluster-Ebene für Amazon Aurora.

Beschränkungen der SQL-Tabellengröße

Bei einem Aurora MySQL-DB-Cluster beträgt die maximale Tabellengröße 64 Tebibyte (TiB). Bei einem Aurora PostgreSQL-DB-Cluster beträgt die maximale Tabellengröße 32 Tebibyte (TiB). Wir empfehlen Ihnen, beim Designen der Tabellen die bewährten Methoden zu befolgen, z. B. bei der Partitionierung von großen Tabellen.

Grenzen der Tabellenraum-ID

Die maximale Tabellenraum-ID für Aurora MySQL ist 2147483647. Wenn Sie häufig Tabellen erstellen und löschen, achten Sie darauf, Ihren Tabellenraum zu berücksichtigen IDs und logische Dumps zu verwenden. Weitere Informationen finden Sie unter Logische Migration von MySQL zu Amazon Aurora MySQL mithilfe von mysqldump.

Fehlerbehebung für Amazon Aurora

Verwenden Sie die folgenden Abschnitte, um Probleme zu beheben, die Sie mit DB-Instances in Amazon RDS und Amazon Aurora haben.

Themen

- Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden
- Amazon-RDS-Sicherheitsprobleme
- Zurücksetzen des Besitzerpassworts der DB-Instance
- Ausfall oder Neustart einer Amazon-RDS-DB-Instance
- Amazon-RDS-DB-Parameter Änderungen wirken sich nicht aus
- Probleme mit freisetzbarem Speicher in Amazon Aurora
- Replikationsprobleme mit Amazon Aurora MySQL

Informationen über das Beheben von Problemen mithilfe der Amazon-RDS-API finden Sie unter Fehlerbehebung für Anwendungen in Aurora.

Verbindung zur Amazon-RDS-DB-Instance kann nicht hergestellt werden

Wenn Sie keine Verbindung zu einer DB-Instance herstellen können, sind die folgenden Punkte häufige Ursachen:

- Regeln für eingehenden Datenverkehr – Die von Ihrer lokalen Firewall erzwungenen Zugriffsregeln und die für den Zugriff auf Ihre DB-Instance autorisierten IP-Adressen stimmen möglicherweise nicht überein. Das Problem sind höchstwahrscheinlich die Regeln für eingehenden Datenverkehr in Ihrer Sicherheitsgruppe.

  Standardmäßig erlauben DB-Instances keinen Zugriff. Zugriff wird über eine Sicherheitsgruppe gewährt, die der VPC zugeordnet ist und Datenverkehr in die und aus der DB-Instance zulässt. Fügen Sie der Sicherheitsgruppe bei Bedarf Regeln für eingehenden und ausgehenden Datenverkehr für Ihre besondere Situation hinzu. Sie können eine IP-Adresse, einen IP-Adressbereich oder eine andere VPC-Sicherheitsgruppe angeben.

> **Note**
>
> Wenn Sie eine neue Regel für eingehenden Datenverkehr hinzufügen, können Sie für Source (Quelle) die Option My IP (Meine IP) auswählen, um den Zugriff auf die DB-Instance von der in Ihrem Browser erkannten IP-Adresse zu ermöglichen.

Weitere Informationen zum Einrichten von Sicherheitsgruppen finden Sie unter Ermöglichen des Zugriffs auf den DB-Cluster in der VPC durch Erstellen einer Sicherheitsgruppe.

> **Note**
>
> Client-Verbindungen von IP-Adressen im Bereich 169.254.0.0/16 sind nicht erlaubt. Dies ist der APIPA-Bereich (Automatic Private IP Addressing), der für die Local-Link-Adressierung verwendet wird.

- Öffentliche Zugänglichkeit– Um eine Verbindung mit Ihrer DB-Instance von außerhalb der VPC herzustellen, z. B. mithilfe einer Client-Anwendung, muss der Instance eine öffentliche IP-Adresse zugewiesen sein.

  Um die Instance öffentlich zugänglich zu machen, ändern Sie sie und wählen Sie unter Public accessibility (Öffentlicher Zugriff) die Option Yes (Ja) aus. Weitere Informationen finden Sie unter Ausblenden einer DB-Clusters in einer VPC vor dem Internet.
- Port – Der Port, den Sie beim Erstellen der DB-Instance angegeben haben, kann aufgrund Ihrer lokalen Firewall-Beschränkungen nicht zum Senden oder Empfangen von Nachrichten verwendet werden. Wenden Sie sich an Ihren Netzwerkadministrator, um herauszufinden, ob Ihr Netzwerk den angegebenen Port für eingehende und ausgehende Kommunikation zulässt.
- Verfügbarkeit – Bei einer neu erstellten DB-Instance lautet ihr Status `creating`, bis die DB-Instance bereit für die Verwendung ist. Wenn sich der Status in `available` ändert, können Sie die Verbindung zur DB-Instance herstellen. Abhängig von der Größe Ihrer DB-Instance kann es bis zu 20 Minuten dauern, bevor eine Instance verfügbar ist.
- Internet-Gateway – Damit öffentlich auf eine DB-Instance zugegriffen werden kann, müssen die Subnetze in seiner DB-Subnetzgruppe über ein Internet-Gateway verfügen.

So konfigurieren Sie ein Internet-Gateway für ein Subnetz

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die Amazon-RDS-Konsole unter https://console.aws.amazon.com/rds/.
2. Wählen Sie im Navigationsbereich Databases (Datenbanken) und dann den Namen der DB-Instance aus.
3. Notieren Sie auf der Registerkarte Connectivity & security (Konnektivität und Sicherheit) die Werte der VPC-ID unter VPC und die Subnetz-ID unter Subnets (Subnetze).
4. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
5. Wählen Sie im Navigationsbereich Internet Gateways aus. Überprüfen Sie, ob Ihrer VPC ein Internet-Gateway angefügt ist. Falls nicht, wählen Sie Create Internet Gateway, um ein Internet-Gateway zu erstellen. Wählen Sie das Internet-Gateway aus, klicken Sie auf die Option Attach to VPC, und folgen Sie den Anleitungen, um das Gateway Ihrer VPC anzufügen.
6. Wählen Sie im Navigationsbereich die Option Subnets und dann Ihr Subnetz aus.
7. Überprüfen Sie, ob auf der Registerkarte Route Table eine Route mit `0.0.0.0/0` unter "Destination" und das Internet-Gateway für Ihre VPC unter „Target“ vorhanden ist.

   Wenn Sie eine Verbindung mit Ihrer Instance mithilfe der IPv6-Adresse herstellen, überprüfen Sie, dass eine Route für den gesamten IPv6-Datenverkehr (`::/0`) vorhanden ist, die zum Internet-Gateway führt. Andernfalls gehen Sie wie folgt vor:

   a. Wählen Sie die ID der Routing-Tabelle (rtb-xxxxxxxx) aus, um zur Routing-Tabelle zu gelangen.
   b. Klicken Sie auf der Registerkarte Routes (Routen) auf Edit routes (Routen bearbeiten). Wählen Sie Add route (Route hinzufügen) aus, verwenden Sie `0.0.0.0/0` als Ziel und das Internet-Gateway als Ziel.

      Wählen Sie für IPv6 Add route (Route hinzufügen) aus, verwenden Sie `::/0` als Ziel und das Internet-Gateway als Ziel.
   c. Wählen Sie Save Rules (Routen speichern) aus.

   Wenn Sie versuchen, eine Verbindung mit dem IPv6-Endpunkt herzustellen, stellen Sie außerdem sicher, dass der IPv6-Adressbereich des Clients berechtigt ist, eine Verbindung mit der DB-Instance herzustellen.

Weitere Informationen finden Sie unter Arbeiten mit einer DB-Cluster in einer VPC.

Testen der Verbindung zu einer DB-Instance

Sie können Ihre Verbindung zu einer DB-Instance mit gängigen Linux- oder Microsoft Windows-Tools testen.

Sie können die Verbindung über ein Linux- oder Unix-Terminal testen, indem Sie Folgendes eingeben. Ersetzen Sie *DB-instance-endpoint* durch den Endpunkt und *port* durch den Port Ihrer DB-Instance.

```
nc -zv DB-instance-endpoint port
```

Das folgende Beispiel zeigt einen Beispielbefehl und den Rückgabewert.

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Windows-Benutzer können Telnet verwenden, um die Verbindung zu einer DB-Instance zu testen. Telnet-Aktionen werden nur zum Testen der Verbindung unterstützt. Wenn eine Verbindung erfolgreich ist, gibt die Aktion keine Nachricht zurück. Wenn eine Verbindung nicht erfolgreich ist, erhalten Sie eine Fehlermeldung wie die folgende.

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819

  Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
 open
  connection to the host, on port 819: Connect failed
```

Wenn Telnet-Aktionen erfolgreich sind, wurde Ihre Sicherheitsgruppe ordnungsgemäß konfiguriert.

> **Note**
>
> Internet Control Message Protocol (ICMP)-Datenverkehr, einschließlich Ping, wird von Amazon RDS nicht akzeptiert.

Fehlerbehebung bei der Verbindungsauthentifizierung

In einigen Fällen können Sie eine Verbindung mit Ihrer DB-Instance herstellen, erhalten jedoch Authentifizierungsfehler. In diesen Fällen sollten Sie das Hauptbenutzerpasswort für die DB-Instance zurücksetzen. Sie können dies tun, indem Sie die RDS-Instance ändern.

Amazon-RDS-Sicherheitsprobleme

Verwenden Sie niemals die E-Mail-Adresse und das Passwort des Root-Benutzers Ihres AWS-Konto für ein Benutzerkonto, um Sicherheitsprobleme zu vermeiden. Die bewährte Methode besteht darin, Ihren Root-Benutzer zu verwenden, um Benutzer zu erstellen und diese DB-Benutzerkonten zuzuweisen. Sie können Ihren Root-Benutzer auch verwenden, um bei Bedarf andere Benutzerkonten zu erstellen.

Informationen zum Erstellen von Benutzern finden Sie unter Erstellen eines IAM-Benutzers in Ihrem AWS-Konto. Informationen zum Erstellen von Benutzern in AWS IAM Identity Center finden Sie unter Verwalten von Identitäten im IAM Identity Center.

Fehlermeldung „Failed to retrieve account attributes, certain console functions may be impaired (Fehler beim Abrufen von Kontoattributen, bestimmte Konsolenfunktionen können beeinträchtigt sein).“

Dieser Fehler kann verschiedene Ursachen haben. Es kann daran liegen, dass Ihrem Konto Berechtigungen fehlen oder Ihr Konto nicht ordnungsgemäß eingerichtet wurde. Wenn Ihr Konto neu ist, haben Sie möglicherweise nicht abgewartet, bis das Konto bereit ist. Wenn dies ein vorhandenes Konto ist, könnten Berechtigungen in Ihren Zugriffsrichtlinien fehlen, um bestimmte Aktionen auszuführen, wie z. B. das Erstellen einer DB-Instance. Um das Problem zu beheben, muss Ihr Administrator die erforderlichen Rollen für Ihr Konto bereitstellen. Weitere Informationen finden Sie in der IAM-Dokumentation.

Zurücksetzen des Besitzerpassworts der DB-Instance

Wenn Sie aus Ihrem DB--Cluster ausgesperrt werden, können Sie sich als Hauptbenutzer anmelden. Anschließend können Sie die Anmeldeinformationen für andere administrative Benutzer oder Rollen zurücksetzen. Wenn Sie sich nicht als Hauptbenutzer anmelden können, kann der AWS-Kontobesitzer das Hauptbenutzerpasswort zurücksetzen. Weitere Informationen zu den

Administratorkonten oder -rollen, die Sie möglicherweise zurücksetzen müssen, finden Sie unter Berechtigungen von Hauptbenutzerkonten.

Sie können das DB-Instance-Passwort mithilfe der Amazon-RDS-Konsole, des AWS CLI-Befehls modify-db-instance oder der API-Operation ModifyDBInstance ändern. Weitere Informationen zum Ändern einer DB-Instance in einen DB-Cluster finden Sie unter Ändern einer DB-Instance in einem DB-Cluster.

Ausfall oder Neustart einer Amazon-RDS-DB-Instance

Ein Ausfall der DB-Instance kann auftreten, wenn eine DB-Instance neu gestartet wird. Er kann auch auftreten, wenn die DB-Instance in einen Zustand versetzt wird, der den Zugriff darauf verhindert, und wenn die Datenbank neu gestartet wird. Ein Neustart kann erfolgen, wenn Sie Ihre DB-Instance manuell neu starten. Ein Neustart kann auch auftreten, wenn Sie eine DB-Instance-Einstellung ändern, für die ein Neustart erforderlich ist, um wirksam zu werden.

Ein Neustart der DB-Instance tritt auf, wenn Sie eine Einstellung ändern, für die ein Neustart erforderlich ist, oder wenn Sie einen Neustart manuell durchführen. Ein Neustart kann sofort erfolgen, wenn Sie eine Einstellung ändern und die Änderung sofort wirksam werden soll. Oder er kann während des Wartungsfensters der DB-Instance auftreten.

Ein Neustart der DB-Instance wird sofort ausgeführt, wenn eine der folgenden Situationen eintritt:

- Sie ändern den Aufbewahrungszeitraum für Backups für eine DB-Instance von 0 auf einen Wert ungleich Null oder von einem Wert ungleich Null auf 0. Anschließend legen Sie Apply Immediately (Sofort anwenden) auf `true` fest.
- Sie ändern die DB-Instance-Klasse und Apply Immediately (Direkt anwenden) ist auf `true` eingestellt.

Ein Neustart der DB-Instance tritt während des Wartungsfensters auf, wenn eine der folgenden Situationen eintritt:

- Sie ändern den Aufbewahrungszeitraum für Backups für eine DB-Instance von 0 auf einen Wert ungleich Null oder von einem Wert ungleich Null auf 0 und Apply Immediately (Direkt anwenden) ist auf `false` festgelegt.
- Sie ändern die DB-Instance-Klasse und Apply Immediately (Direkt anwenden) ist auf `false` eingestellt.

Wenn Sie einen statischen Parameter in einer DB-Parametergruppe ändern, wird die Änderung erst wirksam, wenn die der Parametergruppe zugeordnete DB-Instance neu gestartet wird. Die Änderung erfordert einen manuellen Neustart. Die DB-Instance wird während des Wartungsfensters nicht automatisch neu gestartet.

Amazon-RDS-DB-Parameter Änderungen wirken sich nicht aus

In einigen Fällen können Sie einen Parameter in einer DB-Parametergruppe ändern, aber die Änderungen werden nicht wirksam. Wenn dies der Fall ist, müssen Sie wahrscheinlich die DB-Instance, die der DB-Parametergruppe zugeordnet ist, neu starten. Wenn Sie einen dynamischen Parameter ändern, wird die Änderung sofort wirksam. Wenn Sie einen statischen Parameter ändern, wird die Änderung erst wirksam, wenn Sie die der Parametergruppe zugeordnete DB-Instance neu starten.

Sie können eine DB-Instance mithilfe der RDS-Konsole neu starten. Oder Sie können die API-Operation `RebootDBInstance` explizit aufrufen. Sie können ohne Failover neu starten, wenn sich die DB-Instance in einer Multi-AZ-Bereitstellung befindet. Da die zugeordnete DB-Instance nach der Änderung eines statischen Parameters neu gestartet werden muss, wird das Risiko einer fehlerhaften Konfiguration gesenkt, die API-Aufrufe beeinträchtigen könnte. Ein Beispiel hierfür ist der Aufruf von `ModifyDBInstance` zur Änderung der DB-Instance-Klasse. Weitere Informationen finden Sie unter Ändern von Parametern in einer DB-Parametergruppe in Amazon Aurora.

Probleme mit freisetzbarem Speicher in Amazon Aurora

Freisetzbarer Speicher ist der gesamte Random Access Memory (RAM, Arbeitsspeicher) einer DB-Instance, der der Datenbank-Engine zur Verfügung gestellt werden kann. Es ist die Summe aus dem freien Arbeitsspeicher des Betriebssystems und dem verfügbaren Puffer- und Seitencache-Speicher. Die Datenbank-Engine verwendet den größten Teil des Speichers auf dem Host, aber Betriebssystemprozesse verbrauchen ebenfalls RAM. Speicher, der derzeit der Datenbank-Engine zugewiesen oder von Betriebssystemprozessen verwendet wird, ist nicht im freisetzbaren Speicher enthalten. Wenn der Datenbank-Engine der Speicher ausgeht, kann die DB-Instance den temporären Speicherplatz nutzen, der normalerweise zum Puffern und Zwischenspeichern verwendet wird. Wie bereits erwähnt, ist dieser temporäre Speicherplatz im freisetzbaren Speicher enthalten.

Sie verwenden die Metrik `FreeableMemory` in Amazon CloudWatch zur Überwachung des freisetzbaren Speichers. Weitere Informationen finden Sie unter Überwachungstools für Amazon Aurora.

Wenn Ihre DB-Instance ständig wenig möglichen freien Speicher hat oder Auslagerungsbereiche verwendet, erwägen Sie, auf eine größere DB-Instance-Klasse hochzuskalieren. Weitere Informationen finden Sie unter DB-Instance-Klassen von Amazon Aurora.

Sie können auch die Speichereinstellungen ändern. Beispiel: Bei Aurora MySQL können Sie die Größe des Parameters `innodb_buffer_pool_size` anpassen. Dieser Parameter ist standardmäßig auf 75 Prozent des physischen Speichers festgelegt. Weitere Tipps zur MySQL-Fehlerbehebung finden Sie unter Wie kann ich Probleme mit geringem freisetzbaren Speicher in einer Datenbank von Amazon RDS für MySQL beheben?

Bei Aurora Serverless v2 stellt `FreeableMemory` die Menge des nicht belegten Speichers dar, die verfügbar ist, wenn die Aurora Serverless v2-DB-Instance auf ihre maximale Kapazität skaliert wird. Möglicherweise haben Sie die Instance auf relativ niedrige Kapazität herunterskaliert, aber sie meldet immer noch einen hohen Wert für `FreeableMemory`, weil die Instance hochskalieren kann. Dieser Speicher ist derzeit nicht verfügbar, aber Sie können ihn bei Bedarf erhalten.

Für jede Aurora-Kapazitätseinheit (Aurora Capacity Unit, ACU), bei der die aktuelle Kapazität unter der maximalen Kapazität liegt, erhöht sich `FreeableMemory` ungefähr um 2 GiB. Daher nähert sich diese Metrik erst null, wenn die DB-Instance so hoch wie möglich hochskaliert ist.

Wenn sich diese Metrik dem Wert `0` nähert, ist die DB-Instance möglichst hoch skaliert. Sie nähert sich der Grenze des verfügbaren Speichers. Erwägen Sie, die maximale ACU-Einstellung für den Cluster zu erhöhen. Wenn sich diese Metrik dem Wert `0` nähert, erwägen Sie bei einer Reader-DB-Instance, dem Cluster weitere Reader-DB-Instances hinzuzufügen. Auf diese Weise können Sie den schreibgeschützten Teil der Workload auf mehrere DB-Instances verteilen und so die Speicherauslastung für jede Reader-DB-Instance reduzieren. Weitere Informationen finden Sie unter Wichtige Amazon-CloudWatch-Metriken für Aurora Serverless v2.

Bei Aurora Serverless v1 können Sie den Kapazitätsbereich so ändern, dass mehr ACUs verwendet werden. Weitere Informationen finden Sie unter Ändern eines Aurora Serverless v1-DB-Clusters.

Replikationsprobleme mit Amazon Aurora MySQL

Einige MySQL-Replikationsprobleme gelten auch für Aurora MySQL. Sie können diese Probleme diagnostizieren und beheben.

Themen

- Diagnose und Lösung bei Verzögerungen zwischen Read Replicas (Lesereplikaten)
- Diagnose und Lösung eines Fehlers bei einer MySQL Read Replica

- Fehler „Replication stopped (Replikation gestoppt)“
- Die Replikation des Read Replica kann die Metadatenstruktur nicht initialisieren

Diagnose und Lösung bei Verzögerungen zwischen Read Replicas (Lesereplikaten)

Nachdem Sie ein MySQL-Lesereplikat erstellt haben und das Replikat verfügbar ist, repliziert Amazon RDS zunächst die an der Quell-DB-Instance vorgenommen Änderungen ab dem Zeitpunkt der Operation der Lesereplikaterstellung. Während dieser Phase ist die Verzögerungszeit der Replikation für das Lesereplikat größer als 0. Sie können diese Verzögerungszeit in Amazon CloudWatch anhand der Metrik Amazon RDS `AuroraBinlogReplicaLag` überwachen.

Diese `AuroraBinlogReplicaLag`-Metrik meldet den Wert des `Seconds_Behind_Master`-Feldes des `SHOW REPLICA STATUS`-Befehls von MySQL. Weitere Informationen finden Sie unter SHOW REPLICA STATUS-Anweisung in der MySQL-Dokumentation.

Wenn die Metrik `AuroraBinlogReplicaLag` 0 erreicht, hat das Replica den Stand der Quell-DB-Instance erreicht. Wenn die `AuroraBinlogReplicaLag` Metric auf -1 zurückgeht, ist die Replikation möglicherweise nicht aktiv. Um einen Replikationsfehler zu beheben, lesen Sie Diagnose und Lösung eines Fehlers bei einer MySQL Read Replica. Ein `AuroraBinlogReplicaLag`-Wert von -1 kann auch bedeuten, dass der `Seconds_Behind_Master`-Wert nicht bestimmt werden kann oder `NULL` ist.

> **Note**
>
> Frühere Versionen von Aurora MySQL verwenden `SHOW SLAVE STATUS` anstelle von `SHOW REPLICA STATUS`. Wenn Sie Aurora MySQL Version 1 oder 2 verwenden, verwenden Sie `SHOW SLAVE STATUS`. Verwenden Sie `SHOW REPLICA STATUS` für Aurora MySQL Version 3 und höher.

Die Metrik `AuroraBinlogReplicaLag` gibt während eines Netzwerkausfalls den Wert -1 an oder wenn ein Patch während des Wartungsfensters angewendet wird. Warten Sie in diesem Fall, bis die Netzwerkverbindung wiederhergestellt ist oder die Wartung beendet ist, bevor Sie die Metrik `AuroraBinlogReplicaLag` wieder überprüfen.

Die MySQL-Lesereplikationstechnologie ist asynchron. Daher können Sie gelegentliche Erhöhungen für die `BinLogDiskUsage`-Metrik in der Quell-DB-Instance und für die

`AuroraBinlogReplicaLag`-Metrik auf dem Lesereplikat erwarten. Betrachten Sie beispielsweise eine Situation, in der ein hohes Volumen von Schreiboperationen auf die Quell-DB-Instance parallel ausgeführt wird. Gleichzeitig werden Schreiboperationen in das Lesereplikat mit einem einzelnen I/O-Thread serialisiert. Eine solche Situation kann zu einer Verzögerung zwischen der Quell-Instance und dem Lesereplikat führen.

Weitere Informationen zu Read Replicas und MySQL finden Sie in unter Replication Implementation Details in der MySQL-Dokumentation.

Sie können die Verzögerung zwischen den Updates einer Quell-DB-Instance und den nachfolgenden Updates der Lesereplikate folgendermaßen reduzieren:

- Legen Sie für die DB-Instance-Klasse des Lesereplikats eine Speichergröße fest, die mit der Größe der Quell-DB-Instance vergleichbar ist.
- Stellen Sie sicher, dass die Parametereinstellungen in den DB-Parametergruppen, die von der Quell-DB-Instance und dem Lesereplikat verwendet werden, kompatibel sind. Weitere Informationen und ein Beispiel finden Sie zum Thema der `max_allowed_packet` Parameter im nächsten Abschnitt.
- Deaktivieren Sie den Anfrage-Cache. Bei Tabellen, die häufig geändert werden, kann die Verwendung des Anfragecaches die Replikationsverzögerung erhöhen, da der Cache häufig gesperrt und aktualisiert wird. Wenn dies der Fall ist, reduzieren Sie möglicherweise die Replikationsverzögerung durch die Deaktivierung des Anfragecaches. Sie können den Anfrage-Cache deaktivieren, indem Sie den `query_cache_type parameter` in der DB-Parametergruppe für die DB-Instance auf 0 setzen. Weitere Informationen zum Abfragecache finden Sie unter Abfragecache-Konfiguration.
- Wärmen Sie den Pufferpool auf dem Lesereplikat für InnoDB für MySQL auf. Nehmen wir beispielsweise an, Sie haben eine kleine Gruppe von Tabellen, die häufig aktualisiert werden, und Sie verwenden das InnoDB- oder XTraDB-Tabellenschema. Legen Sie in diesem Fall diese Tabellen auf dem Lesereplikat ab. Dies führt dazu, dass die Datenbank-Engine die Zeilen dieser Tabellen auf dem Datenträger durchsucht und sie dann im Pufferpool zwischenspeichert. Dieser Ansatz kann Replikatverzögerung reduzieren. Es folgt ein Beispiel.

  Für Linux, macOS oder Unix:

```
PROMPT> mysqldump \
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
```

```
    -p <password> \
    database_name table1 table2 > /dev/null
```

Für Windows:

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

Diagnose und Lösung eines Fehlers bei einer MySQL Read Replica

Amazon RDS überwacht den Replikationsstatus Ihrer Lesereplikate. RDS aktualisiert das Feld Replication State (Replikationsstatus) der Lesereplikat-Instance auf `Error`, wenn die Replikation aus irgendeinem Grund angehalten wird. Die Einzelheiten des von den MySQL-Engines ausgelösten Fehlers finden Sie im Feld Replication Error (Replikationsfehler). Ereignisse, die den Status des Lesereplikats angeben, werden ebenfalls generiert, einschließlich RDS-EVENT-0045, RDS-EVENT-0046 und RDS-EVENT-0057. Weitere Informationen über Ereignisse und Abonnements zu Ereignissen finden Sie unter Arbeiten mit Amazon-RDS-Ereignisbenachrichtigungen. Wenn eine MySQL-Fehlermeldung zurückgegeben wird, lesen Sie den Fehler in der MySQL Fehlermeldungsdokumentation nach.

Die folgenden, allgemeinen Situationen können häufig zu Replikationsfehlern führen:

- Der Wert für den `max_allowed_packet`-Parameter für ein Lesereplikat ist niedriger als der `max_allowed_packet`-Parameter für die Quell-DB-Instance.

  Der `max_allowed_packet`-Parameter ist ein benutzerdefinierter Parameter, den Sie in einer DB-Parametergruppe festlegen können. Der `max_allowed_packet`-Parameter wird verwendet, um die maximale Größe der Data Manipulation Language (DML) anzugeben, die in der Datenbank ausgeführt werden kann. In einigen Fällen ist der `max_allowed_packet`-Wert für die Quell-DB-Instance möglicherweise größer als der `max_allowed_packet`-Wert des Lesereplikats. In diesen Fällen kann der Replikationsprozess einen Fehler ausgeben und die Replikation anhalten. Der häufigste Fehler ist `packet bigger than 'max_allowed_packet' bytes`. Sie können den Fehler beheben, indem Sie die DB-Parametergruppe der Quelle und des Lesereplikats mit denselben Parameterwerten für `max_allowed_packet` verwenden.

- Schreibvorgänge auf Tabellen in einem Lesereplikat. Wenn Sie Indizes für ein Lesereplikat erstellen, müssen Sie den `read_only`-Parameter auf 0 setzen, um die Indizes zu erstellen. Wenn Sie in Tabellen auf dem Lesereplikat schreiben, kann die Replikation unterbrochen werden.
- Die Verwendung einer nicht-transaktionalen Speicher-Engine wie MyISAM: Read Replicas erfordern eine transaktionale Speicher-Engine. Die Replikation wird nur für die folgenden Speicher-Engines unterstützt: InnoDB for MySQL oder MariaDB.

  Führen Sie zum Umwandeln einer MyISAM-Tabelle in eine InnoDB-Tabelle den folgenden Befehl aus:

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- Verwenden von nicht-deterministischen Abfragen wie `SYSDATE()`. Weitere Informationen finden Sie unter Determination of Safe and Unsafe Statements in Binary Logging in der MySQL-Dokumentation.

Mit den folgenden Schritten können Sie Ihren Replikationsfehler beheben:

- Wenn Sie einen logischen Fehler feststellen und den Fehler sicher überspringen können, befolgen Sie die Schritte in Überspringen von Fehlern für die aktuelle Replikation. Ihre Aurora MySQL-DB-Instance muss eine Version ausführen, die das `mysql_rds_skip_repl_error`-Verfahren umfasst. Weitere Informationen finden Sie unter mysql_rds_skip_repl_error.
- Wenn ein Problem mit der Position des Binärprotokolls (Binlog) auftritt, können Sie die Replikatswiedergabeposition ändern. Bei Aurora-MySQL-Version 1 und 2 verwenden Sie dazu den Befehl `mysql.rds_next_master_log`. Bei Aurora-MySQL-Version 3 und höher verwenden Sie dazu den Befehl `mysql.rds_next_source_log`. Auf Ihrer Aurora MySQL-DB-Instance muss eine Version ausgeführt werden, die den -Befehl zum Ändern der Replikatswiedergabeposition unterstützt. Weitere Versionsinformationen finden Sie unter mysql_rds_next_master_log.
- Wenn aufgrund einer hohen DML-Last ein temporäres Leistungsproblem auftritt, können Sie den `innodb_flush_log_at_trx_commit`-Parameter in der DB-Parametergruppe auf dem Lesereplikat auf 2 stellen. Dies kann das Aufholen des Lesereplikats unterstützen, auch wenn es vorübergehend die Atomizität, die Konsistenz, die Isolation und die Haltbarkeit (Atomicity, Consistency, Isolation und Durability – ACID) verringert.
- Sie können das Lesereplikat löschen und eine Instance mit derselben DB-Instance-Kennung erstellen. Auf diese Weise bleibt der Endpunkt derselbe wie der Ihres alten Lesereplikats.

Wenn ein Replikationsfehler korrigiert wird, ändert sich der Wert unter Replication State (Replikationsstatus) zu Replicating (Replizierend). Weitere Informationen finden Sie unter Fehlerbehebung für ein Problem mit einem MySQL-Read Replica.

Fehler „Replication stopped (Replikation gestoppt)“

Wenn Sie den `mysql.rds_skip_repl_error` Befehl aufrufen, wird möglicherweise eine Fehlermeldung angezeigt, die besagt, dass die Replikation heruntergefahren oder deaktiviert ist.

Diese Fehlermeldung wird angezeigt, wenn die Replikation angehalten wird und nicht mehr neu gestartet werden kann.

Wenn Sie eine größere Anzahl von Fehlern ignorieren müssen, kann die Dauer der Verzögerung der Replikation den standardmäßigen Aufbewahrungszeitraum für binäre Protokolldateien überschreiten. In diesem Fall kann es zu einem schwerwiegenden Fehler kommen, weil binäre Protokolldateien bereinigt werden, bevor ihr Inhalt im Replikat repliziert wurde. Diese Bereinigung führt zur Beendigung der Replikation, und Sie können den Befehl `mysql.rds_skip_repl_error` nicht mehr aufrufen, um Replikationsfehler zu überspringen und zu ignorieren.

Sie können dieses Problem umgehen, indem Sie die Anzahl der Stunden erhöhen, für die binäre Protokolldateien in der Replikationsquelle beibehalten werden. Nachdem Sie die Aufbewahrungsdauer für binäre Protokolldateien verlängert haben, können Sie die Replikation neu starten und nach Bedarf den Befehl `mysql.rds_skip_repl_error` aufrufen.

Um den Aufbewahrungszeitraum für Binärprotokolle festzulegen, verwenden Sie das Verfahren mysql_rds_set_configuration. Geben Sie einen Konfigurationsparameter namens "binlog retention hours" und einen zugehörigen Wert in Stunden an, um festzulegen, wie viele Stunden binäre Protokolldateien auf dem DB-Cluster vorgehalten werden sollen (maximal 2160 Stunden = 90 Tage). Der Standardwert für Aurora MySQL ist 24 (1 Tag). Beim folgenden Beispiel wird die Aufbewahrungszeit für binäre Protokolle auf 48 Stunden festgelegt.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

Die Replikation des Read Replica kann die Metadatenstruktur nicht initialisieren

Beim Versuch, die Replikation zu starten, haben Sie die folgende Fehlermeldung erhalten:

```
Read Replica Replication Error - SQLError: 13124, reason: Replica failed to initialize
 applier metadata structure from the repository
```

Dieser Fehler tritt auf, wenn ein Problem mit der Metadatenstruktur des Replikats vorliegt. Um die Metadatenstruktur zu korrigieren, müssen Sie ein neues Replikat erstellen.

Führen Sie eine der folgenden Aktionen durch, um dies in Zukunft zu verhindern:

- Deaktivieren Sie nach Möglichkeit Multithreading auf Ihren Replikaten. Ab MySQL 8.0.27 ist Multithreading standardmäßig aktiviert.
- Wenn Sie Multithreading für Ihre Replikate verwenden müssen, empfehlen wir Ihnen, die GTID-basierte Replikation zu verwenden. Weitere Informationen finden Sie unter Verwenden der GTID-basierten Replikation.

Amazon RDS-API-Referenz

Zusätzlich zur AWS-Managementkonsole und zur AWS Command Line Interface (AWS CLI) bietet Amazon RDS auch eine API. Mithilfe der API können Sie Aufgaben zur Verwaltung Ihrer DB-Instances und anderer Objekte in Amazon RDS automatisieren.

- Eine alphabetische Liste der API-Operationen finden Sie unter Aktionen.
- Eine alphabetische Liste der Datentypen finden Sie unter Datentypen.
- Eine Liste der häufigen Abfrageparameter finden Sie unter Häufige Parameter.
- Beschreibungen der Fehlercodes finden Sie unter Häufige Fehler.

Weitere Informationen zur AWS CLI finden Sie in der AWS Command Line Interface-Amazon-RDS-Referenz.

Themen

- Verwenden der Abfrage-API
- Fehlerbehebung für Anwendungen in Aurora

Verwenden der Abfrage-API

In den folgenden Abschnitten werden die Parameter und die Abfrageauthentifizierung beschrieben, die für die Abfrage-API verwendet werden.

Allgemeine Informationen zur Funktionsweise der Abfrage-API finden Sie unter Abfrageanforderungen im Amazon EC2 API Reference.

Abfrageparameter

HTTP-Query-basierte Anfragen sind HTTP-Anfragen, die das HTTP-Verb GET oder POST und einen Query-Parameter namens verwende `Action`.

Jede Query-Anfrage muss einige allgemeine Parameter enthalten, um die Authentifizierung und Auswahl einer Aktion zu bearbeiten.

Einige Operationen verwenden Parameterlisten. Diese Listen werden mit der Notation `param.`*n* definiert. Werte von *n* sind Ganzzahlen ab 1.

Informationen zu Amazon-RDS-Regionen und -Endpunkten finden Sie unter Amazon Relational Database Service (RDS) im Abschnitt zu Regionen und Endpunkten der Allgemeine Amazon Web Services-Referenz.

Authentifizierung von Abfrageanforderungen

Sie können Query-Anfragen nur über HTTPS senden und müssen in jede Query-Anfrage eine Signatur einschließen. Sie müssen entweder AWS-Signature Version 4 oder -Signature Version 2 verwenden. Weitere Informationen finden Sie unter Signaturprozess mit Signature Version 4 und Signaturprozess mit Signature Version 2.

Fehlerbehebung für Anwendungen in Aurora

Amazon RDS stellt spezifische und beschreibende Fehlermeldungen bereit, um Sie bei der Behebung von Problemen während der Interaktion mit der Amazon RDS-API zu unterstützen.

Themen

- Fehler bei Abrufen
- Tipps zur Problembehebung

Weitere Informationen zur Fehlerbehebung bei Amazon RDS-DB-Instances finden Sie unter Fehlerbehebung für Amazon Aurora.

Fehler bei Abrufen

In der Regel sollte Ihre Anwendung überprüfen, ob eine Anforderung einen Fehler verursacht hat, bevor Sie Zeit für die Verarbeitung von Ergebnissen aufwenden. Die einfachste Möglichkeit, herauszufinden, ob ein Fehler aufgetreten ist, besteht darin, nach einem `Error`-Knoten in der Antwort aus der Amazon RDS-API zu suchen.

Die XPath-Syntax bietet eine einfache Möglichkeit, nach einem `Error`-Knoten zu suchen. Darüber hinaus vereinfacht sie den Abruf von Fehlercode und Fehlermeldung. Der folgende Codeausschnitt verwendet Perl und das XML::XPath-Modul, um zu ermitteln, ob während einer Anfrage ein Fehler aufgetreten ist. Wenn ein Fehler aufgetreten ist, gibt der Code den ersten Fehlercode und die erste Fehlermeldung in der Antwort an.

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
```

```
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

Tipps zur Problembehebung

Die folgenden Prozesse werden empfohlen, um Probleme mit der Amazon-RDS-API zu diagnostizieren und zu beheben.

- Überprüfen Sie, ob Amazon RDS in der AWS-Region normal ausgeführt wird, indem Sie http://status.aws.amazon.com aufrufen.
- Überprüfen Sie die Struktur Ihrer Anforderung.

  Jede Amazon RDS-Operation verfügt über eine Referenzseite in der Amazon RDS-API-Referenz. Prüfen Sie nochmals, dass Sie die Parameter korrekt verwenden. Betrachten Sie die Beispielanforderungen oder Benutzerszenarien, um zu sehen, ob ähnliche Operationen ausgeführt werden, und um eine Vorstellung von möglichen Fehlern zu erhalten.
- Überprüfen Sie AWS re:Post.

  Amazon RDS besitzt ein Entwickler-Community, in der Sie nach Lösungen für Probleme suchen können, die andere Entwickler bereits hatten. Zum Anzeigen der Themen navigieren Sie zu AWS re:Post.

Dokumentverlauf

Aktuelle API-Version:2014-10-31

In der folgenden Tabelle werden wichtige Änderungen am Amazon Aurora Benutzerhandbuch beschrieben. Um Benachrichtigungen über Aktualisierungen dieser Dokumentation zu erhalten, können Sie einen RSS-Feed abonnieren. Informationen zu Amazon Relational Database Service (Amazon RDS) finden Sie im Amazon Relational Database Service Benutzerhandbuch.

> **Note**
>
> Vor dem 31. August 2018 wurde Amazon Aurora im Amazon Relational Database Service Benutzerhandbuch dokumentiert. Informationen zum früheren Dokumentverlauf von Aurora finden Sie unter Dokumentverlauf im Amazon Relational Database Service-Benutzerhandbuch.

Sie können neue Amazon Aurora Funktionen auf der Was ist neu mit Datenbank?-Seite filtern. FürProdukte wählen Sie Amazon Aurora aus. Suchen Sie dann mit Schlüsselwörtern wie **`global database`** oder **`Serverless`**.

Änderung	Beschreibung	Datum
Aurora PostgreSQL Zero-ETL-Integrationen mit Unterstützung zusätzlicher Regionen Amazon SageMaker	Aurora PostgreSQL Zero-ETL-Integrationen mit Amazon SageMaker sind jetzt in weiteren Regionen verfügbar. Weitere Informationen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Zero-ETL-Integrationen.	15. Oktober 2025
Aurora PostgreSQL Limitless Database in einer weiteren Region verfügbar AWS	Aurora PostgreSQL Limitless Database ist jetzt in der verfügbar. AWS GovCloud (US) Regions Weitere	16. September 2025

	Informationen finden Sie unter Anforderungen und Überlegungen zu Aurora PostgreSQL Limitless Database.	
Amazon RDS-Daten-API unterstützt IPv6	Die Amazon RDS Data API unterstützt jetzt IPv6 Konnektivität, sodass Sie über IPv6 Adressen eine Verbindung zu Ihren Aurora-Datenbanken herstellen können. Weitere Informationen finden Sie unter Verwenden der Daten-API.	30. August 2025
Amazon Aurora ist in der Region Asien-Pazifik (Neuseeland) verfügbar	Amazon Aurora ist jetzt in der Region Asien-Pazifik (Neuseeland) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	28. August 2025
Amazon RDS Data API in einer zusätzlichen AWS Region verfügbar	Amazon-RDS-Daten-API ist jetzt in der Region Europa (Spanien) verfügbar. Weitere Informationen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für RDS-Daten-API.	1. August 2025

Null-ETL-Integrationen in Amazon SageMaker Lakehouse sind allgemein verfügbar	Null-ETL-Integrationen machen Transaktionsdaten innerhalb von Sekunden, nachdem diese in eine DB-Instance von Aurora MySQL geschrieben wurden, in Amazon SageMaker verfügbar . Weitere Informationen finden Sie unter Arbeiten mit Aurora-Null-ETL-Integrationen.	30. Juni 2025
Amazon Aurora ist in der Region Asien-Pazifik (Taipeh) verfügbar	Amazon Aurora ist jetzt in der Region Asien-Pazifik (Taipeh) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	5. Juni 2025
End-of-life Informationen für Performance Insights	AWS hat den end-of-life Termin für Performance Insights bekannt gegeben: 30. November 2025. Nach diesem Datum unterstützt Amazon RDS die Performance-Insigh ts-Konsolenumgebung, die flexiblen Aufbewahrungsfrist en und die zugehörigen Preise nicht mehr. Wir empfehlen, Ihre DB-Cluster vor diesem Datum auf Database Insights zu aktualisieren. Weitere Informationen finden Sie unter Überwachen der DB-Last mit Performance Insights auf Amazon Aurora.	30. Mai 2025

Aurora PostgreSQL Limitless Database unterstützt Database Insights CloudWatch	Sie können CloudWatch Database Insights jetzt verwenden, um Ihre Amazon Aurora PostgreSQL Limitless Database in großem Umfang zu überwachen. Weitere Informationen finden Sie unter Amazon Aurora PostgreSQL Limitless Database with CloudWatch Database Insights überwachen.	22. Mai 2025
Anzeigen der Support-Daten für die Hauptversionen der Open-Source-Engine	Sie können jetzt Informationen zu den Start- und Enddaten für Open-Source-Engine-Hauptversionen in RDS Standard Support und RDS Extended Support anzeigen, indem Sie die AWS CLI und die RDS-API verwenden. Weitere Informationen finden Sie unter Anzeigen der Support-Daten für Engine-Versionen in Amazon RDS Extended Support.	20. Mai 2025
Null-ETL-Integrationen von Aurora PostgreSQL unterstützen zusätzliche Regionen	Null-ETL-Integrationen von Aurora PostgreSQL sind nun in zusätzlichen Regionen verfügbar. Weitere Informationen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Null-ETL-Integrationen mit Amazon Redshift.	17. Februar 2025

Amazon Aurora ist in der Region Mexiko (Zentral) verfügbar	Amazon Aurora ist jetzt in der Region Mexiko (Zentral) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	14. Januar 2025
Amazon Aurora ist in der Region Asien-Pazifik (Thailand) verfügbar	Amazon Aurora ist jetzt in der Region Asien-Pazifik (Thailand) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	7. Januar 2025

End-of-life Informationen für Aurora Serverless v1	AWS hat den end-of-life Termin bekannt gegeben fürAurora Serverless v1: 31. März 2025. Wir empfehlen dringend, alle DB-Cluster mit Aurora Serverless v1 vor diesem Datum auf Aurora Serverless v2 zu aktualisieren. Das Upgrade kann eine Änderung der Hauptversionsnummer der Datenbank-Engine beinhalten. Daher ist es wichtig, diese Umstellung vor dem Datum zu planen, zu testen und zu implementieren. end-of-life Ab dem 8. Januar 2025 können Kunden keine neuen Aurora Serverless v1 Cluster oder Instances mehr mit der AWS-Managementkonsole oder der CLI erstellen. Informationen zum Upgrade-Verfahren von Aurora Serverless v1 auf Aurora Serverless v2 finden Sie unter Upgrade von einem Aurora Serverless v1- Cluster auf Aurora Serverless v2.	6. Dezember 2024

Aurora unterstützt CloudWatch Database Insights	Sie können CloudWatch Database Insights jetzt verwenden, um Ihre Amazon Aurora Aurora-DB-Cluster in großem Umfang zu überwachen. Weitere Informationen finden Sie unter Überwachen von Amazon Aurora Aurora-Datenbanken mit CloudWatch Database Insights.	01. Dezember 2024
Amazon Aurora unterstützt die db.r8g-Instance-Klassen	Sie können jetzt die db.r8g-Instance-Klassen verwenden, um Aurora-DB-Cluster zu erstellen. Weitere Informationen finden Sie unter Aurora-DB-Instance-Klassen.	21. November 2024
Aurora Serverless v2 unterstützt automatisches Pausieren und Fortsetzen	Aurora Serverless v2DB-Instances können jetzt nach einer gewissen Zeit der Inaktivität bis auf null Aurora-Kapazitätseinheiten (ACUs) heruntergeskaliert werden. Um diese Funktion zu aktivieren, setzen Sie den Mindestkapazitätswert auf 0 ACUs. Weitere Informationen finden Sie unter Automatisches Pausieren und Fortsetzen für Aurora Serverless v2.	20. November 2024

Amazon Aurora unterstützt die automatische Migration EC2 von Datenbanken	Sie können die RDS-Konsole verwenden, um eine EC2 Datenbank zu Aurora zu migrieren. Aurora verwendet AWS Database Migration Service (AWS DMS), um Ihre EC2 Quelldatenbank zu migrieren. AWS DMS ermöglicht es Ihnen, relationale Datenbanken in Ihre AWS Cloud zu migrieren. Weitere Informationen finden Sie unter Automatische Migration von EC2 Datenbanken zu Amazon Aurora mithilfe von AWS Database Migration Service.	20. November 2024
AWS Advanced NodeJS Wrapper allgemein verfügbar	Der Advanced-NodeJS-Wrapper von Amazon Web Services (AWS) ergänzt und erweitert die Funktionen eines vorhandenen NodeJS-Treibers. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu Aurora-DB-Clustern mit den AWS Treibern.	19. November 2024
Amazon Aurora unterstützt die Instance-Klassen db.r7i	Sie können jetzt die Instance-Klassen vom Typ „db.r7i“ verwenden, um Aurora-DB-Cluster zu erstellen. Weitere Informationen finden Sie unter Aurora-DB-Instance-Klassen.	18. November 2024

Die RDS-Daten-API für Aurora MySQL unterstützt alle Performance-Insights-Ansichten	Sie können jetzt die Ansichten Top-Hosts oder Top-Anwendungen für Performance Insights verwenden, um SQL-Vorgänge zu überwachen, die mit der RDS Data API für Aurora MySQL ausgeführt wurden. Weitere Informationen zur Unterstützung von Performance Insights finden Sie unter Überwachen von RDS-Daten-API-Abfragen mit Performance Insights.	31. Oktober 2024
Aurora PostgreSQL Limitless Database	Amazon Aurora PostgreSQL Limitless Database ist eine neue, automatisierte, horizontale Skalierungsfunktion (Sharding) von Amazon Aurora. Mit Aurora PostgreSQL Limitless Database können Sie über die vorhandenen Aurora-Grenzwerte für Schreibdurchsatz und Speicherplatz hinaus skalieren, indem Sie einen Datenbank-Workload auf mehrere Aurora-Writer-Instances verteilen und gleichzeitig die Möglichkeit beibehalten, sie als eine einzige Datenbank zu verwenden. Weitere Informationen finden Sie unter Amazon Aurora PostgreSQL Limitless Database.	31. Oktober 2024

Ende der Standardunterstützung von Amazon Aurora MySQL Version	Aurora-MySQL-Version 2 erreichte am 31. Oktober 2024 das Ende des Standard-Supports. Weitere Informationen finden Sie unter Vorbereiten auf das Lebenszyklusende der mit Amazon Aurora MySQL kompatiblen Edition Version 2.	31. Oktober 2024
Amazon Aurora unterstützt Betriebssystem-Upgrades auf Cluster-Ebene	Sie können Aurora-Betriebssystem-Upgrades jetzt auf DB-Cluster-Ebene anwenden. Bei fortlaufenden Upgrades werden automatisch Upgrades auf einige wenige Reader-DB-Instances gleichzeitig angewendet, sodass die Leseverfügbarkeit erhalten bleibt. Weitere Informationen finden Sie unter Betriebssystemupdates für Aurora-DB-Cluster.	30. Oktober 2024

Globale Aurora-Cluster unterstützen Tagging	Sie können jetzt Tags zu globalen Aurora-Clustern hinzufügen. Ein globaler Aurora-Cluster ist die übergeordnete Ressource, die die primären und sekundären DB-Cluster in einer globalen Aurora-Datenbank enthält. Weitere Informationen zu Tagging finden Sie unter Taggen von Aurora- und Amazon-RDS-Ressourcen. Weitere Informationen zum Taggen mit globalen Aurora-Datenbanken finden Sie unter Taggen der Ressourcen von globalen Aurora-Datenbanken.	24. Oktober 2024
Writer-Endpunkte globaler Aurora-Datenbanken	Jede globale Aurora-Datenbank verfügt über einen Writer-Endpunkt, der automatisch von Aurora aktualisiert wird, um Anforderungen an die aktuelle Writer-Instance des primären DB-Clusters weiterzuleiten. Weitere Informationen zur Verwendung des Writer-Endpunkts zusammen mit den Umstellungs- und Failover-Funktionen von globalen Aurora-Datenbanken finden Sie unter Verwenden von Umstellung oder Failover in Amazon Aurora Global Database.	22. Oktober 2024

Null-ETL-Integrationen von Aurora PostgreSQL mit Amazon Redshift sind allgemein verfügbar	Null-ETL-Integrationen machen Transaktionsdaten innerhalb von Sekunden in Amazon Redshift verfügbar, nachdem sie in einen Quell-DB-Cluster von Aurora PostgreSQL geschrieben wurden. Das Feature ist jetzt allgemein verfügbar. Weitere Informationen finden Sie unter Arbeiten mit Null-ETL-Integrationen von Aurora in Amazon Redshift.	15. Oktober 2024
Aurora Serverless v2unterstützt 256 ACUs	Sie können jetzt Aurora Serverless v2 DB-Cluster mit einer maximalen Kapazität von 256 Aurora-Kapazitätseinheiten (ACUs) erstellen. Weitere Informationen finden Sie unter Kapazität von Aurora Serverless v2.	3. Oktober 2024

Amazon Aurora unterstützt Console-to-Code	Sie können es jetzt verwenden Console-to-Code, um Code aus Aktionen zu generiere n, die Sie in der RDS-Konso le ausführen. Der generiert e Code kann Ihnen beim Schreiben von Code helfen, um Ihre Nutzung anderer AWS Dienste zu automatis ieren. Weitere Informationen finden Sie unter Verwenden, Console-to-Code um Code für Ihre Aurora-Konsolenaktionen zu generieren.	3. Oktober 2024
RDS-Daten-API unterstützt Performance Insights	Sie können jetzt Performan ce Insights verwenden, um SQL-Vorgänge zu überwache n, die mit der RDS-Daten -API ausgeführt wurden. Weitere Informationen zur Unterstützung von Performan ce Insights finden Sie unter Überwachen von RDS-Daten-API-Abfragen mit Performance Insights.	26. September 2024

RDS-Daten-API für Aurora Serverless v2 und bereitgestellte ist für Aurora MySQL verfügbar	Die RDS-Daten-API ist jetzt für Aurora MySQL-Cluster verfügbar, die Aurora Serverless v2 oder bereitgestellte Instances verwenden. Informationen zur Nutzung der RDS-Daten-API finden Sie unter Verwenden der RDS-Daten-API. Informationen zur Aurora MySQL-Version und AWS regionalen Unterstützung für die RDS-Daten-API finden Sie unter Daten-API mit Aurora MySQL Serverless v2 und bereitgestellt.	26. September 2024
Amazon Aurora ist in der Region Asien-Pazifik (Malaysia) verfügbar	Amazon Aurora ist nun in der Region Asien-Pazifik (Malaysia) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	22. August 2024
Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat die Berechtigung `sns:Publish` aus der `AmazonRDSBetaServiceRolePolicy` der serviceverknüpften Rolle `AWSServiceRoleForRDSBeta` entfernt. Weitere Informationen finden Sie unter Amazon RDS-Updates für AWS verwaltete Richtlinien.	7. August 2024

Aktualisierung auf eine bestehende Richtlinie	Amazon RDS hat die Berechtigung `sns:Publish` aus der `AmazonRDSPreviewServiceRolePolicy` der serviceverknüpften Rolle `AWSServiceRoleForRDSPreview` entfernt. Weitere Informationen finden Sie unter Amazon RDS-Updates für AWS verwaltete Richtlinien.	7. August 2024
Aurora Serverless v2unterstützt Auto-Pause () scale-to-zero	Aurora Serverless v2DB-Instances können jetzt nach einer gewissen Zeit der Inaktivität bis auf null Aurora-Kapazitätseinheiten (ACUs) heruntergeskaliert werden. Um dieses Feature zu aktivieren, setzen Sie den Mindestkapazitätswert auf 0. Weitere Informationen finden Sie unter Verwendung von Aurora Serverless v2.	25. Juli 2024
AWS ODBC-Treiber für MySQL allgemein verfügbar	Der Amazon Web Services (AWS)-ODBC-Treiber für MySQL ist ein Client-Treiber, der für die hohe Verfügbarkeit von Aurora MySQL konzipiert wurde. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit Aurora MySQL mit dem Amazon Web Services (AWS)-ODBC-Treiber für MySQL.	18. Juli 2024

Die RDS-Daten-API für Aurora Serverless v2 ist in weiteren Regionen verfügbar	Die RDS-Daten-API ist jetzt für Aurora PostgreSQL in mehreren zusätzlichen AWS Regionen verfügbar. Weitere Informationen zur Unterstüt zung von Regionen für die RDS-Daten-API finden Sie unter Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellt.	9. Juli 2024
AWS Python-Treiber allgemein verfügbar	Der Python-Treiber für Amazon Web Services (AWS) ist als fortschrittlicher Python-Wrapper konzipiert. Dieser Wrapper ergänzt den Open-Source-Treiber Psycopg und erweitert dessen Funktione n. Weitere Informationen finden Sie unter Herstellen einer Verbindung zu Aurora-DB-Clustern mit den AWS Treibern.	23. Mai 2024
Null-ETL-Integrationen in Regionen in China verfügbar	Zero-ETL-Integrationen sind jetzt in AWS-Regionen China (Peking) und China (Ningxia) verfügbar. Weitere Informati onen finden Sie unter Null-ETL-Integrationen in Amazon Redshift.	21. Mai 2024

RDS-Proxy in weiteren Regionen verfügbar	RDS-Proxy ist jetzt in den Regionen Asien-Pazifik (Hyderabad), Asien-Pazifik (Melbourne), Naher Osten (VAE), Israel (Tel Aviv), Kanada West (Calgary) und Europa (Zürich) verfügbar. Weitere Informationen über RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	21. Mai 2024
Amazon RDS unterstützt differenzierten Zugriff für Performance Insights	Sie können jetzt den Zugriff auf einzelne Dimensionen in Performance Insights zulassen oder verweigern. Dieser differenzierte Zugriff kann für `GetResourceMetrics` -, `DescribeDimensionKeys` - und `GetDimensionKeyDetails` -Aktionen verwendet werden. Weitere Informationen finden Sie unter https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_PerfInsights.access-control.dimensionAccess-policy.html.	21. Mai 2024

Amazon RDS Extended Support	Beim Erstellen oder Wiederherstellen einer Datenbank von Aurora-MySQL-Version 2 oder Aurora-PostgreSQL-Version 11 wird diese Datenbank jetzt automatisch für Amazon RDS Extended Support registriert, sodass Ihre vorhandenen Anwendungen weiterhin wie gewohnt so funktionieren. Sie können sich nach Ablauf des Standard-Supports von Aurora für Ihre Datenbank-Engine vom RDS Extended Support abmelden, um Gebühren zu vermeiden. Weitere Informationen finden Sie unter Verwenden von Amazon RDS Extended Support.	21. März 2024
Datenfilterung für Null-ETL-Integrationen	Amazon RDS unterstützt Datenfilterung auf Datenbank- und Tabellenebene für Null-ETL-Integrationen in Amazon Redshift. Weitere Informationen finden Sie unter Datenfilterung für Null-ETL-Integrationen von Aurora in Amazon Redshift.	20. März 2024

Aurora-MySQL-Integrationen in Amazon Bedrock	Sie können jetzt MySQL-Datenbanken von Amazon Aurora in Amazon Bedrock integrieren, um Anwendungen mit generativer KI zu unterstützen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Machine Learning mit Aurora MySQL.	8. März 2024
AWS Neue verwaltete Richtlinie	Amazon RDS hat eine neue verwaltete Richtlinie hinzugefügt `AmazonRDSCustomInstanceProfileRolePolicy`, mit der RDS Custom Automatisierungsaktionen und Datenbankverwaltungsaufgaben über ein EC2 Instance-Profil ausführen kann. Weitere Informationen finden Sie unter Änderungen von Amazon RDS an von AWS verwalteten Richtlinien.	27. Februar 2024
Amazon RDS-Unterstützung für AWS Secrets Manager die Region Israel (Tel Aviv)	Amazon RDS unterstützt Secrets Manager in der Region Israel (Tel Aviv). Weitere Informationen finden Sie unter Passwortverwaltung mit Amazon RDS und AWS Secrets Manager.	21. Februar 2024

Amazon RDS Extended Support	Amazon RDS aktiviert jetzt automatisch den Amazon RDS Extended Support, wenn die Engine-Hauptversionen von Aurora MySQL und Aurora PostgreSQL in Ihren DB-Instances und DB-Cluste rn mit Multi-AZ das Ende des Standard-Supports von RDS erreichen. Weitere Informationen finden Sie unter Verwenden von Amazon RDS Extended Support.	15. Februar 2024
Aurora PostgreSQL 16.1 unterstützt Babelfish für Aurora PostgreSQL 4.0.0	Aurora PostgreSQL 16.1 unterstützt Babelfish 4.0.0. Eine Liste der neuen Funktionen finden Sie unter 16.1. Eine Liste der Funktionen, die in jeder Babelfish-Version unterstüt zt werden, finden Sie unter Unterstützte Funktionalität in Babelfish nach Version. Weitere Nutzungsinformatio nen finden Sie unter Arbeiten mit Babelfish für Aurora PostgreSQL.	31. Januar 2024
Aktualisieren auf das Standard-CA-Zertifikat	Das Standard-CA-Zertifikat ist auf `rds-ca-rsa2048-g1` festgelegt. Weitere Informationen finden Sie unter Verwenden, um eine Verbindung SSL/TLS zu einem DB-Cluster zu verschlüsseln.	26. Januar 2024

RDS-Proxy ist in der Region Europa (Spanien) verfügbar	RDS-Proxy ist jetzt in der Region Europa (Spanien) verfügbar. Weitere Informationen über RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	8. Januar 2024
RDS-Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellt	Sie können jetzt die RDS-Daten-API mit Aurora PostgreSQL Serverless v2 und bereitgestellten DB-Clustern verwenden. Mit der RDS Data API können Sie über einen sicheren HTTP-Endpunkt auf Ihre Aurora-Cluster zugreifen und SQL-Anweisungen ausführen, ohne Datenbanktreiber zu verwenden oder Verbindungen zu verwalten. Weitere Informationen finden Sie unter Verwenden der RDS-Daten-API.	21. Dezember 2023
Aurora PostgreSQL-Integrationen in Amazon Bedrock	Sie können jetzt Datenbanken von Amazon Aurora PostgreSQL in Amazon Bedrock integrieren, um Anwendungen mit generativer KI zu unterstützen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora Machine Learning mit Aurora PostgreSQL.	21. Dezember 2023

Amazon Aurora ist in der Region Kanada West (Calgary) verfügbar	Amazon Aurora ist jetzt in der Region Kanada West (Calgary) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	20. Dezember 2023
Amazon RDS unterstützt das Anzeigen und Beantworten von Empfehlungen	Amazon-Aurora-Empf ehlungen umfassen jetzt schwellenwertbasierte proaktive und Machine-L earning-basierte reaktive Empfehlungen. Weitere Informationen finden Sie unter Empfehlungen für Amazon Aurora.	19. Dezember 2023
Null-ETL-Integrationen von Amazon PostgreSQL in Amazon Redshift (Vorschau)	Sie können jetzt Null-ETL-Integrationen in Amazon Redshift mithilfe eines Quell-DB-Clusters von Aurora PostgreSQL erstellen. Für die Vorschauversion müssen Sie alle Integrationen in der Amazon RDS Database Preview Environment im Osten der USA (Ohio) (us-east-2) erstellen. AWS-Region Weitere Informationen finden Sie unter Arbeiten mit Null-ETL-Integrationen von Aurora in Amazon Redshift.	28. November 2023

Amazon Aurora PostgreSQL unterstützt die globale Weiterleitung von Datenbank-Schreibvorgängen	Sie können jetzt die Schreibweiterleitung auf sekundären Clustern in einer globalen Datenbank auf der Grundlage von Aurora PostgreSQL aktivieren. Weitere Informationen finden Sie unter Verwenden der Schreibweiterleitung in einer globalen Aurora-PostgreSQL-Datenbank.	9. November 2023
Aurora-PostgreSQL-Unterstützung für optimierte Lesevorgänge	Mit Aurora-optimierten Lesevorgängen können Sie eine schnellere Abfrageverarbeitung mit Aurora PostgreSQL erreichen. Weitere Informationen finden Sie unter Verbesserung der Abfrageleistung für Aurora PostgreSQL mit Aurora-optimierten Lesevorgängen.	8. November 2023

Amazon RDS exportiert Performance Insights Insights-Metriken nach Amazon CloudWatch	Mit Performance Insights können Sie die vorkonfig urierten oder benutzerd efinierten Metrik-Dashboards nach Amazon exportieren. CloudWatch Die exportier ten Metrik-Dashboards können in der Konsole angezeigt werden. CloudWatc h Sie können auch ein ausgewähltes Performan ce Insights Insights-Metrik-Widget exportieren und die Metrikdaten in der CloudWatc h Konsole anzeigen. Weitere Informationen finden Sie unter Performance Insights Insights-Metriken exportieren nach CloudWatch.	8. November 2023
Allgemeine Verfügbarkeit von Aurora MySQL Null-ETL-Integrationen mit Amazon Redshift	Null-ETL-Integrationen mit Amazon Redshift sind jetzt allgemein für Aurora MySQL verfügbar. Weitere Informati onen finden Sie unter Arbeiten mit Null-ETL-Integrationen von Aurora in Amazon Redshift.	7. November 2023
Aurora PostgreSQL-Unterst ützung für RDS-Bereitstellung en Blue/Green	Sie können jetzt eine blue/ green Bereitstellung aus einem Aurora PostgreSQL-DB-Cluster erstellen. Weitere Informationen finden Sie unter Verwenden von Amazon Blue/ Green RDS-Bereitstellungen für Datenbank-Updates.	26. Oktober 2023

Aurora MySQL unterstützt serverseitige Verschlüsselung mit AWS KMS keys (SSE-KMS)	In Aurora MySQL Version 3.05 und höher können Sie SSE-KMS, einschließlich Von AWS verwaltete Schlüssel kundenverwalteter Schlüssel, für die serverseitige Verschlüsselung von Daten verwenden, die Sie aus Amazon S3 laden oder dort speichern. Weitere Informationen finden Sie unter Laden von Daten aus Textdateien in einem Amazon-S3-Bucket in einen Amazon-Aurora-MySQL-DB-Cluster und Speichern von Daten aus einem Amazon-Aurora-MySQL-DB-Cluster aus Textdateien in einem Amazon-S3-Bucket.	25. Oktober 2023
Aurora-MySQL-Optimierungen reduzieren die Neustartzeit der Datenbank	In Aurora MySQL Version 3.05 und höher haben wir Optimierungen eingeführt, die die Neustartzeit der Datenbank reduzieren. Diese Optimierungen sorgen für bis zu 65 % weniger Ausfallzeiten als ohne Optimierungen und weniger Unterbrechungen Ihrer Datenbank-Workloads nach einem Neustart. Weitere Informationen finden Sie unter Optimierungen zur Verkürzung der Datenbankneustartzeit.	25. Oktober 2023

Aktualisierung der verwalteten Richtlinien AWS	Die von `AmazonRDSPerformanceInsightsReadOnly` und `AmazonRDSPerformanceInsightsFullAccess` verwalteten Richtlinien enthalten jetzt `Sid` (Statement-ID) als Bezeichner in der Richtlinienerklärung. Weitere Informationen finden Sie unter Änderungen von Amazon RDS an von AWS verwalteten Richtlinien.	23. Oktober 2023
Amazon RDS veröffentlicht Performance Insights Insights-Zählermetriken auf Amazon CloudWatch	Die metrische Rechenfunktion DB_PERF_INSIGHTS in der CloudWatch Konsole ermöglicht es Ihnen, Amazon RDS nach Performance Insights Insights-Zählermetriken abzufragen. Weitere Informationen finden Sie unter CloudWatch Alarme zur Überwachung von Amazon Aurora erstellen.	20. September 2023
Amazon Aurora unterstützt point-in-time Recovery (PITR) mit AWS Backup	Sie können jetzt automatische (kontinuierliche) Aurora-Backups verwalten AWS Backup und von dort aus zu bestimmten Zeiten wiederherstellen. Weitere Informationen finden Sie unter Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit mit AWS Backup.	07. September 2023

Amazon RDS Extended Support	Amazon Aurora kündigt die bevorstehende Möglichkeit an, die Engine-Hauptversionen von Aurora MySQL und Aurora PostgreSQL in Ihren DB-Instances auch nach dem Ende des Standard-Supports von Aurora weiter auszuführen. Weitere Informationen finden Sie unter Verwenden von Amazon RDS Extended Support.	1. September 2023
Amazon Aurora MySQL erweitert die Unterstützung für Percona XtraBackup	Sie können jetzt physische Migrationen von MySQL-8.0-Datenbanken zu DB-Clustern in Aurora MySQL Version 3 durchführen. Weitere Informationen finden Sie unter Physische Migration von MySQL mithilfe von Percona XtraBackup und Amazon S3.	24. August 2023

Die globale Aurora-Datenbank unterstü.tzt globales Datenbank-Failover	Die globale Aurora-Datenbank unterstützt jetzt verwaltetes globales Failover, sodass Sie sie nach einem echten regionalen Notfall oder einem vollständigen Service-Ausfall einfacher wiederherstellen können. Weitere Informationen zu dieser Funktion finden Sie unter Ausführen von verwalteten Failovern für globale Aurora-Datenbanken .. Die Funktion, die zuvor als „verwaltetes geplantes Failover“ bezeichnet wurde, heißt jetzt „Umstellung“. Informationen zu Umstellungen finden Sie unter Durchführen von Umstellungen für Amazon Aurora Global Databases.	21. August 2023
Aktualisierung der AWS verwalteten Richtlinienberechtigungen	Die `AmazonRDSFullAccess` -verwaltete Richtlinie umfasst neue Berechtigungen, mit denen Sie den Leistungsanalysebericht für einen bestimmten Zeitraum erstellen, anzeigen und löschen können. Weitere Informationen finden Sie unter Amazon RDS-Updates für AWS verwaltete Richtlinien.	17. August 2023

Aktualisierung der AWS verwalteten Richtlinienberechtigungen	Das Hinzufügen neuer Berechtigungen zur `AmazonRDSPerformanceInsightsReadOnly` -verwalteten Richtlinie und das Hinzufügen einer neuen verwalteten Richtlinie `AmazonRDSPerformanceInsightsFullAccess` ermöglicht es Ihnen, einen DB-Lastanalysebericht für einen bestimmten Zeitraum zu erstellen. Weitere Informationen finden Sie unter Amazon RDS-Updates für AWS verwaltete Richtlinien.	16. August 2023
Amazon RDS unterstützt die Analyse von DB-Ladezeiträumen mit Performance Insights	Performance Insights ermöglicht Ihnen, einen Leistungsanalysebericht für einen bestimmten Zeitraum zu erstellen. Der Bericht enthält die identifizierten Einblicke und Empfehlungen zum Beheben von Leistungsproblemen. Weitere Informationen finden Sie unter Analysieren der DB-Last über einen bestimmten Zeitraum.	16. August 2023

Amazon Aurora unterstützt die Beibehaltung automatisierter Backups für DB-Cluster	Sie können jetzt automatisierte Backups für gelöschte Aurora-Cluster beibehalten und sie zu einem bestimmten Zeitpunkt wiederherstellen. Weitere Informationen finden Sie unter Beibehaltung automatisierter Backups.	4. August 2023
Amazon Aurora ist in der Region Israel (Tel Aviv) verfügbar	Amazon Aurora ist jetzt in der Region Israel (Tel Aviv) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	1. August 2023
Amazon Aurora MySQL unterstützt die lokale (clusterinterne) Schreibweiterleitung	Sie können jetzt Schreibvorgänge von einer Reader-DB-Instance an eine Writer-DB-Instance innerhalb eines DB-Clusters von Aurora MySQL weiterleiten. Weitere Informationen finden Sie unter Verwendung der Schreibweiterleitung in einem DB-Cluster von Amazon Aurora MySQL.	31. Juli 2023
Amazon Aurora unterstützt Aurora Serverless v2 zusätzlich AWS-Region	Sie können jetzt Aurora Serverless v2 DB-Cluster im asiatisch-pazifischen Raum (Melbourne) erstellen AWS-Region. Weitere Informationen zu Aurora Serverless v2 finden Sie unter Verwendung von Aurora Serverless v2.	28. Juni 2023

Amazon Aurora führt Null-ETL-Integrationen in Amazon Redshift ein (Vorschau)	Null-ETL-Integrationen bieten eine vollständig verwaltete Lösung, die Transaktionsdaten innerhalb von Sekunden, nachdem diese in einen Aurora-MySQL-DB-Cluster geschrieben wurden, in Amazon Redshift verfügbar macht. Weitere Informationen finden Sie unter Arbeiten mit Null-ETL-Integrationen von Aurora in Amazon Redshift.	28. Juni 2023
Amazon RDS bietet eine kombinierte Performance Insights- und CloudWatch Metrikansicht im Performance Insights Insights-Dashboard	Amazon RDS bietet jetzt eine konsolidierte Ansicht von Performance Insights und CloudWatch Metriken im Performance Insights Insights-Dashboard. Weitere Informationen finden Sie unter Anzeigen kombinierter Metriken mit dem Performance-Insights-Dashboard.	24. Mai 2023
Amazon Aurora unterstützt die Instance-Klassen db.r7g	Sie können jetzt die Instance-Klassen vom Typ „db.r7g“ verwenden, um Aurora-DB-Cluster zu erstellen. Weitere Informationen finden Sie unter Aurora-DB-Instance-Klassen.	11. Mai 2023

Amazon Aurora unterstützt eine neue Speicherkonfiguration für DB-Cluster	Mit Aurora I/O-Optimized zahlen Sie nur für die Nutzung und Speicherung Ihrer DB-Cluster, ohne zusätzliche Gebühren für Lese- und I/O Schreibvorgänge. Weitere Informationen finden Sie unter Speicherkonfigurationen für DB-Cluster von Amazon Aurora.	11. Mai 2023
Amazon Aurora unterstützt Aurora Serverless v2 zusätzlich AWS-Regionen	Sie können jetzt Aurora Serverless v2 DB-Cluster in den folgenden Ländern erstellen AWS-Regionen: Asien-Pazifik (Hyderabad), Europa (Spanien), Europa (Zürich) und Naher Osten (VAE). Weitere Informationen zu Aurora Serverless v2 finden Sie unter Verwendung von Aurora Serverless v2.	4. Mai 2023
Aurora Serverless v1 unterstützt die Konvertierung zu bereitgestellten Clustern	Sie können einen DB-Cluster von Aurora Serverless v1 direkt zu einem bereitgestellten DB-Cluster konvertieren. Weitere Informationen finden Sie unter Konvertieren eines DB-Clusters von Aurora Serverless v1 zu einem bereitgestellten Cluster.	27. April 2023

Aurora Serverless v1 unterstützt Amazon Aurora PostgreSQL Version 13	Sie können jetzt DB-Cluster von Aurora Serverless v1 erstellen, auf denen Aurora PostgreSQL Version 13 ausgeführt wird. Weitere Informationen finden Sie unter Aurora Serverless v1.	27. April 2023
Amazon Aurora Aurora-Unterstützung für AWS Secrets Manager die Regionen China	Amazon Aurora unterstützt Secrets Manager in den Regionen China (Peking) und China (Ningxia). Weitere Informationen finden Sie unter Passwortverwaltung mit Amazon Aurora und AWS Secrets Manager.	20. April 2023
Amazon Aurora unterstützt das Veröffentlichen von Ereignissen mit Tags für Themen-Subscriber	Amazon Aurora Aurora-Ereignisbenachrichtigungen, die an Amazon Simple Notification Service (Amazon SNS) oder Amazon gesendet werden, enthalten EventBridge jetzt Ereignis-Tags im Nachrichtentext. Diese Tags liefern Daten über die Ressource, die vom Serviceereignis betroffen war. Weitere Informationen finden Sie unter Tags und Attribute von Amazon-RDS-Ereignisbenachrichtigungen.	17. April 2023

Aktualisieren auf Berechtigungen für serviceverknüpfte IAM-Rollen	Die `AmazonRDSReadOnlyAccess` Richtlinien `AmazonRDSFullAccess` und gewähren jetzt zusätzliche Berechtigungen, um die Anzeige von Amazon DevOps Guru-Ergebnissen in der RDS-Konsole zu ermöglichen. Weitere Informationen finden Sie unter Amazon RDS-Updates für AWS verwaltete Richtlinien.	30. März 2023
Amazon Aurora unterstützt globale Datenbanken in der Region Asien-Pazifik (Melbourne)	Sie können jetzt globale Aurora-Datenbanken in der Region Asien-Pazifik (Melbourne) erstellen. Weitere Informationen zu globalen Aurora-Datenbanken finden Sie unter Verwenden von globalen Amazon-Aurora-Datenbanken.	22. März 2023
Aktualisierung der AWS verwalteten Richtlinienberechtigungen	Die `AmazonRDSReadOnlyAccess` Richtlinien `AmazonRDSFullAccess` und gewähren Amazon jetzt zusätzliche Berechtigungen CloudWatch. Weitere Informationen finden Sie unter Amazon RDS-Updates für AWS verwaltete Richtlinien.	16. März 2023

RDS-Proxy ist in den chinesischen Regionen verfügbar	RDS-Proxy ist jetzt in den Regionen China (Peking) und China (Ningxia) verfügbar. Weitere Informationen über RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	15. März 2023
Amazon Aurora unterstützt Aurora Serverless v2 in den chinesischen Regionen	Aurora Serverless v2 ist jetzt in den Regionen China (Peking) und China (Ningxia) verfügbar. Weitere Informationen finden Sie unter Aurora Serverless v2.	15. März 2023
RDS-Proxy ist in der Region Asien-Pazifik (Jakarta) verfügbar	RDS-Proxy ist jetzt in der Region Asien-Pazifik (Jakarta) verfügbar. Weitere Informationen über RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	08. März 2023
Amazon Aurora MySQL unterstützt die Kerberos-Authentifizierung	Sie können jetzt die Kerberos-Authentifizierung verwenden, um Benutzer zu authentifizieren, wenn sie sich mit Ihren Aurora-MySQL-DB-Clustern verbinden. Weitere Informationen finden Sie unter Verwenden der Kerberos-Authentifizierung für Aurora MySQL.	08. März 2023

Amazon Aurora unterstützt zusätzlich globale Datenbanken AWS-Regionen	Sie können jetzt globale Aurora-Datenbanken in den folgenden Regionen erstellen: Afrika (Kapstadt), Asien-Pazifik (Hongkong), Asien-Pazifik (Hyderabad), Asien-Pazifik (Jakarta), Europa (Mailand), Europa (Spanien), Europa (Zürich), Naher Osten (Bahrain) und Naher Osten (VAE). Weitere Informationen zu globalen Aurora-Datenbanken finden Sie unter Verwenden von globalen Amazon-Aurora-Datenbanken.	6. März 2023
Amazon Aurora unterstützt zusätzlich das Kopieren von DB-Cluster-Snapshots AWS-Regionen	Sie können jetzt DB-Cluster-Snapshots in den foglenden Regionen kopieren: Afrika (Kapstadt), Asien-Pazifik (Hongkong), Asien-Pazifik (Hyderabad), Asien-Pazifik (Jakarta), Asien-Pazifik (Melbourne), Europa (Mailand), Europa (Spanien), Europa (Zürich), Naher Osten (Bahrain) und Naher Osten (VAE). Informationen zum Kopieren von DB-Cluster-Snapshots finden Sie unter Kopieren eines DB-Cluster-Snapshots.	6. März 2023

Amazon DevOps Guru for RDS unterstützt proaktive Einblicke	Amazon DevOps Guru for RDS veröffentlicht proaktive Einblicke mit Empfehlungen, die Ihnen helfen, Probleme in Ihren Aurora-Datenbanken zu beheben, bevor sie voraussic htlich auftreten. Weitere Informationen finden Sie unter So funktioniert DevOps Guru for RDS.	28. Februar 2023
Amazon-Aurora-MySQL-Version 1 ist veraltet	Aurora-MySQL-Version 1 (mit MySQL 5.6 kompatibel) ist veraltet. Weitere Informationen finden Sie unter Wie lange bleiben Amazon-Aurora-Haup tversionen verfügbar?	28. Februar 2023
Aurora Serverless v1 unterstüt zt die Einstellung des DB-Cluster-Wartungsfensters	Sie können jetzt das Wartungsfenster für Aurora Serverless v1-DB-Cluster festlegen. Weitere Informati onen finden Sie unter Anpassen des bevorzugten DB-Cluster-Wartungsfensters.	27. Februar 2023
Amazon Aurora unterstützt Datenbank-Aktivitätsstreams in den Regionen Asien-Paz ifik (Hyderabad), Europa (Spanien) und Naher Osten (VAE).	Weitere Informationen finden Sie unter Datenbankaktivitäts-Streams.	27. Januar 2023

Amazon Aurora ist in der Region Asien-Pazifik (Melbourne) verfügbar	Amazon Aurora ist jetzt in der Region Asien-Pazifik (Melbourne) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	23. Januar 2023
Angeben der Zertifizierungsstelle (CA) bei der DB-Cluster-Erstellung	Sie können jetzt bei der Erstellung des DB-Clusters angeben, welche CA für das Serverzertifikat eines DB-Clusters verwendet werden soll. Weitere Informationen finden Sie unter Zertifizierungsstellen.	5. Januar 2023
Aurora MySQL 3.*-Unterstützung für die Rückverfolgung	Aurora MySQL 3.*-Versionen unterstützen jetzt die schnelle Wiederherstellung nach Benutzerfehlern wie Löschen der falschen Tabelle oder Löschen der falschen Zeile. Mithilfe de Rückverfolgung können Sie Ihre Datenbank zu einem früheren Zeitpunkt verschieben, ohne dass eine Wiederherstellung aus einem Backup erforderlich ist. Der Vorgang ist selbst bei großen Datenbanken innerhalb weniger Sekunden abgeschlossen. Einzelheiten finden Sie unter Rückverfolgen eines Aurora-DB-Clusters.	4. Januar 2023

Verwenden Sie Amazon Blue/Green RDS-Bereitstellungen, die zusätzlich verfügbar sind AWS-Regionen	Die Blue/Green Bereitstellungsfunktion ist jetzt in den Regionen China (Peking) und China (Ningxia) verfügbar. Weitere Informationen finden Sie unter Verwenden von Amazon Blue/Green RDS-Bereitstellungen für Datenbank-Updates.	22. Dezember 2022
Aktualisieren auf Berechtigungen für serviceverknüpfte IAM-Rollen	Die RDSService RolePolicy Amazon-Richtlinie gewährt jetzt zusätzliche Berechtigungen für AWS Secrets Manager. Weitere Informationen finden Sie unter Amazon RDS-Updates für AWS verwaltete Richtlinien.	22. Dezember 2022
Amazon Aurora lässt sich AWS Secrets Manager für die Passwortverwaltung integrieren	Aurora kann das Hauptbenutzerpasswort für einen DB-Cluster in Secrets Manager verwalten. Weitere Informationen finden Sie unter Passwortverwaltung mit Amazon Aurora und AWS Secrets Manager.	22. Dezember 2022
Amazon Aurora unterstützt Aurora Serverless v2 zusätzlich AWS-Regionen	Aurora Serverless v2 ist jetzt in den Regionen Afrika (Kapstadt) und Europa (Mailand) verfügbar. Weitere Informationen finden Sie unter Aurora Serverless v2.	21. Dezember 2022

Aurora PostgreSQL unterstützt RDS-Proxy mit PostgreSQL 14	Sie können jetzt einen RDS-Proxy mit einem DB-Cluster von Aurora PostgreSQL 14 erstellen. Weitere Informationen zu RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	13. Dezember 2022
Amazon Aurora informiert Sie über kürzlich von Amazon Guru entdeckte Anomalien DevOps	Auf der Seite mit den Datenbankdetails der Konsole werden Sie sowohl über aktuelle als auch über Anomalien informiert, die in den letzten 24 Stunden aufgetreten sind. Weitere Informationen finden Sie unter So funktioniert DevOps Guru for RDS.	13. Dezember 2022
Amazon-RDS-Proxy unterstützt globale Datenbanken	Sie können jetzt RDS-Proxy mit globalen Aurora-Datenbanken verwenden. Weitere Informationen finden Sie unter Verwenden von RDS-Proxy mit globalen Aurora-Datenbanken.	7. Dezember 2022

DB-Cluster von Aurora PostgreSQL unterstützen Trusted Language Extensions für PostgreSQL	Trusted Language Extensions für PostgreSQL ist ein Open-Source-Entwicklungskit, mit dem Sie leistungsstarke PostgreSQL-Erweiterungen erstellen und diese sicher auf Ihrem DB-Cluster von Aurora PostgreSQL ausführen können. Weitere Informationen finden Sie unter Arbeiten mit Trusted Language Extensions für PostgreSQL.	30. November 2022
Amazon GuardDuty RDS Protection überwacht auf Zugriffsbedrohungen	Wenn Sie GuardDuty RDS Protection aktivieren, verarbeitet GuardDuty RDS-Anmeldeereignisse aus Ihren Aurora-Datenbanken, überwacht diese Ereignisse und erstellt Profile für potenzielle Insider-Bedrohungen oder externe Akteure. Wenn GuardDuty RDS Protection eine potenzielle Bedrohung erkennt, GuardDuty generiert es ein neues Ergebnis mit Details über die potenziell gefährdete Datenbank. Weitere Informationen finden Sie unter Überwachen von Bedrohungen mit GuardDuty RDS Protection.	30. November 2022

Verwenden Sie Amazon RDS Blue/Green Deployments für Datenbank-Updates	Sie können Änderungen an einem DB-Cluster in einer Staging-Umgebung vornehmen und die Änderungen testen, ohne dass sich dies auf Ihren Produktions-DB-Cluster auswirkt. Wenn Sie bereit sind, können Sie die Staging-Umgebung zur neuen Produktionsdatenbankumgebung hochstufen, wobei die Ausfallzeit minimal ist. Weitere Informationen finden Sie unter Verwenden von Amazon Blue/Green RDS-Bereitstellungen für Datenbank-Updates.	27. November 2022
Amazon Aurora ist in der Region Asien-Pazifik (Hyderabad) verfügbar	Amazon Aurora ist jetzt in der Region Asien-Pazifik (Hyderabad) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	22. November 2022
Amazon Aurora ist in der Region Europa (Spanien) verfügbar	Amazon Aurora ist jetzt in der Region Europa (Spanien) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	16. November 2022

Amazon Aurora ist in der Region Europa (Zürich) verfügbar	Amazon Aurora ist jetzt in der Region Europa (Zürich) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	9. November 2022
Amazon Aurora unterstützt das Exportieren von Daten aus DB-Clustern nach Amazon S3	Sie können Aurora-Cluster-Daten jetzt direkt nach S3 exportieren, ohne zuerst einen Snapshot erstellen zu müssen. Weitere Informationen finden Sie unter Exportieren von DB-Cluster-Daten nach Amazon S3.	27. Oktober 2022
Amazon Aurora MySQL unterstützt schnellere Exporte nach Amazon S3	Sie können jetzt von einer bis zu zehnmal schnelleren Leistung beim Export von DB-Cluster-Snapshot-Daten nach S3 für MySQL 5.7- und 8.0-kompatible Aurora-MySQL-Cluster profitieren. Weitere Informationen finden Sie unter Exportieren von DB-Cluster-Snapshot-Daten nach Amazon S3.	20. Oktober 2022

Amazon Aurora unterstützt die automatische Einrichtung von Konnektivität zwischen einem Aurora-DB-Cluster und einer EC2 Instance.	Sie können den verwenden AWS-Managementkonsole , um Konnektivität zwischen einem vorhandenen Aurora-DB-Cluster und einer EC2 Instance einzurichten. Weitere Informationen finden Sie unter Automatisches Verbinden einer EC2 Instance und eines Aurora-DB-Clusters.	14. Oktober 2022
AWS JDBC-Treiber für PostgreSQL allgemein verfügbar	Der AWS JDBC-Treiber für PostgreSQL ist ein Client-Treiber, der für Aurora PostgreSQL entwickelt wurde. Der AWS JDBC-Treiber für PostgreSQL ist jetzt allgemein verfügbar. Weitere Informationen finden Sie unter Verbindung mit dem AWS JDBC-Treiber für PostgreSQL herstellen.	6. Oktober 2022
Amazon Aurora unterstützt das direkte Upgrade für MySQL-5.7-kompatibles Aurora MySQL	Sie können ein direktes Upgrade vornehmen, um einen vorhandenen mit MySQL 5.7 kompatiblen Aurora-MySQL-Cluster in einen mit MySQL 8.0 kompatiblen Aurora-MySQL-Cluster zu ändern. Weitere Informationen finden Sie unter Upgrade von Aurora MySQL 2.x auf 3.x.	26. September 2022

Performance-Insights zeigt die 25 wichtigsten SQL-Abfragen	Auf der Registerkarte Top SQL (Top-SQL) werden die 25 SQL-Abfragen angezeigt, die am meisten zur DB-Last beitragen. Weitere Informationen finden Sie unter Übersicht über die Registerkarte „Top SQL“ (Top-SQL).	13. September 2022
Aurora MySQL unterstützt eine neue DB-Instance-Klasse	Sie können jetzt die DB-Instance-Klasse db.r6i für DB-Cluster von Aurora MySQL verwenden. Weitere Informationen finden Sie unter DB-Instance-Klassen.	13. September 2022
Amazon Aurora ist in der Region Naher Osten (VAE) verfügbar	Amazon Aurora ist jetzt in der Region Naher Osten (VAE) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	30. August 2022
Amazon Aurora unterstützt das automatische Einrichten von Konnektivität mit einer EC2 Instance	Wenn Sie einen Aurora-DB-Cluster erstellen, können Sie den verwenden, AWS-Managementkonsole um die Konnektivität zwischen einer Amazon Elastic Compute Cloud-Instance und dem neuen DB-Cluster einzurichten. Weitere Informationen finden Sie unter Automatische Netzwerkkonnektivität mit einer EC2 Instance konfigurieren.	22. August 2022

Amazon RDS unterstützt den Dual-Stack-Modus	DB-Cluster können jetzt im Dual-Stack-Modus ausgeführt werden. Im Dual-Stack-Modus können Ressourcen über, oder beides IPv4 IPv6, mit dem DB-Cluster kommunizieren. Weitere Informationen finden Sie unter Amazon Aurora-IP-Adressierung.	17. August 2022
Amazon Aurora unterstützt In-Place-Upgrade für PostgreSQL-kompatible Aurora Serverless v1	Sie können ein In-Place-Upgrade für einen PostgreSQL 10-kompatiblen Aurora Serverless v1 Cluster durchführen, um einen bestehenden Cluster in einen PostgreSQL 11-kompatiblen Aurora Serverless v1 Cluster zu verwandeln. Informationen zum In-Place-Upgrade-Verfahren finden Sie unter Ändern eines Aurora Serverless v1 DB-Clusters.	08. August 2022
Performance Insights unterstützt die Region Asien-Pazifik (Jakarta)	Bisher konnten Sie Performance Insights in der Region Asien-Pazifik (Jakarta) nicht verwenden. Diese Einschränkung wurde entfernt. Weitere Informationen finden Sie unter Support der AWS-Region für Performance Insights.	21. Juli 2022

Amazon Aurora unterstützt eine neue DB-Instance-Klasse	Sie können jetzt die db.r6i DB-Instance-Klasse für Aurora PostgreSQL DB-Cluster verwenden. Weitere Informationen finden Sie unter DB-Instance-Klassen.	14. Juli 2022
RDS Performance Insights unterstützt zusätzliche Aufbewahrungsfristen	Bisher bot Performance Insights nur zwei Aufbewahrungsfristen an: 7 Tage (Standard) oder 2 Jahre (731 Tage). Wenn Sie Ihre Leistungsdaten jetzt länger als 7 Tage aufbewahren müssen, können Sie zwischen 1 und 24 Monaten angeben. Weitere Informationen finden Sie unterPreisgestaltung und Datenspeicherung für Performance Insights.	01. Juli 2022
Amazon Aurora unterstützt das In-Place-Upgrade für MySQL-kompatible Aurora Serverless v1	Sie können ein direktes Upgrade für einen mit MySQL 5.6 kompatiblen Aurora Serverless v1-Cluster vornehmen, um einen vorhandenen Cluster in einen mit MySQL 5.7 kompatiblen Aurora Serverless v1-Cluster zu ändern. Informationen zum In-Place-Upgrade-Verfahren finden Sie unter Ändern eines Aurora Serverless v1 DB-Clusters.	16. Juni 2022

Aurora unterstützt das Einschalten von Amazon DevOps Guru in der RDS-Konsole	Sie können die DevOps Guru-Abdeckung für Ihre Aurora-Datenbanken von der RDS-Konsole aus aktivieren. Weitere Informationen finden Sie unter DevOpsGuru für RDS einrichten.	9. Juni 2022
Amazon Aurora unterstützt das Veröffentlichen von Ereignissen in verschlüsselten Amazon-SNS-Themen	Amazon Aurora kann jetzt Ereignisse in Amazon Simple Notification Service (Amazon-SNS)-Themen veröffentlichen, bei denen serversei tige Verschlüsselung (SSE) aktiviert ist, um Ereignisse, die sensible Daten enthalten , zusätzlich zu schützen. Weitere Informationen finden Sie unter Abonniere n von Amazon-RDS-Ereignisbenachrichtigungen.	1. Juni 2022
Amazon RDS veröffentlicht Nutzungsmetriken auf Amazon CloudWatch	Der `AWS/Usage` Namespace in Amazon CloudWatch enthält Nutzungsmetriken auf Kontoebene für Ihre Amazon RDS-Servicekonting enten. Weitere Informationen finden Sie unter CloudWatch Amazon-Nutzungsmetriken für Amazon Aurora.	28. April 2022

Daten-API-Ergebnissätze im JSON-Format	Ein optionaler Parameter für die Funktion `ExecuteStatement` bewirkt, dass der Abfrageergebnissatz als Zeichenfolge im JSON-Format zurückgegeben wird. Der JSON-Ergebnissatz lässt sich einfach und bequem in eine Datenstruktur in der Sprache Ihrer Anwendung umwandeln. Weitere Informationen finden Sie unter Verarbeiten von Abfrageergebnissen im JSON-Format.	27. April 2022
Amazon Aurora Serverless v2 ist jetzt allgemein verfügbar.	Amazon Aurora Serverless v2 ist für alle Benutzer allgemein verfügbar. Weitere Informationen finden Sie unter Verwendung von Aurora Serverless v2.	21. April 2022
Aurora MySQL unterstützt konfigurierbare Cipher-Suites	Mit Aurora MySQL können Sie nun konfigurierbare Cipher-Suites verwenden, um die Verbindungsverschlüsselung zu steuern, die Ihr Datenbankserver akzeptiert. Weitere Informationen finden Sie unter Konfigurieren von Cipher-Suites für Verbindungen mit Aurora-MySQL-DB-Clustern.	15. April 2022

Aurora PostgreSQL unterstützt RDS-Proxy mit PostgreSQL 13	Sie können jetzt einen RDS-Proxy mit einem Aurora PostgreSQL 13-DB-Cluster erstellen. Weitere Informationen zu RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	4. April 2022
Versionshinweise für Aurora PostgreSQL	Es gibt jetzt einen separaten Leitfaden für die Versionshinweise von Amazon Aurora PostgreSQL. Weitere Informationen finden Sie unter Versionshinweise für Aurora PostgreSQL.	22. März 2022
Versionshinweise für Aurora MySQL	Es gibt jetzt einen separaten Leitfaden für die Versionshinweise von Amazon Aurora MySQL. Weitere Informationen finden Sie unter Versionshinweise für Aurora MySQL.	22. März 2022
Aurora PostgreSQL unterstützt Upgrades auf Multi-Hauptversionen	Sie können jetzt Versionsupgrades von Aurora-PostgreSQL-DB-Clustern über mehrere Hauptversionen hinweg durchführen. Weitere Informationen finden Sie unter So führen Sie ein Hauptversionsupgrade durch.	4. März 2022

Aurora PostgreSQL unterstützt konfigurierbare Cipher-Suites	Mit Aurora PostgreSQL Version 11.8 und höher können Sie jetzt konfiguri erbare Cipher-Suites verwenden, um die Verbindun gsverschlüsselung zu steuern, die Ihr Datenbankserver akzeptiert. Weitere Informati onen zur Verwendung konfigurierbarer Cipher-Suites mit Aurora PostgreSQL finden Sie unter Konfigurieren von Cipher-Suites für Verbindun gen mit Aurora-PostgreSQL-DB-Clustern.	4. März 2022
AWS JDBC-Treiber für MySQL allgemein verfügbar	Der AWS JDBC-Treiber für MySQL ist ein Client-Treiber, der für die Hochverfügbarkeit von Aurora MySQL entwickel t wurde. Der AWS JDBC-Treiber für MySQL ist jetzt allgemein verfügbar. Weitere Informationen finden Sie unter Herstellen einer Verbindun g mit dem Amazon Web Services JDBC-Treiber für MySQL.	2. März 2022

Aurora PostgreSQL 13.5 unterstützt Babelfish für Aurora PostgreSQL 1.1.0	Aurora PostgreSQL 13.5 unterstützt Babelfish 1.1.0. Eine Liste der neuen Funktionen finden Sie unter 13.5. Eine Liste der Funktionen, die in jeder Babelfish-Version unterstützt werden, finden Sie unter Unterstützte Funktionalität in Babelfish nach Version. Weitere Nutzungsinformationen finden Sie unter Arbeiten mit Babelfish für Aurora PostgreSQL.	28. Februar 2022
Amazon Aurora unterstützt Datenbank-Aktivitäts-Streams in der Region Asien-Pazifik (Jakarta)	Weitere Informationen finden Sie unter Support AWS-Regionen für Datenbankaktivitätsstreams.	16. Februar 2022
Performance Insights unterstützt neue API-Operationen	Performance Insights unterstützt die folgenden API-Operationen: `GetResourceMetadata`, `ListAvailableResourceDimensions` und `ListAvailableResourceMetrics`. Weiteren Informationen finden Sie unter Abrufen von Metriken mit der Performance Insights API in diesem Handbuch und in der Referenz zur Amazon RDS Performance Insights API.	12. Januar 2022

Amazon-RDS-Proxy unterstützt Ereignisse	RDS Proxy generiert jetzt Ereignisse, die Sie abonnieren und unter CloudWatch Ereignisse anzeigen oder so konfigurieren können, dass sie an Amazon gesendet EventBridge werden. Weitere Informationen finden Sie unter Arbeiten mit RDS-Proxy-Ereignissen.	11. Januar 2022
RDS Proxy ist zusätzlich erhältlich AWS-Regionen	RDS-Proxy ist jetzt in den folgenden Regionen erhältlich: Afrika (Kapstadt), Asien-Pazifik (Hongkong, Osaka), Europa (Milan, Paris, Stockholm), Naher Osten (Bahrain) und Südamerika (São Paulo). Weitere Informationen über RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	5. Januar 2022
Amazon Aurora ist in der Region Asien-Pazifik (Jakarta) verfügbar	Amazon Aurora ist jetzt in der Region Asien-Pazifik (Jakarta) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	13. Dezember 2021

DevOpsGuru for Amazon RDS bietet detaillierte Einblicke und Empfehlungen für Amazon Aurora	DevOpsGuru for RDS nutzt Performance Insights für leistungsbezogene Daten. Anhand dieser Daten analysiert der Service die Leistung Ihrer Amazon Aurora DB-Instances und kann Ihnen bei der Lösung von Leistungsproblemen helfen. Weitere Informationen finden Sie unter Analysieren von Leistungsanomalien mit DevOps Guru for RDS in diesem Handbuch und unter Überblick über DevOps Guru for RDS im Amazon DevOps Guru-Benutzerhandbuch.	1. Dezember 2021
Aurora PostgreSQL unterstützt RDS-Proxy mit PostgreSQL 12	Sie können jetzt einen RDS-Proxy mit einem Aurora PostgreSQL 12-Datenbank-Cluster erstellen. Weitere Informationen zu RDS-Proxy finden Sie unter Verwenden von Amazon-RDS-Proxy.	22. November 2021
Aurora unterstützt AWS Graviton2-Instanzklassen für Datenbankaktivitätsstreams	Sie können Datenbank-Aktivitätsstreams mit der Instance-klasse db.r6g für Aurora MySQL und Aurora PostgreSQL verwenden. Weitere Informationen finden Sie unter Unterstützte DB-Instance-Klassen.	3. November 2021

Amazon Aurora Aurora-Unterstützung für kontoüber greifende Konten AWS KMS keys	Sie können einen KMS-Schlüssel von einem anderen AWS-Konto für die Verschlüsselung verwenden, wenn Sie DB-Snapshots nach Amazon S3 exportieren. Weitere Informationen finden Sie unter Exportieren von DB-Snapshot-Daten nach Amazon S3.	3. November 2021
Amazon Aurora unterstützt Babelfish für Aurora PostgreSQL	Babelfish for Aurora PostgreSQL erweitert Ihre Amazon Aurora PostgreSQL-kompatible Edition-Datenbank um die Möglichkeit, Datenbankverbindungen von Microsoft SQL Server-Clients zu akzeptieren. Weitere Informationen finden Sie unter Arbeiten mit Babelfish für Aurora PostgreSQL.	28. Oktober 2021
Aurora Serverless v1 kann SSL für Verbindungen benötigen	Die Aurora-Clusterparameter `force_ssl` für PostgreSQL und `require_secure_transport` für MySQL werden für Aurora Serverless v1 jetzt unterstützt. Weitere Informationen finden Sie unter Verwenden TLS/SSL mit Aurora Serverless Version 1.	26. Oktober 2021

Amazon Aurora unterstützt Performance Insights zusätzlich AWS-Regionen	Performance Insights ist in den Regionen Naher Osten (Bahrain), Afrika (Kapstadt), Europa (Mailand) und Asien-Pazifik (Osaka) verfügbar. Weitere Informationen finden Sie unter Support der AWS-Region für Performance Insights.	5. Oktober 2021
Konfigurierbares Autoskalierungs-Timeout für Aurora Serverless v1	Sie können auswählen, wie lange Aurora Serverless v1 wartet, um einen Punkt für die automatische Skalierung zu finden. Wenn in diesem Zeitraum kein Autoskalierungspunkt gefunden wird, bricht Aurora Serverless v1 das Skalierungsereignis ab oder erzwingt die Kapazitätsänderung, je nach der von Ihnen ausgewählten Timeout-Aktion. Weitere Informationen finden Sie unter Autoskalierung für Aurora Serverless v1.	10 September 2021

Aurora unterstützt X2g- und T4g-Instance-Klassen	Sowohl Aurora MySQL als auch Aurora PostgreSQL können jetzt X2g- und T4g-Instance-Klassen verwenden . Die Instance-Klassen, die Sie verwenden können, hängen von der Version von Aurora MySQL oder Aurora PostgreSQL ab. Informationen zu den unterstützten Instance-Typen finden Sie unter DB-Instance-Klassen.	10. September 2021
Amazon RDS unterstützt RDS-Proxy in einer gemeinsam genutzten VPC	Jetzt können Sie einen RDS-Proxy in einer gemeinsam genutzten Virtual Private Cloud (VPC) erstellen. Weitere Informationen zu RDS-Proxy finden Sie unter „Verwalten von Verbindungen mit Amazon-RDS-Proxy“ im Amazon-RDS-Benutzerhandbuch oder im Aurora-Benutzerhandbuch.	6. August 2021
Aurora-Versionsrichtlinienseite	Das Amazon-Aurora-Benutzerhandbuch enthält jetzt einen Abschnitt mit allgemeinen Informationen zu Aurora-Versionen und den zugehörigen Richtlinien. Details dazu finden Sie unter Amazon-Aurora-Versionen.	14. Juli 2021

Schließt Daten-API-Ereignisse von einem AWS CloudTrail Trail aus	Sie können Daten-API-Ereignisse von einem CloudTrail Trail ausschließen. Weitere Informationen finden Sie unter Daten-API-Ereignisse aus einem AWS CloudTrail Trail ausschließen.	2. Juli 2021
Amazon Aurora PostgreSQL-kompatible Edition unterstützt zusätzliche Erweiterungen	Zu den neu unterstützten Erweiterungen gehören pg_bigm, pg_cron, pg_partman und pg_proctab. Weitere Informationen finden Sie unter Erweiterungsversionen für Amazon-Aurora-PostgreSQL-kompatible Edition.	17. Juni 2021
Klonen für Aurora Serverless-Cluster	Sie können jetzt geklonte Cluster erstellen, die Aurora Serverless sind. Weitere Informationen zum Klonen finden Sie unter Klonen eines Aurora-DB-Clusters.	16. Juni 2021
Aurora globale Datenbanken sind in China (Peking)- und China (Ningxia)-Regionen verfügbar	Sie können jetzt Aurora globale Datenbanken in den Regionen China (Peking) und China (Ningxia) erstellen. Weitere Informationen zu globalen Aurora-Datenbanken finden Sie unter Arbeiten mit globalen Amazon Aurora-Datenbanken.	19. Mai 2021

FIPS 140-2-Unterstützung für Daten-API	Die Daten-API unterstützt die Federal Information Processing Standard Publication 140-2 (FIPS 140-2) für Verbindungen. SSL/TLS Weitere Informationen finden Sie unter Verfügbarkeit der Daten-API .	14. Mai 2021
AWS JDBC-Treiber für PostgreSQL (Vorschau)	Der AWS JDBC-Treiber für PostgreSQL, der jetzt als Vorschauversion verfügbar ist, ist ein Client-Treiber, der für die Hochverfügbarkeit von Aurora PostgreSQL entwickelt wurde. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit Amazon Web Services JDBC-Treiber für PostgreSQL (Vorschau).	27. April 2021
Die Daten-API ist zusätzlich verfügbar AWS-Regionen	Die Daten-API ist jetzt in den Regionen Asien-Pazifik (Seoul) und Kanada (Zentral) verfügbar. Weitere Informationen finden Sie unter Verfügbarkeit der Daten-API.	9. April 2021
Amazon Aurora unterstützt die Graviton2 DB-Instance-Klassen	Sie können jetzt die Graviton2-DB-Instance-Klassen db.r6g.x verwenden, um DB-Cluster mit MySQL oder PostgreSQL zu erstellen. Weitere Informationen finden Sie unter DB-Instance-Klassenarten.	12. März 2021

Erweiterungen des RDS-Proxy-Endpunkts	Sie können zusätzliche Endpunkte erstellen, die jedem RDS-Proxy zugeordnet sind. Das Erstellen eines Endpunkts in einer anderen VPC ermöglicht den VPC-übergreifenden Zugriff für den Proxy. Proxies für Aurora MySQL-Cluster können auch schreibgeschützte Endpunkte haben. Diese Reader-Endpunkte stellen eine Verbindung zu Reader-DB-Instances in den Clustern her und können die Leseskalierbarkeit und Verfügbarkeit für abfragein tensive Anwendungen verbessern. Weitere Informationen zu RDS-Proxy finden Sie unter „Verwalten von Verbindungen mit Amazon-RDS-Proxy“ im Amazon-RDS-Benutzerhandbuch oder im Aurora-Benutzerhandbuch.	8. März 2021
Amazon Aurora ist in der Region Asien-Pazifik (Osaka) verfügbar	Amazon Aurora ist jetzt in der Region Asien-Pazifik (Osaka) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	1. März 2021

Aurora PostgreSQL unterstützt die Aktivierung von IAM- und Kerberos-Authentifizierung auf demselben DB-Cluster	Aurora PostgreSQL unterstützt jetzt die Aktivierung sowohl der IAM-Authentifizierung als auch der Kerberos-Authentifizierung auf demselben DB-Cluster. Weitere Informationen finden Sie unter Datenbank-Authentifizierung mit Amazon Aurora .	24. Februar 2021
Die globale Aurora-Datenbank unterstützt jetzt verwaltetes geplantes Failover	Die globale Aurora-Datenbank unterstützt jetzt verwaltetes geplantes Failover, sodass Sie die primäre AWS -Region Ihrer globalen Aurora-Datenbank einfacher ändern können. Sie können das verwaltete geplante Failover nur mit fehlerfreien globalen Aurora-Datenbanken verwenden . Weitere Informationen finden Sie unter Notfallwiederherstellung und globale Amazon Aurora-Datenbanken . Referenzinformationen finden Sie im Amazon RDS API Reference unter `FailoverGlobalCluster` .	11. Februar 2021

Die Daten-API für Aurora Serverless unterstützt jetzt mehr Datentypen	Mit der Daten-API für Aurora Serverless können Sie jetzt `UUID`- und `JSON`-Datentypen als Eingabe für die Datenbank verwenden. Auch mit der Daten-API für Aurora Serverless können Sie jetzt den Typwert `LONG` als `STRING`-Wert aus Ihrer Datenbank zurückgeben lassen. Weitere Informationen finden Sie unter Aufrufen der Daten-API. Referenzinformationen zu unterstützten Datentypen finden Sie im Data Service-API-Referenz von Amazon RDS unter `SqlParameter`.	2. Februar 2021
Aurora PostgreSQL unterstützt Upgrades auf Hauptversionen für PostgreSQL 12	Mit Aurora PostgreSQL können Sie nun die DB-Engine auf Hauptversion 12 aktualisieren. Weitere Informationen finden Sie unter Aktualisieren der PostgreSQL-DB-Engine für Aurora PostgreSQL.	28. Januar 2021

Aurora MySQL unterstützt ein direktes Upgrade	Sie können Ihren Aurora MySQL 1.x-Cluster auf Aurora MySQL 2.x upgraden, wobei die DB-Instances, Endpunkte usw. des ursprünglichen Clusters erhalten bleiben. Diese integrierte Upgrade-Technik vermeidet die Unannehmlichkeit beim Einrichten eines ganz neuen Clusters durch Wiederherstellen eines Snapshots. Sie vermeidet auch den Aufwand beim Kopieren aller Ihrer Tabellendaten in einen neuen Cluster. Weitere Informationen finden Sie unter Upgrade der Hauptversion eines Aurora MySQL DB-Clusters von 1.x auf 2.x.	11. Januar 2021
AWS JDBC-Treiber für MySQL (Vorschau)	Der AWS JDBC-Treiber für MySQL, der jetzt als Vorschauversion verfügbar ist, ist ein Client-Treiber, der für die hohe Verfügbarkeit von Aurora MySQL entwickelt wurde. Weitere Informationen finden Sie unter Herstellen einer Verbindung mit dem Amazon Web Services JDBC-Treiber für MySQL (Vorschau).	7. Januar 2021

Aurora unterstützt Datenbank aktivitäts-Streams auf sekundären Clustern einer globalen Datenbank	Sie können eine Datenbank mit einem Datenbankaktivität sstream auf einem primären oder sekundären Cluster von Aurora PostgreSQL oder Aurora MySQL starten. Informationen zu unterstützten Engine-Versionen finden Sie unter Einschränkungen der globalen Aurora-Datenbanken.	22. Dezember 2020
Multi-Master-Cluster mit 4 DB-Instances	Die maximale Anzahl von DB-Instances in einem Aurora MySQL Multi-Master-Clust er beträgt jetzt vier. Früher waren zwei DB-Instances das Maximum. Weitere Informati onen finden Sie unter Arbeiten mit Aurora-Multi-Master-Clustern.	17. Dezember 2020
Aurora PostgreSQL unterstützt Funktionen AWS Lambda	Sie können jetzt die AWS Lambda Funktion für Ihre Aurora PostgreSQL-DB-Clus ter aufrufen. Weitere Informati onen finden Sie unter Aufrufen einer Lambda-Funktion aus einem Aurora PostgreSQL-DB-Cluster.	11. Dezember 2020

Amazon Aurora unterstützt die Graviton2 DB-Instance-Klassen in der Vorschau	Sie können jetzt die Graviton2 -DB-Instance-Klassen db.r6g.x in der Vorschau verwenden , um DB-Cluster mit MySQL oder PostgreSQL zu erstellen . Weitere Informationen finden Sie unter DB-Instance-Klassenarten.	11. Dezember 2020
Amazon Aurora Serverless v2 ist jetzt in der Vorschauversion erhältlich.	Amazon Aurora Serverless v2 ist in der Vorschauversion verfügbar. Fordern Sie Zugriff an, um mit Amazon Aurora Serverless v2 arbeiten zu können. Weitere Informationen finden Sie auf der Seite für Aurora Serverless v2.	1. Dezember 2020

Aurora PostgreSQL ist jetzt für Aurora Serverless in weiteren AWS-Regionen verfügbar.	Aurora PostgreSQL ist jetzt für Aurora Serverless in weiteren AWS-Regionen verfügbar. Sie können jetzt wählen, ob Sie dasselbe Aurora PostgreSQL Serverless v1 AWS-Regionen Angebot verwenden möchten. Aurora MySQL Serverless v1 Zu den weiteren AWS-Regionen Aurora Serverless Unterstützungsangeboten gehören USA West (Nordkalifornien), Asien-Pazifik (Singapur), Asien-Pazifik (Sydney), Asien-Pazifik (Seoul), Asien-Pazifik (Mumbai), Kanada (Mittel-) Europa (London) und Europa (Paris). Eine Liste aller Regionen und unterstützten Aurora-DB-Engines für Aurora Serverless finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v1. Die Amazon-RDS-Daten-API für Aurora Serverless ist jetzt auch in denselben AWS-Regionen verfügbar. Eine Liste aller Regionen mit Unterstützung für die Daten-API für Aurora Serverless finden Sie unter Daten-API für Aurora MySQL Serverless v1	24. November 2020

Amazon-RDS-Performance Insights führt neue Dimensionen ein	Sie können die Datenbanklast nach den Dimensionsgruppen für Datenbank, Anwendung (PostgreSQL) und Sitzungstyp (PostgreSQL) gruppieren. Amazon RDS unterstützt auch die Dimensionen db.name, db.application.name (PostgreSQL) und db.session_type.name (PostgreSQL). Weitere Informationen finden Sie unter Hauptlast-Tabelle.	24. November 2020
Aurora Serverless unterstützt jetzt Version 10.12 von Aurora PostgreSQL	Aurora PostgreSQL für Aurora Serverless wurde in allen AWS -Regionen, in denen Aurora PostgreSQL für Aurora Serverless unterstützt wird, auf Aurora PostgreSQL Version 10.12 aktualisiert. Weitere Informationen finden Sie unter Unterstützte Regionen und Aurora-DB-Engines für Aurora Serverless v1.	4. November 2020

Die Daten-API unterstützt jetzt die Tag-basierte Autorisierung	Die Daten-API unterstützt die Tag-basierte Autorisierung. Wenn Sie Ihre RDS-Clusterressourcen mit Tags gekennzeichnet haben, können Sie diese Tags in Ihren Richtlinienanweisungen verwenden, um den Zugriff über die Daten-API zu steuern. Weitere Informationen finden Sie unter Autorisieren des Zugriffs auf die Daten-API.	27. Oktober 2020
Amazon Aurora erweitert Unterstützung für den Export von Snapshots nach Amazon S3	Sie können jetzt in allen kommerziellen AWS-Regionen DB-Snapshot-Daten nach Amazon S3 exportieren. Weitere Informationen finden Sie unter Exportieren von DB-Snapshot-Daten nach Amazon S3.	22. Oktober 2020

Die Globale Aurora-Datenbank unterstützt das Klonen	Sie können jetzt Klone der primären und sekundären DB-Cluster Ihrer globalen Aurora-Datenbanken erstellen. Sie können dies tun, indem Sie die Menüoption „Klon erstellen “ verwenden AWS-Manag ementkonsole und wählen. Sie können auch den Befehl verwenden AWS CLI und den `restore-db-cluster -to-point-in-time` Befehl mit der `--restore -type copy-on-write` Option ausführen. Mit dem AWS-Managementkonsole oder dem AWS CLI können Sie auch DB-Cluster aus Ihren globalen Aurora-Datenbanken AWS kontenübergreifend klonen. Weitere Informati onen zum Klonen finden Sie unter Klonen eines Aurora DB-Cluster-Volumes.	19. Oktober 2020

Amazon Aurora unterstützt die dynamische Größenanpassung für das Cluster-Volume	Beginnend mit Aurora MySQL 1.23 und 2.09 sowie Aurora PostgreSQL 3.3.0 und Aurora PostgreSQL 2.6.0 reduziert Aurora die Größe des Cluster-Volumes nach dem Entfernen von Daten über Operationen wie `DROP TABLE`. Um diese Erweiterung nutzen zu können, aktualisieren Sie auf eine der entsprechenden Versionen, abhängig von der Datenbank-Engine, die Ihr Cluster verwendet. Weitere Informationen zu dieser Funktion und zur Überprüfung des verwendeten und verfügbaren Speicherplatzes für einen Aurora-Cluster finden Sie unter Verwalten von Performance und Skalierung für Aurora DB-Cluster.	13. Oktober 2020
Amazon Aurora unterstützt Volumegrößen bis zu 128 TiB	Neue und bestehende Aurora-Cluster-Volumes können jetzt auf eine maximale Größe von 128 tebibytes (TiB) erweitert werden. Weitere Informationen finden Sie unter Informationen zum Anwachsen des Aurora-Speichers.	22. September 2020

Aurora PostgreSQL unterstützt die DB-Instance-Klassen db.r5 und db.t3 in der Region China (Ningxia)	Sie können jetzt Aurora–PostgreSQL-DB-Cluster in der Region China (Ningxia) erstellen, die die DB-Instance-Klassen db.r5 und db.t3 verwenden. Weitere Informationen finden Sie unter DB-Instance-Klassen.	3. September 2020

Aurora-Erweiterungen für parallele Abfragen	Ab Aurora MySQL 2.09 und 1.23 können Sie die Vorteile der Verbesserungen für die Funktion für parallele Abfragen nutzen. Für das Erstellen eines parallele n Abfrageclusters ist kein spezieller Engine-Modus mehr erforderlich. Sie können die parallele Abfrage jetzt mithilfe der Konfigurationsopti on `aurora_parallel_qu ery` für jeden bereitges tellten Cluster aktivieren und deaktivieren, auf dem eine kompatible Aurora MySQL-Version ausgeführ t wird. Sie können einen vorhandenen Cluster auf eine kompatible Aurora MySQL-Version aktualisieren und eine parallele Abfrage verwenden, anstatt einen neuen Cluster zu erstellen und in diesen Daten zu importier en. Sie können Performan ce Insights für parallele Abfragecluster verwenden . Sie können parallele Abfragecluster stoppen und starten. Sie können parallele Aurora-Abfragecluster erstellen, die mit MySQL 5.7 kompatibel sind. Die parallele Abfrage funktioniert für Tabellen, die das Zeilenfor	2. September 2020

	mat `DYNAMIC` verwenden . Parallele Abfragecluster können die AWS Identity and Access Managemen t (IAM-) Authentifizierung verwenden. DB-Reader -Instances in parallelen Abfrageclustern können die Vorteile der Isolationsstufe `READ COMMITTED` nutzen. Sie können jetzt auch parallele Abfrage-Cluster in zusätzlic hen AWS-Regionen erstellen . Weitere Informationen zur Funktion für parallele Abfragen und zu diesen Verbesser ungen finden Sie unter Parallel Query für Aurora MySQL.	
Änderungen am Aurora MySQL-Parameter binlog_ro ws_query_log_events	Sie können nun den Wert des Aurora MySQL-Kon figurationsparameters ändern `binlog_rows_query_ log_events` . Früher war dieser Parameter nicht veränderbar.	26. August 2020

Unterstützung für automatische Nebenversions-Upgrades für Aurora MySQL	Mit Aurora MySQL wird die Einstellung Enable auto minor version upgrade (Automatisches Nebenversions-Upgrade aktivieren) jetzt wirksam, wenn Sie sie für einen Aurora MySQL-DB-Cluster angeben. Wenn Sie das automatische Nebenversions-Upgrade aktivieren, wird Aurora automatisch auf neue Nebenversionen aktualisiert, sobald diese veröffentlicht werden. Die automatischen Upgrades erfolgen während des Wartungsfensters für die Datenbank. Für Aurora MySQL gilt diese Funktion nur für Aurora MySQL Version 2, die mit MySQL 5.7 kompatibel ist. Zunächst bringt das automatische Upgrade-Verfahren Aurora MySQL-DB-Cluster auf Version 2.07.2. Weitere Informationen zur Funktionsweise dieser Funktion mit Aurora MySQL finden Sie unter Datenbank-Upgrades und Patches für Amazon Aurora MySQL.	3. August 2020

Aurora PostgreSQL unterstützt Upgrades auf Hauptversionen für PostgreSQL Version 11	Mit Aurora PostgreSQL können Sie nun die DB-Engine auf Hauptversion 11 aktualisieren. Weitere Informationen finden Sie unter Aktualisieren der PostgreSQL-DB-Engine für Aurora PostgreSQL.	28. Juli 2020
Amazon Aurora unterstützt AWS PrivateLink	Amazon Aurora unterstützt jetzt die Erstellung von Amazon VPC-Endpunkten für Amazon RDS-API-Aufrufe, um den Verkehr zwischen Anwendungen und Aurora im AWS Netzwerk aufrechtzuerhalten. Weitere Informationen finden Sie unter Amazon Aurora und Schnittstellen-VPC-Endpunkte (AWS PrivateLink).	9. Juli 2020
RDS-Proxy allgemein verfügbar	RDS-Proxy ist jetzt allgemein verfügbar. Sie können RDS-Proxy mit RDS MySQL, Aurora MySQL, RDS für PostgreSQL und Aurora PostgreSQL für Produktions-Workloads verwenden. Weitere Informationen zu RDS-Proxy finden Sie unter „Verwalten von Verbindungen mit Amazon-RDS-Proxy“ im Amazon-RDS-Benutzerhandbuch oder im Aurora-Benutzerhandbuch.	30. Juni 2020

Schreibweiterleitung in der globalen Aurora-Datenbank	Sie können jetzt die Schreibfunktion für sekundäre Cluster in einer globalen Datenbank aktivieren. Bei der Schreibweiterleitung geben Sie DML-Anweisungen in einem sekundären Cluster aus, Aurora leiten die Schreiboperation an den primären Cluster weiter, und die aktualisierten Daten werden auf alle sekundären Cluster repliziert. Weitere Informationen finden Sie unter Schreibweiterleitung für sekundäre Zwecke AWS-Regionen mit einer globalen Aurora-Datenbank.	18. Juni 2020
Aurora unterstützt die Integration mit AWS Backup	Sie können es verwenden AWS Backup , um Backups von Aurora-DB-Clustern zu verwalten. Weitere Informationen finden Sie unter Übersicht über das Sichern und Wiederherstellen eines Aurora-DB-Clusters.	10. Juni 2020
Aurora PostgreSQL unterstützt DB-Instance-Klassen vom Typ „db.t3.large“	Sie können jetzt Aurora PostgreSQL-DB-Cluster erstellen, die DB-Instance-Klassen vom Typ „db.t3.large“ verwenden. Weitere Informationen finden Sie unter DB-Instance-Klassen.	5. Juni 2020

Die globale Aurora-Datenbank unterstützt PostgreSQL-Version 11.7 und die verwaltete Recovery Point Objective (RPO)	Sie können jetzt globale Aurora-Datenbanken für die PostgreSQL-Datenbank-Engine-Version 11.7. erstellen. Sie können auch verwalten, wie eine globale PostgreSQL-Datenbank mithilfe eines Recovery Point Objective (RPO, Definition eines Zeitraums zwischen zwei Datensicherungen) nach einem Ausfall wiederhergestellt wird. Weitere Informationen finden Sie unter Überregionale Notfallwiederherstellung für globale Aurora-Datenbanken.	4. Juni 2020
Aurora MySQL unterstützt Datenbanküberwachung mit Datenbankaktivitäts-Streams	Aurora MySQL enthält jetzt Datenbankaktivitätsstreams, die einen near-real-time Datenstrom der Datenbankaktivität in Ihrer relationalen Datenbank bereitstellen. Weitere Informationen finden Sie unter Verwenden von Datenbankaktivitäts-Streams.	2. Juni 2020
Der Abfrage-Editor ist zusätzlich verfügbar AWS-Regionen	Der Abfrage-Editor für Aurora Serverless ist jetzt zusätzlich AWS-Regionen verfügbar. Weitere Informationen finden Sie unter Verfügbarkeit des Abfrage-Editors.	28. Mai 2020

Die Daten-API ist zusätzlich verfügbar AWS-Regionen	Die Daten-API ist jetzt zusätzlich verfügbar AWS-Regionen. Weitere Informationen finden Sie unter Verfügbarkeit der Daten-API.	28. Mai 2020
RDS-Proxy verfügbar in Region Kanada (Zentral)	Sie können nun die RDS-Proxy-Vorschau in der Region Region Kanada (Zentral) verwenden. Weitere Informationen zu RDS-Proxy finden Sie unter Verwalten von Verbindungen mit Amazon-RDS-Proxy (Vorschau).	28. Mai 2020
Globale Aurora-Datenbank und regionsübergreifende Read Replicas (Lesereplikate)	Sie können für eine globale Aurora-Datenbank nun ein regionsübergreifendes Aurora MySQL-Read Replica erstellen, wenn sich der primäre Cluster in derselben Region wie ein sekundärer Cluster befindet. Weitere Informationen zu Aurora Globale Datenbank und regionsübergreifenden Read Replicas finden Sie unter Arbeiten mit Amazon Aurora Global Database und Replizieren von Amazon Aurora MySQL DB.	18. Mai 2020

RDS Proxy ist in weiteren Sprachen verfügbar AWS-Regionen	Sie können nun die RDS-Proxy-Vorversion in der Region USA West (Nordkalifornien), der Region Europa (London), der Region Europa (Frankfurt), der Region Asien-Pazifik (Seoul), der Region Asien-Pazifik (Mumbai), der Region Asien-Pazifik (Singapur) und der Region Asien-Pazifik (Sydney) verwenden. Weitere Informationen zu RDS-Proxy finden Sie unter Verwalten von Verbindungen mit Amazon-RDS-Proxy (Vorschau).	13. Mai 2020
Aurora PostgreSQL-kompatible Edition unterstützt On-Premise- oder selbstgehostetes Microsoft Active Directory	Sie können nun ein lokales oder selbstgehostetes Active Directory für die Kerberos-Authentifizierung von Benutzern verwenden, wenn sie eine Verbindung zu Ihren Aurora PostgreSQL-DB-Clustern herstellen. Weitere Informationen finden Sie unter Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL.	7. Mai 2020

Aurora MySQL-Multimaster-Cluster sind in mehr verfügbar AWS-Regionen	Sie können jetzt Aurora-Multi-Master-Cluster in der Region Asien-Pazifik (Seoul), der Region Asien-Pazifik (Tokio), der Region Asien-Pazifik (Mumbai) und der Region Europa (Frankfurt) erstellen . Weitere Informationen zu Multi-Master-Clustern finden Sie unter Arbeiten mit Aurora-Multi-Master-Clustern.	7. Mai 2020
Performance Insights unterstützt die Analyse von Statistiken zu laufenden Aurora MySQL-Abfragen	Sie können jetzt Statistiken laufender Abfragen mit Performance Insights für Aurora MySQL-DB-Instances analysieren. Weitere Informationen finden Sie unter Analysieren von Statistiken zu laufenden Abfragen.	5. Mai 2020
Java-Client-Bibliothek für Daten-API allgemein verfügbar	Die Java-Client-Bibliothek für die Daten-API ist jetzt allgemein verfügbar. Sie können eine Java Client-Bibliothek für Data API herunterladen und verwenden. Mithilfe dieser Bibliothek können Sie Ihre clientseitigen Klassen zu Anforderungen und Antworten der Daten-API zuweisen. Weitere Informationen finden Sie unter Verwenden der Java Client-Bibliothek für die Daten-API.	30. April 2020

Amazon Aurora ist in der Region Europa (Mailand) verfügbar	Amazon Aurora ist jetzt in der Region Europa (Mailand) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	28. April 2020
Amazon Aurora ist in der Region Europa (Mailand) verfügbar	Amazon Aurora ist jetzt in der Region Europa (Mailand) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	27. April 2020
Amazon Aurora ist in der Region Afrika (Kapstadt) verfügbar	Amazon Aurora ist jetzt in der Region Afrika (Kapstadt) verfügbar. Weitere Informationen finden Sie unter Regionen und Availability Zones.	22. April 2020
Aurora PostgreSQL unterstützt jetzt DB-Instance-Klassen db.r5.16xlarge und db.r5.8xlarge	Sie können jetzt Aurora PostgreSQL-DB-Cluster mit PostgreSQL erstellen, die die DB-Instance-Klassen db.r5.16xlarge und db.r5.8xlarge verwenden. Weitere Informationen finden Sie unter Hardware-Spezifikationen für DB-Instance-Klassen für Aurora.	8. April 2020

Amazon-RDS-Proxy für PostgreSQL	Amazon-RDS-Proxy ist jetzt für PostgreSQL verfügbar. Sie können den RDS-Proxy verwenden, um den Overhead für die Verbindungsverwaltung in Ihrem Cluster zu reduzieren und auch die Wahrscheinlichkeit des Fehlers „zu viele Verbindungen“ zu reduzieren. Der RDS-Proxy ist derzeit als öffentliche Vorversion für PostgreSQL verfügbar. Weitere Informationen finden Sie unter Verwalten von Verbindungen mit Amazon-RDS-Proxy (Vorversion).	8. April 2020
Globale Aurora-Datenbanken unterstützen jetzt Aurora PostgreSQL	Sie können jetzt globale Aurora-Datenbanken für das PostgreSQL-Datenbank-Engine erstellen. Eine globale Aurora-Datenbank erstreckt sich über mehrere AWS-Regionen und ermöglicht globale Lesevorgänge mit geringer Latenz und Notfallwiederherstellung nach regionalen Ausfällen. Weitere Informationen finden Sie unter Arbeiten mit Amazon Aurora Global Database.	10. März 2020

Unterstützung für Upgrades von Hauptversionen für Aurora PostgreSQL	Mit Aurora PostgreSQL können Sie nun die DB-Engine auf eine Hauptversion aktualisieren. Auf diese Weise können Sie zu einer neueren Hauptversion springen, wenn Sie ausgewählte PostgreSQL-Engine-Versionen aktualisieren. Weitere Informationen finden Sie unter Aktualisieren der PostgreSQL-DB-Engine für Aurora PostgreSQL.	4. März 2020
Aurora PostgreSQL unterstützt die Kerberos-Authentifizierung	Sie können jetzt die Kerberos-Authentifizierung verwenden, um Benutzer zu authentifizieren, wenn sie sich mit Ihren Aurora PostgreSQL-DB-Clustern verbinden. Weitere Informationen finden Sie unter Verwenden der Kerberos-Authentifizierung mit Aurora PostgreSQL.	28. Februar 2020
Die Daten-API unterstützt AWS PrivateLink	Die Daten-API unterstützt jetzt die Erstellung von Amazon VPC-Endpunkten für Daten-API-Aufrufe, um den Verkehr zwischen Anwendungen und der Daten-API im AWS Netzwerk aufrechtzuerhalten. Weitere Informationen finden Sie unter Erstellen eines Amazon-VPC-Endpunkts (AWS PrivateLink) für die Daten-API.	6. Februar 2020

Unterstützung von Aurora Machine Learning in Aurora PostgreSQL	Die `aws_ml` Aurora PostgreSQL-Erweiterung bietet Funktionen, die Sie in Ihren Datenbankabfragen verwenden, um Amazon Comprehend zur Stimmungsanalyse aufzurufen, und SageMaker KI, um Ihre eigenen Machine-Learning-Modelle auszuführen. Weitere Informationen finden Sie unter Verwenden von Machine Learning (ML)-Funktionen mit Aurora.	5. Februar 2020
Aurora PostgreSQL unterstützt das Exportieren von Daten nach Amazon S3	Sie können Daten aus einem Aurora PostgreSQL-DB-Cluster abfragen und direkt in Dateien exportieren, die in einem Amazon S3-Bucket gespeichert sind. Weitere Informationen finden Sie unter Exportieren von Daten aus einem Aurora PostgreSQL-DB-Cluster zu Amazon S3.	5. Februar 2020
Unterstützung für den Export von DB-Snapshot-Daten nach Amazon S3	Amazon Aurora unterstützt den Export von DB-Snapshot-Daten nach Amazon S3 for MySQL und PostgreSQL. Weitere Informationen finden Sie unter Exportieren von DB-Snapshot-Daten zu Amazon S3.	9. Januar 2020

Aurora MySQL-Versionshinweise im Dokumentverlauf	Dieser Abschnitt enthält nun Verlaufseinträge für Aurora MySQL-kompatible Edition-Versionshinweise für Versionen, die nach dem 31. August 2018 veröffentlicht wurden. Wählen Sie den Link in der ersten Spalte des Verlaufseintrags, um die vollständigen Versionshinweise für eine spezifische Version anzuzeigen.	13. Dezember 2019
Amazon-RDS-Proxy	Sie können den Overhead für die Verbindungsverwaltung in Ihrem Cluster reduzieren und die Wahrscheinlichkeit des Fehlers "zu viele Verbindungen" reduzieren, indem Sie Amazon-RDS-Proxy verwenden. Sie ordnen jeden Proxy einer RDS DB-Instance oder einem Aurora DB-Cluster zu. Dann verwenden Sie den Proxy-Endpunkt in der Verbindungszeichenfolge für Ihre Anwendung. Der Amazon-RDS-Proxy ist derzeit als öffentliche Vorversion verfügbar. Die Aurora MySQL-Datenbank-Engine wird unterstützt. Weitere Informationen finden Sie unter Verwalten von Verbindungen mit Amazon-RDS-Proxy (Vorversion).	03. Dezember 2019

Data API for Aurora Serverless v1 unterstützt Datentyp-Mapping-Hinweise	Sie können nun einen Hinweis verwenden, damit die Data API for Aurora Serverless v1 einen `String`-Wert als einen anderen Typ zur Datenbank sendet. Weitere Informationen finden Sie unter Aufrufen der Daten-API.	26. November 2019
Data API for Aurora Serverless v1 unterstützt eine Java-Clientbibliothek (Vorversion)	Sie können eine Java Client-Bibliothek für Data API herunterladen und verwenden. Mithilfe dieser Bibliothek können Sie Ihre clientseitigen Klassen zu Anforderungen und Antworten der Daten-API zuweisen. Weitere Informationen finden Sie unter Verwenden der Java Client-Bibliothek für die Daten-API.	26. November 2019
Aurora PostgreSQL ist mit FedRAMP HIGH konform	Aurora PostgreSQL ist mit FedRAMP HIGH konform. Einzelheiten zu den Compliance-Maßnahmen AWS und den Bemühungen zur Einhaltung der Vorschriften finden Sie unter AWS Services im Umfang der einzelnen Compliance-Programme.	26. November 2019

Isolationsstufe READ COMMITTED für Amazon Aurora MySQL-Replikate aktiviert	Sie können jetzt die Isolationsstufe `READ COMMITTED` für Aurora MySQL-Replikate aktivieren. Hierzu müssen Sie die Konfigurationseinstellung `aurora_read_replica_read_committed_isolation_enabled` auf Sitzungsebene aktivieren. Die Verwendung der Isolationsstufe `READ COMMITTED` für langfristig ausgeführte Abfragen auf OLTP-Clustern kann helfen, Probleme mit der Länge der Verlaufsliste zu lösen. Vor der Aktivierung dieser Einstellung müssen Sie verstehen, wie sich das Isolationsverhalten auf Aurora-Replikaten von der gewöhnlichen MySQL-Implementierung von unterscheide `READ COMMITTED` . Weitere Informationen finden Sie unter Aurora MySQL-Isolationsstufen.	25. November 2019
Performance Insights unterstützt die Analyse von Statistiken zu laufenden Aurora PostgreSQL-Abfragen	Sie können jetzt Statistiken laufender Abfragen mit Performance Insights für Aurora PostgreSQL-DB-Instances analysieren. Weitere Informationen finden Sie unter Analysieren von Statistiken zu laufenden Abfragen.	25. November 2019

Mehr Cluster in einer globalen Aurora-Datenbank	Sie können jetzt einer globalen Aurora-Datenbank mehrere sekundäre Regionen hinzufügen. Sie können globale Lesevorgänge mit niedriger Latenz und Notfallwiederherstellungen in einem größeren geografischen Bereich nutzen. Weitere Informationen zu globalen Aurora-Datenbanken finden Sie unter Arbeiten mit Amazon Aurora Global Databases.	25. November 2019
Unterstützung von Aurora Machine Learning in Aurora MySQL	In Aurora MySQL 2.07 und höher können Sie Amazon Comprehend für Stimmungsanalysen und SageMaker KI für eine Vielzahl von Algorithmen für maschinelles Lernen aufrufen. Sie können die Ergebnisse direkt in Ihrer Datenbankanwendung verwenden, indem Sie Aufrufe an gespeicherte Funktionen in Ihre Abfragen einbetten. Weitere Informationen finden Sie unter Verwenden von Machine Learning (ML)-Funktionen mit Aurora.	25. November 2019

Die globale Aurora-Datenbank erfordert keine Engine-Modus-Einstellung mehr	Sie müssen nicht mehr `--engine-mode=global` angeben, wenn Sie ein Cluster erstellen, das Teil einer globalen Aurora-Datenbank sein soll. Alle Aurora-Cluster, die die Kompatibilitätsanforderungen erfüllen, können Teil einer globalen Datenbank sein. Der Cluster muss beispielsweise zurzeit Aurora MySQL Version 1 mit MySQL 5.6-Kompatibilität verwenden. Weitere Informationen zu globalen Aurora-Datenbanken finden Sie unter Arbeiten mit globalen Amazon Aurora-Datenbanken.	25. November 2019
Die globale Aurora-Datenbank ist für Aurora MySQL Version 2 verfügbar	Ab Aurora MySQL 2.07 können Sie eine globale Aurora-Datenbank erstellen, die mit MySQL 5.7 kompatibel ist. Sie müssen den Engine-Modus `global` für die primären oder sekundären Cluster nicht angeben. Sie können jedes neue bereitgestellte Cluster mit Aurora MySQL 2.07 oder höher zu einer Aurora Global Database hinzufügen. Weitere Informationen zu Aurora Global Database finden Sie unter Arbeiten mit Amazon Aurora Global Database.	25. November 2019

Aurora MySQL Hot-Row-Contention-Optimierung ohne Labor-Modus verfügbar	Die Hot Row Contention-Optimierung ist jetzt allgemein für Aurora MySQL verfügbar und erfordert kein Festlegen des Aurora-Labor-Modus auf „ON (EIN)“. Diese Funktion sorgt für eine deutliche Verbesserung des Durchsatzes bei Workloads mit zahlreichen Transaktionen, die um Zeilen auf derselben Seite konkurrieren. Diese Verbesserung umfasst die Veränderung des von Aurora MySQL verwendeten Entsperrungsalgorithmus.	19. November 2019
Aurora MySQL Hash Joins ohne Labor-Modus verfügbar	Die Hash Join-Funktion ist jetzt allgemein für Aurora MySQL verfügbar und erfordert kein Festlegen des Aurora-Labor-Modus-auf „ON (EIN)“. Diese Funktion kann die Abfrageleistung verbessern, wenn Sie eine große Datenmenge mithilfe eines Equijoins verbinden müssen. Weitere Informationen zum Verwenden dieser Funktion finden Sie unter Arbeiten mit Hash Joins in Aurora MySQL.	19. November 2019

Aurora MySQL 2.*-Unterstützung für weitere db.r5-Instance-Klassen	Aurora MySQL-Cluster unterstützen jetzt die Instance-Typen db.r5.8xlarge, db.r5.16xlarge und db.r5.24xlarge. Weitere Informationen zu Instance-Typen für Aurora MySQL-Cluster finden Sie unter Auswahl der DB-Instance-Klasse.	19. November 2019
Aurora MySQL 2.*-Unterstützung für die Rückverfolgung	Aurora MySQL 2.*-Versionen unterstützen jetzt die schnelle Wiederherstellung nach Benutzerfehlern wie Löschen der falschen Tabelle oder Löschen der falschen Zeile. Mithilfe de Rückverfolgung können Sie Ihre Datenbank zu einem früheren Zeitpunkt verschieben, ohne dass eine Wiederherstellung aus einem Backup erforderlich ist. Der Vorgang ist selbst bei großen Datenbanken innerhalb weniger Sekunden abgeschlossen. Einzelheiten finden Sie unter Rückverfolgen eines Aurora-DB-Clusters.	19. November 2019

Unterstützung von Abrechnungstags für Aurora	Sie können nun Tags verwenden, um die Kostenverrechnung für Ressourcen wie Aurora-Cluster, DB-Instances innerhalb von Aurora-Clustern, I/O, Backups, Snapshots usw. zu verfolgen. Mit dem Cost Explorer können Sie sich die mit jedem Tag verbundenen AWS Kosten anzeigen lassen. Weitere Informationen zur Verwendung von Tags mit Aurora finden Sie unter Tagging von Amazon-RDS-Ressourcen. Allgemeine Informationen zu Tags und deren Verwendung für die Kostenanalyse finden Sie unter Verwendung von Kostenzuordnungs-Tags und Benutzerdefinierte Kostenzuordnungs-Tags.	23. Oktober 2019
Daten-API für Aurora PostgreSQL	Aurora PostgreSQL unterstützt jetzt die Verwendung der Data API mit Amazon Aurora Serverless v1-DB-Clustern. Weitere Informationen finden Sie unter Verwenden der Data API for Aurora Serverless v1.	23. September 2019

Aurora PostgreSQL unterstützt das Hochladen von Datenbankprotokollen in Protokolle CloudWatch	Sie können Ihren Aurora PostgreSQL-DB-Cluster so konfigurieren, dass Protokolldaten in einer Protokollgruppe in Amazon CloudWatch Logs veröffentlicht werden. Mit CloudWatch Logs können Sie eine Echtzeitanalyse der Protokolldaten durchführen und diese CloudWatch zum Erstellen von Alarmen und zum Anzeigen von Metriken verwenden. Sie können CloudWatch Logs verwenden, um Ihre Protokolldatensätze in einem äußerst langlebigen Speicher zu speichern. Weitere Informationen finden Sie unter Veröffentlichen von Aurora PostgreSQL-Protokollen in Amazon CloudWatch Logs.	9. August 2019
Multi-Master-Cluster für Aurora MySQL	Sie können Aurora MySQL Multi-Master-Cluster einrichten. In diesen Clustern verfügt jede DB-Instance über Lese- und Schreibfähigkeit. Weitere Informationen finden Sie unter Arbeiten mit Aurora-Multi-Master-Clustern.	8. August 2019

Aurora PostgreSQL unterstützt Aurora Serverless v1	Sie können jetzt Amazon Aurora Serverless v1 mit Aurora PostgreSQL verwenden. Ein Aurora Serverless DB-Cluster kann abhängig von Ihren Anwendungsanforderungen automatisch starten, herunterfahren und die Kapazität steigern oder senken. Weitere Informationen finden Sie unter Verwendung von Amazon Aurora Serverless v1.	9. Juli 2019
Kontoübergreifendes Klonen für Aurora MySQL	Sie können jetzt das Cluster-Volume für einen Aurora MySQL-DB-Cluster zwischen AWS Konten klonen. Sie autorisieren die gemeinsame Nutzung über AWS Resource Access Manager (AWS RAM). Das geklonte Cluster-Volume verwendet einen copy-on-write Mechanismus, der nur zusätzlichen Speicherplatz für neue oder geänderte Daten benötigt. Weitere Informationen zum Klonen für Aurora finden Sie unter Klonen von Datenbanken in einem Aurora-DB-Cluster.	2. Juli 2019

Aurora PostgreSQL unterstützt db.t3-DB-Instance-Klassen	Sie können jetzt Aurora PostgreSQL-DB-Cluster erstellen, die DB-Instance-Klassen vom Typ "db.t3" verwenden. Weitere Informationen finden Sie unter DB-Instance-Klasse.	20. Juni 2019
Unterstützung für den Import von Daten aus Amazon S3 for Aurora PostgreSQL	Sie können nun Daten aus einer Amazon S3-Datei in eine Tabelle in einem Aurora PostgreSQL DB-Cluster importieren. Weitere Informationen finden Sie unter Importieren von Amazon S3-Daten in einen Aurora PostgreSQL DB-Cluster.	19. Juni 2019
Aurora PostgreSQL jetzt mit schneller Failover-Wiederherstellung und Cluster-Cacheverwaltung	Aurora PostgreSQL bietet jetzt eine Cluster-Cacheverwaltung, um im Falle eines Failovers eine schnelle Wiederherstellung der primären DB-Instance sicherzustellen. Weitere Informationen finden Sie unter Schnelle Wiederherstellung nach Failover mit Cluster-Cacheverwaltung.	11. Juni 2019
Data API for Aurora Serverless v1 allgemein verfügbar	Sie können mit Webservice-basierten Anwendungen über die Data API auf Aurora Serverless v1-Cluster zugreifen. Weitere Informationen finden Sie unter Verwenden der Data API for Aurora Serverless v1.	30. Mai 2019

Aurora PostgreSQL unterstützt die Datenbanküberwachung mit Datenbankaktivitäts-Streams	Aurora PostgreSQL enthält jetzt Datenbankaktivitätsstreams, die einen near-real-time Datenstrom der Datenbankaktivität in Ihrer relationalen Datenbank bereitstellen. Weitere Informationen finden Sie unter Verwenden von Datenbankaktivitäts-Streams.	30. Mai 2019
Amazon-Aurora-Empfehlungen	Amazon Aurora bietet nun automatisierte Empfehlungen für Aurora-Ressourcen. Weitere Informationen finden Sie unter Verwenden von Amazon Aurora-Empfehlungen.	22. Mai 2019
Performance-Insights-Support für Aurora Global Database	Sie können Performance Insights jetzt mit Aurora Global Database nutzen. Weitere Informationen über Performance-Insights für Aurora finden Sie unter Verwenden von Amazon RDS Performance Insights. Weitere Informationen zu Aurora Global Databases finden Sie unter Arbeiten mit Aurora Global Database.	13. Mai 2019

Performance Insights ist für Aurora MySQL 5.7 verfügbar	Amazon RDS Performance Insights ist jetzt für Aurora MySQL-Versionen 2.x verfügbar, die mit MySQL 5.7 kompatibel sind. Weitere Informationen finden Sie unter Verwenden von Amazon-RDS-Performance-Insights.	3. Mai 2019
Die globalen Aurora-Datenbanken sind in weiteren Sprachen verfügbar AWS-Regionen	Sie können jetzt globale Aurora-Datenbanken in den meisten Ländern erstellen, in AWS-Regionen denen Aurora verfügbar ist. Weitere Informationen zu globalen Aurora-Datenbanken finden Sie unter Arbeiten mit globalen Amazon Aurora-Datenbanken.	30. April 2019
Mindestkapazität 1 für Aurora Serverless v1	Für einen Aurora Serverless v1-Cluster müssen Sie eine Mindestkapazitätseinstellung von 1 verwenden. Zuvor betrug die Mindestkapazität 2. Weitere Informationen zum Angeben von Aurora-Serverless-Kapazitätswerten finden Sie unter Festlegen der Kapazität eines Aurora Serverless v1-Serverless-DB-Clusters.	29. April 2019

Aurora Serverless v1-Timeout-Aktion	Sie können jetzt die Aktion angeben, die ausgeführt werden soll, wenn für eine Aurora Serverless v1-Kapazitätsänderung ein Timeout eintritt. Weitere Informationen finden Sie unter Timeout-Aktion für Kapazitätsänderungen.	29. April 2019
Sekundengenaue Abrechnung	Amazon RDS wird jetzt in allen On-Demand-Instances in 1-Sekunden-Schritten abgerechnet, AWS-Regionen mit Ausnahme von AWS GovCloud (USA). Weitere Informationen finden Sie unter Abrechnung von DB-Instances für Aurora.	25. April 2019
Schnappschüsse teilen Aurora Serverless v1AWS-Regionen	Im Fall von Aurora Serverless v1 werden Snapshots stets verschlüsselt. Wenn Sie den Snapshot mit Ihrem eigenen verschlüsseln AWS KMS key, können Sie den Snapshot jetzt kopieren oder teilen. AWS-Regionen Informationen zu Snapshots von Aurora Serverless v1-DB-Clustern finden Sie unter Aurora Serverless v1 und Snapshots.	17. April 2019

Wiederherstellen von MySQL 5.7-Backups aus Amazon S3	Sie können jetzt ein Backup Ihrer MySQL Version 5.7-Datenbank erstellen und auf Amazon S3 speichern und die Sicherungsdatei anschließend auf einem neuen Aurora MySQL DB-Cluster wiederherstellen. Weitere Informationen finden Sie unter Migrieren von Daten aus einer externen MySQL-Datenbank in einen Aurora MySQL-DB-Cluster.	17. April 2019
Teilen von Aurora Serverless v1-Snapshots über Regionen hinweg	Im Fall von Aurora Serverless v1 werden Snapshots stets verschlüsselt. Wenn Sie den Snapshot mit Ihrem eigenen verschlüsseln AWS KMS key, können Sie den Snapshot jetzt regionsübergreifend kopieren oder teilen. Informationen zu Snapshots von Aurora Serverless v1-DB-Clustern finden Sie unter Aurora Serverless und Snapshots.	16. April 2019
proof-of-conceptAurora-Anleitung	Sie können lernen, wie ein Machbarkeitsnachweis durchgeführt wird, um Ihre Anwendung und Workload mit Aurora auszuprobieren. Ein komplettes Tutorial finden Sie unter Durchführen eines Aurora-Machbarkeitsnachweises.	16. April 2019

Aurora Serverless v1 unterstützt das Wiederherstellen aus einem Amazon S3-Backup	Sie können jetzt Backups von Amazon S3 in einen Aurora Serverless-Cluster importieren. Einzelheiten zu diesem Verfahren finden Sie unter Migrieren von Daten aus MySQL mithilfe eines Amazon S3-Buckets.	16. April 2019
Neue modifizierbare Parameter für Aurora Serverless v1	Sie können jetzt die folgenden DB-Parameter für einen Aurora Serverless v1-Cluster modifizieren: `innodb_file_format` , `innodb_file_per_table` , `innodb_large_prefix` , `innodb_lock_wait_timeout` , `innodb_monitor_disable` , `innodb_monitor_enable` , `innodb_monitor_reset` , `innodb_monitor_reset_all` , `innodb_print_all_deadlocks` , `log_warnings` , `net_read_timeout` , `net_retry_count` , `net_write_timeout` , `sql_mode` und `tx_isolation` . Weitere Informationen zu Konfigurationsparametern für Aurora Serverless v1-Cluster finden Sie unter Aurora Serverless v1 und Parametergruppen.	4. April 2019

Aurora PostgreSQL unterstützt db.r5-DB-Instance-Klassen	Sie können jetzt Aurora PostgreSQL-DB-Cluster erstellen, die DB-Instance-Klassen vom Typ "db.r5" verwenden. Weitere Informationen finden Sie unter DB-Instance-Klasse.	4. April 2019
Logische Aurora PostgreSQL-Replikation	Sie können nun die logische PostgreSQL-Replikation nutzen, um Teile einer Datenbank für einen Aurora PostgreSQL-DB-Cluster zu replizieren. Weitere Informationen finden Sie unter Verwenden der logischen Replikation von PostgreSQL.	28. März 2019

GTID-Unterstützung für Aurora MySQL 2.04	Sie können die Replikation nun mit der Funktion der globalen Transaktions-ID (GTID) von MySQL 5.7 verwenden. Diese Funktion vereinfacht das Durchführen der Binärprotokoll-(binlog)Replikation zwischen Aurora MySQL und einer externen MySQL 5.7-kompatiblen Datenbank. Die Replikation kann den Aurora MySQL-Cluster als Quelle oder als Ziel verwenden. Diese Funktion ist für Aurora MySQL 2.04 und höher verfügbar. Weitere Informationen über GTID-basierte Replikation und Aurora MySQL finden Sie unter Verwenden der GTID-basierten Replikation für Aurora MySQL.	25. März 2019

Aurora Serverless v1Logs auf Amazon hochladen CloudWatch	Sie können Aurora jetzt Datenbankprotokolle CloudWatch für einen Aurora Serverless v1 Cluster hochladen lassen. Weitere Informationen finden Sie unter Anzeigen von Aurora Serverless-DB-Clustern. Als Teil dieser Erweiterung können Sie nun Werte für Parameter auf Instance-Ebene in einer DB-Clusterparamete rgruppe definieren und diese Werte auf alle DB-Instances im Cluster anwenden, sofern sie nicht in der DB-Parame tergruppe überschrieben werden. Weitere Informationen finden Sie unter Arbeiten mit DB-Parametergruppen und DB-Clusterparametergruppen.	25. Februar 2019
Aurora MySQL unterstützt db.t3-DB-Instance-Klassen	Sie können jetzt Aurora MySQL-DB-Cluster erstellen , die DB-Instance-Klassen vom Typ "db.t3" verwenden. Weitere Informationen finden Sie unter DB-Instance-Klasse.	25. Februar 2019
Aurora MySQL unterstützt db.r5-DB-Instance-Klassen	Sie können jetzt Aurora MySQL-DB-Cluster erstellen , die DB-Instance-Klassen vom Typ "db.r5" verwenden. Weitere Informationen finden Sie unter DB-Instance-Klasse.	25. Februar 2019

Performance-Insights-Zähler für Aurora MySQL	Sie können nun Leistungszähler zu Ihren Performance-Insights-Diagrammen für Aurora MySQL-DB-Instances hinzufügen. Weitere Informationen finden Sie unter Verwenden der Performance-Insights-Dashboard-Komponenten .	19. Februar 2019
Amazon RDS Performance Insights unterstützt Anzeige von mehr SQL-Text für Aurora MySQL	Amazon RDS Performance Insights unterstützt jetzt die Anzeige von mehr SQL-Text im Performance-Insights-Dashboard für Aurora MySQL-DB-Instances. Weitere Informationen finden Sie unter Anzeigen von mehr SQL-Text im Performance-Insights-Dashboard.	6. Februar 2019
Amazon RDS Performance Insights unterstützt Anzeige von mehr SQL-Text für Aurora PostgreSQL	Amazon RDS Performance Insights unterstützt jetzt die Anzeige von mehr SQL-Text im Performance-Insights-Dashboard für Aurora PostgreSQL-DB-Instances. Weitere Informationen finden Sie unter Anzeigen von mehr SQL-Text im Performance-Insights-Dashboard.	24. Januar 2019

Aurora-Sicherungsfakturierung	Sie können die CloudWatch Amazon-Metriken `TotalBackupStorageBilled` , und verwenden `SnapshotStorageUsed` , `BackupRetentionPeriodStorageUsed` um die Speicherbelegung Ihrer Aurora-Backups zu überwachen. Weitere Informationen zur Verwendung von CloudWatch Metriken finden Sie unter Überblick über die Überwachung. Weitere Informationen über die Verwaltung von Sicherungsdaten finden Sie unter Grundlegendes zur Aurora-Sicherungsspeichernutzung.	3. Januar 2019
Performance-Insights-Zähler	Sie können nun Leistungsindikatoren zu Ihren Performance-Insights-Diagrammen hinzufügen. Weitere Informationen finden Sie unter Verwenden der Performance-Insights-Dashboard-Komponenten .	6. Dezember 2018

Globale Aurora-Datenbank	Sie können jetzt globale Aurora-Datenbanken erstellen . Eine globale Aurora-Da tenbank erstreckt sich über mehrere AWS-Regionen und ermöglicht globale Lesevorgä nge mit geringer Latenz und Notfallwiederherstellung nach regionalen Ausfällen. Weitere Informationen finden Sie unter Arbeiten mit Amazon Aurora Global Database.	28. November 2018
Abfrageplanverwaltung in Aurora PostgreSQL	Aurora PostgreSQL bietet nun eine Abfrageplanverwalt ung, mit der Sie PostgreSQ L-Abfrageausführungspläne verwalten können. Weitere Informationen finden Sie unter Verwalten von Abfrageau sführungsplänen für Aurora PostgreSQL.	20. November 2018
Abfrage-Editor für Aurora Serverless v1 (Beta)	Sie können SQL-Anweisungen in der Amazon-RDS-Konsole auf Aurora Serverless v1-Clustern ausführen. Weitere Informationen finden Sie unter Verwenden des Abfrage-E ditors für Aurora Serverless v1.	20. November 2018

Data API für Aurora Serverless v1 (Beta)	Sie können mit Webservic e-basierten Anwendung en über die Data API auf Aurora Serverless v1-Cluste r zugreifen. Weitere Informati onen finden Sie unter Using the Data API for Aurora Serverless.	20. November 2018
TLS-Unterstützung für Aurora Serverless v1	Aurora Serverless v1Cluster unterstützen Verschlüsselung. TLS/SSL Weitere Informati onen finden Sie unter TLS/ SSL für Aurora Serverless.	19. November 2018
Benutzerdefinierte Endpunkte	Sie können nun Endpunkte erstellen, die einer beliebige n Menge von DB-Instan ces zugeordnet sind. Diese Funktion hilft beim Lastausgl eich und der Hochverfü gbarkeit für Aurora-Cl uster, bei denen einige DB-Instances eine andere Kapazität oder Konfigura tion haben als andere. Sie können benutzerdefinierte Endpunkte verwenden, anstatt eine Verbindung zu einer bestimmten DB-Instance über deren Instance-Endpunkt herzustellen. Weitere Informati onen finden Sie unter Amazon Aurora-Endpunkte.	12. November 2018

Unterstützung der IAM-Authentifizierung in Aurora PostgreSQL	Aurora PostgreSQL unterstützt nun die IAM-Authentifizierung. Weitere Informationen finden Sie unter IAM-Datenbankauthentifizierung.	8. November 2018
Benutzerdefinierte Parametergruppen für die Wiederherstellung und zeitpunktbezogene Wiederherstellung	Sie können jetzt bei einer Snapshot-Wiederherstellung oder einer zeitpunktbezogenen Wiederherstellung eine benutzerdefinierte Parametergruppe angeben. Weitere Informationen finden Sie unter Wiederherstellen aus einem DB-Cluster-Snapshot und Wiederherstellen eines DB-Clusters zu einer bestimmten Zeit.	15. Oktober 2018
Löschschutz für Aurora-DB-Cluster	Wenn Sie für einen DB-Cluster Löschschutz aktivieren, kann die Datenbank von keinem Benutzer gelöscht werden. Weitere Informationen finden Sie unter Löschen eines DB-Clusters.	26. September 2018
Stoppen/Starten-Funktion für Aurora	Sie können jetzt mit einer einzelnen Operation einen kompletten Aurora-Cluster stoppen oder starten. Weitere Informationen finden Sie im Abschnitt zum Stoppen und Starten eines Aurora-Clusters.	24. September 2018

Parallelabfrage-Funktion für Aurora MySQL	Aurora MySQL bietet jetzt eine Option zur Parallelisierung der I/O Arbeit für Abfragen in der gesamten Aurora-Speicherinfrastruktur. Mit dieser Funktion können datenintensive analytische Abfragen, die häufig die zeitaufwendigsten Vorgänge in einer Workload darstellen, beschleunigt werden. Weitere Informationen finden Sie unter Parallele Abfrage für Aurora MySQL.	20. September 2018
Neues Handbuch	Dies ist die erste Version des Amazon Aurora-Benutzerhandbuchs.	31. August 2018

AWS Glossar

Die neueste AWS Terminologie finden Sie im AWS Glossar in der AWS-Glossar Referenz.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.